UNIVERSIDAD DE GRANADA

MH

Metaheurísticas

Curso: 3º **Grupo:** 1

Práctica 3
MDP

Autor: Mario Carmona Segovia
DNI: 45922466E **E-mail:** mcs2000carmona@correo.ugr.es

Profesor: Daniel Molina

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Curso 2020 - 2021

Índice

Índice de figuras

Índice de tablas

1. Descripción del problema

El Problema de la Máxima Diversidad (en ingles, Maximum Diversity Problem, MDP) consiste en la elección de un subconjunto de M puntos, de un conjunto mayor de N puntos. Este subconjunto debe maximizar la diversidad, Z; es decir, maximizar la distancia entre los elementos del subconjunto.

La fórmula para obtener la diversidad de un subconjunto es la siguiente:

$$Z = \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} d_{ij}$$

Siendo:

- i y j índices de los elementos del conjunto
- d_{ij} la distancia entre los puntos con índice i y j

2. Descripción de los aspectos comunes

2.1. Representación entera

2.1.1. Representación de las soluciones

Cada elemento del conjunto total se ha etiquetado con un número entero positivo, empezando por el cero.

Como una solución a este problema es un subconjunto de este conjunto total, he representado el subconjunto como un lista de enteros.

2.1.2. Función objetivo

La función objetivo valora como de buena es una solución. La calidad se mide como la suma total de todas las distancias entre los elementos que forman la solución. Una solución es de mayor calidad cuanto mayor es la distancia total, ya que a mayor distancia mayor diversidad en la solución.

Algorithm 1 Función objetivo

Input: *solucion*: lista con la etiquetas de los elementos que forman la solución.
Input: *distancias*: matriz que guarda la distancia entre cada par de elementos del conjunto total.
Output: *distancia_total*: es la distancia total que existe en la *solucion*.

```
1:
2: distancia_total ← 0.0
3: /* Calculamos la suma de todas la distancias entre los elementos
4:  de la solución */
5: solucion_size ← número de elementos en la solucion
6: for i ∈ {0, ..., solucion_size − 1} do
7:     for j ∈ {i + 1, ..., solucion_size} do
8:         distancia_total ← sumar distancia entre i y j
9: return distancia_total
```

Además de esta versión de la función objetivo se ha creado otra función objetivo que además de calcular la distancia total de una solución, calcula la contribución de cada elemento en la distancia total, es decir, calcula la distancia total de cierto elemento con el resto de elementos que conforman la solución. La distancia entre el elemento x e y será considerada tanto para la contribución de x como de y, de esta manera se aprovechan las operaciones realizadas para calcular la función objetivo.

Algorithm 2 Función objetivo (extendida)

Input: *solucion*: lista con la etiquetas de los elementos que forman la solución.
Input: *distancias*: matriz que guarda la distancia entre cada par de elementos del conjunto total.
Output: *distancia_total*: es la distancia total que existe en la *solucion*.
Output: *contribuciones*: vector de flotantes que indican la contribución de cada elemento a la *distancia_total*.

```
2: distancia_total ← 0.0
   /* Calculamos la suma de todas la distancias entre los elementos
4:  de la solución */
   solucion_size ← número de elementos en la solucion
6: for i ∈ {0, ..., solucion_size − 1} do
       for j ∈ {i + 1, ..., solucion_size} do
8:         distancia_total ← sumar distancia entre i y j
           contribuciones ← sumar contribución al elemento i
10:        contribuciones ← sumar contribución al elemento j
   return distancia_total, contribuciones
```

2.2. Lectura de datos

Los datos utilizados para el problema están guardados en un archivo de texto plano. Este archivo contiene los siguiente datos:

- Nº de elementos del conjunto
- Nº de elementos que debe tener la solución
- Distancia entre cada par de elementos del conjunto

El formato de la información es el siguiente:

```
1 {Número de elementos del conjunto} {Número de elementos que debe tener la solución}
2 {Elemento 0} {Elemento 1} {Distancia entre elemento 0 y elemento 1}
3 {Elemento 0} {Elemento 2} {Distancia entre elemento 0 y elemento 2}
4 ...
```

Para obtener esta información y colocarla en una estructura de datos más manejable para el programa, he creado la siguiente función:

2.3. Generación de soluciones aleatorias

He creado una función que se encarga de generar una lista de enteros aleatoria, que representa una solución. Esta función ha sido usada tanto en la BMB como en la ILS.

Algorithm 3 Leer archivo

Input: *nombre_archivo*: nombre del archivo de texto plano que contiene la información.
Output: *num_elem_selec*: número de elementos que debe tener la solución al problema.
Output: *distancias*: matriz de distancias entre pares de elementos del conjunto.

```
archivo ← abrir el archivo con nombre "nombre_archivo"
numElemTotal ← obtener el número de elementos del conjunto completo
num_elem_selec ← obtener el número de elemento que debe tener la solución
distancias ← crear la matriz de distancias
while !final_archivo do
    i, j ← obtener el par de elementos
    distancias ← añadir distancia entre el par de elementos
return num_elem_selec, distancias
```

Algorithm 4 Generación aleatoria de una solución

Input: n: número de elementos que hay en el problema.
Input: m: número de elementos que debe tener la solución al problema.
Output: *solucion*: lista con las etiquetas de los elementos que forman la solución actual.
Output: *no_seleccionados*: lista con las etiquetas de los elementos que no forman parte de la solución.

```
/* Generar la solución aleatoria */
tamSolucion ← 0
solucion ← Vacío
while tamSolucion ¡m do
    nuevoElem ← Número aleatorio entre 0 y n − 1
    solucion ← Añadir el nuevoElem a la solución
    solucion ← Eliminar los elementos repetidos
    tamSolucion ← Número de elementos actualmente en la solución
/* Generar los elementos no seleccionados a partir de la solución generada */
no_seleccionados ← Vacío
for i ∈ {0, ..., n − 1} do
    encontrado ← Comprobarsielelementoiestáenlasolución
    /* Si el elemento no está en la solución */
    if !encontrado then
        no_seleccionados ← Añadir el elemento i a los elementos no seleccionados
return solucion, no_seleccionados
```

3. Descripción de la Búsqueda Local

3.1. Funciones comunes

3.1.1. Operador de generación de vecino

Como operador de generación de vecino se ha utilizado el intercambio, que consiste en elegir un elemento de la solución y otro elemento no seleccionado para la solución, e intercambiarlos de listas. Su implementación en pseudocódigo es la siguiente:

Algorithm 5 Intercambio

Input: *solucion_actual*: lista con las etiquetas de los elementos que forman la solución actual.
Input: *elemento_a_sustituir*: índice del elemento que se quiere sustituir.
Input: *no_seleccionados*: lista con las etiquetas de los elementos que no forman parte de la solución.
Input: *elemento_a_incluir*: índice del elemento que se quiere incluir en la solución.
Input: *distancias*: matriz que guarda la distancia entre cada par de elementos del conjunto total.
Input: *contribuciones_actual*: vector de flotantes que contiene las contribuciones de cada elemento de la solución actual.
Output: *nueva_solucion*: lista con las etiquetas de los elementos que forman la nueva solución.
Output: *nuevas_contribuciones*: vector de flotantes que contiene las contribuciones de cada elemento de la nueva solución.

```
contri_elem_a_incluir ← 0
nuevas_contribuciones ←contribuciones_actual
/* Modificar las contribuciones de los elementos que no son
intercambiados */
for i ∈ solucion_actual do
    if i != elemento_a_sustituir then
        nuevas_contribuciones ← restar distancia del elemento i al elemento_a_sustituir
        nuevas_contribuciones ← sumar distancia del elemento i al elemento_a_incluir
        contri_elem_a_incluir ← añadir la distancia sumada a la contribución del nuevo elemento
nueva_solucion ← intercambiamos el nuevo elemento por el elemento elegido para ser sustituido
nuevas_contribuciones ← intercambiar también las contribuciones de los elementos intercambiados
return nueva_solucion, nuevas_contribuciones
```

3.1.2. Factorización de la BL

Dado que que la creación de un nuevo vecino sólo modifica un elemento de la solución original, por lo que el coste de la solución sólo es alterado por las distancias de ambos elementos involucrados en el operador de intercambio, parece mejor opción realizar una factorización de la función objetivo; es decir, en vez de volver a calcular todas las distancias de la nueva solución, sólo tenemos que restar las distancias del elemento sustituido al resto de elementos de la solución y

sumar las distancias del elemento incluido al resto de elementos de la solución. Su implementación en pseudocódigo es la siguiente:

Algorithm 6 Función objetivo factorizada

Input: *solucion*: lista con las etiquetas de los elementos que forman la solución.
Input: *elemento_sustituido*: índice del elemento que se ha sustituido.
Input: *elemento_incluido*: índice del elemento que se ha incluido en la solución.
Input: *distancias*: matriz que guarda la distancia entre cada par de elementos del conjunto total.
Input: *coste_actual*: suma de la distancia total de la solución actual.
Output: *nuevo_coste*: suma de la distancia total de la solución modificada.
Output: *mejorado*: variable booleana que indica si ha mejorado el coste de la función.
Output: *elemento_incluido*: índice del elemento que se ha incluido en la solución.

```
nuevo_coste ←coste_actual
/* Modificar el coste de la nueva solución */
for i ∈ solucion do
    if i != elemento_sustituido then
        nuevo_coste ← restar distancia del elemento i al elemento_sustituido
        nuevo_coste ← sumardistanciadelelemento i al elemento_incluido)
/* Indicar si se ha mejorado el coste o no */
if nuevo_coste > coste_actual then
    mejorado ← true
else
    mejorado ← false
return mejorado, elemento_incluido, nuevo_coste
```

3.1.3. Generación de soluciones aleatorias

Tanto las soluciones de la Búsqueda Local del Primero Mejor como de la Búsqueda Local del Mejor se crean de forma aleatoria.

En el caso del Mejor se crea de forma aleatoria la solución inicial, y no se crea de forma aleatoria las soluciones vecinas, ya que en este algoritmo se examina todo el entorno, por lo que es absurdo gastar tiempo en crear una solución aleatoria cuando se van a examinar todas.

Para crear la solución inicial se van eligiendo elementos del conjunto de forma aleatoria hasta tener el número de elementos necesario para forma una solución. El resto de elementos del conjunto se introducen en la lista de no seleccionados.

En el caso del Primero Mejor se crea de forma aleatoria tanto la solución inicial, como las soluciones vecinas, ya que en este caso se cogerá como vecina la primera solución que mejora la actual, por lo que no se exploran siempre todos los vecinos.

La solución inicial se genera de la misma forma que en del Mejor. La solución vecina se crea sustituyendo el elemento que menos aporta por uno de los elementos no seleccionados, para que la creación sea aleatoria antes de empezar a generar vecinos para ver cuál mejora la solución

actual, se baraja de forma aleatoria los elementos no seleccionados, para que a la hora de recorre los elementos de forma iterativa sean aleatorios los elementos que nos vayamos encontrando, esta acción se realiza con la función *shuffle*.

3.2. Búsqueda Local del Primero Mejor

3.2.1. Método de exploración del entorno

En esta variante de la Búsqueda Local se van generando vecinos hasta encontrar uno que mejore la solución actual, si se encuentra uno que la mejore se continua con la búsqueda; si no se encuentra un vecino que la mejore, se para el proceso de búsqueda y nos quedamos con la solución actual. Además se puede parar el proceso si se llega a un límite de iteraciones, y se incrementa una iteración por cada llamada a al función objetivo.

Algorithm 7 Método de exploración del entorno (BL_PM)

```
solucion_actual ← generar solución inicial aleatoria
iteraciones ← 0
hay_mejora ← true
while iteraciones ≤ 100000 & hay_mejora do
    noSeleccionadosNuevo ← barajar aleatoriamente la lista de elementos no seleccionados
    hay_mejora, elegido ← obtener el primer vecino que mejora el coste de la solución
    /* Si se encuentra un vecino que mejore el coste */
    if hay_mejora then
        solucion_modificada ← intercambiar el elemento elegido por el elemento de la solución
que menos aporte
        contribuciones_modi ← actualizar contribuciones con el nuevo elemento de la solución
```

3.3. Búsqueda Local del Mejor

3.3.1. Método de exploración del entorno

En esta variante de la Búsqueda Local se genera todo el entorno de una solución, y se elige la solución que consiga una mayor mejora de la solución actual, si no se alcanza ninguna mejor se para el proceso de búsqueda. Además se puede parar el proceso si se llega a un límite de iteraciones, y se incrementa una iteración por cada llamada a al función objetivo. Su implementación en pseudocódigo es la siguiente:

Algorithm 8 Método de exploración del entorno (BL_M)

solucion_actual ← *generar solución inicial aleatoria*
iteraciones ← 0
hay_mejora ← *true*
while iteraciones ≤ 100000 & *hay_mejora* **do**
noSeleccionadosNuevo ← *barajar aleatoriamente la lista de elementos no seleccionados*
hay_mejora, elegido ← *obtener el vecino que mejora en mayor medida el coste de la solución*
/* Si se encuentra un vecino que mejore el coste */
if *hay_mejora* **then**
solucion_modificada ← *intercambiar el elemento elegido por el elemento de la solución que menos aporte*
contribuciones_modi ← *actualizar contribuciones con el nuevo elemento de la solución*

4. Descripción del Greedy

4.1. Algoritmo de comparación

Para elegir que elemento incluir en la solución en cada momento se tiene en cuenta sus distancias con el resto de elementos.

En primer lugar al insertar el primer elemento se tiene en cuenta que el elemento insertado sea el que mayor distancia acumulada tenga.

Algorithm 9 Elegir primer elemento a insertar en la solución

Input: *distancias*: matriz que guarda la distancia entre cada par de elementos del conjunto total.
Input: *noSeleccionados*: lista de elementos no incluidos en la solución.
Output: *elegido*: elemento elegido para ser insertado.

```
distanciasAcu ← inicializar vector a 0.0
noSeleccionados_size ← número de elementos no seleccionados
/* Calcular la distancia acumulada de cada elemento no seleccionado */
for i ∈ {0, ..., noSeleccionados_size − 1} do
    for j ∈ {i + 1, ..., noSeleccionados_size} do
        distanciasAcu ← aumentar distancia acumulada del elemento i
        distanciasAcu ← aumentar distancia acumulada del elemento j
distAcuMax ← 0.0
elegido ← vacio
/* Elegir el elemento con mayor distancia acumulada */
for i ∈ {noSeleccionados} do
    if distAcuMax < {distancia acumulada de i} then
        disAcuMax ← asigna la distancia acumulada del elemento i
        elegido ← i
return elegido
```

En el resto de elementos a insertar lo que que se tiene en cuenta es que tenga la mayor distancia de las mínimas distancias entre los elementos de la solución y los elementos no seleccionados.

Algorithm 10 Elegir el resto de elementos a insertar en la solución

Input: *distancias*: matriz que guarda la distancia entre cada par de elementos del conjunto total.
Input: *solucion*: lista de elementos incluidos en la solución.
Input: *noSeleccionados*: lista de elementos no incluidos en la solución.
Output: *elegido*: elemento elegido para ser insertado.

```
    distanciaMin ← inicializar vector al máximo valor de un flotante
    /* Calcular la distancia mínima entre cada elemento de la solución y el resto de elementos
    no seleccionados */
    for i ∈ solucion do
        for j ∈ noSeleccionados do
            distanciaMin ← quedarse con lamínima distancia al elemento i
    distMax ← −1
    elegido ← vacio
    /* Elegir el elemento con mayor distancia mínima */
 9: for i ∈ solucion do
        if distMax < {distancia mínima de i} then
            distMax ← asigna la distancia mínima del elemento i
            elegido ← i
    return elegido
```

Se sigue el proceso de inserción de elementos hasta que la solución alcanza su tamaño necesario para ser válida como solución.

5. Descripción del Algoritmo Genético

5.1. Esquema de evolución y reemplazamiento

Para implementar el algoritmo genético hemos utilizado dos modelos:

5.1.1. Generacional

Algorithm 11 Modelo generacional

Input: *poblacionIni*: Primer padre.
Input: *distancias*: matriz de distancias entre pares de elementos del conjunto.
Input: *numGenesFactible*: número de elementos necesarios para que la solución sea factible.
Input: *probabilidadCruce*: probabilidad de que una pareja de individuos se cruce.
Input: *probabilidadMutacion*: probabilidad de que un gen mute.
Output: *fitness*: mejor fitness encontrado en la ejecución del algoritmo.

```
poblacionActual ← poblacionIni
iteraciones ← 0
while iteraciones < 100000 do
    mejorPadre ← Primer elemento de la población actual
    poblacionActual ← Realizar el operador de selección generacional
    poblacionActual ← Realizar el operador de cruce
    poblacionActual ← Realizar el operador demutación
    poblacionActual ← Calcular el fitness de la población
    poblacionActual ← Realizar el operador de reemplazo generacional
    iteraciones ← iteraciones + número de evaluaciones
return fitness
```

Algorithm 12 Operador de reemplazo generacional

Input: *poblacion*: lista de individuos.
Input: *mejorPadre*: mejor individuo de la población anterior.
Output: *poblacionActual*: lista de individuos después del reemplazo.

```
poblacionActual ← poblacion
poblacionActual ← Añadir el mejorPadre
poblacionActual ← Ordenardemayoramenorfitness

/* Búsqueda del mejor padre */
tam ← Tamaño de poblacionActual
encontrado ← 0
for i ∈ {0, ..., tam − 1} do
    if encontrado == 2 then
        break
    fitness ← Obtener el fitness del cromosoma situado en la posición i de la poblacionActual
    fitnessPadre ← Obtener el fitness del mejorPadre
    if fitness < fitnessPadre then
        break

    if fitness == fitnessPadre then
18:     cromosoma ← Obtener el cromosoma situado en la posición i de la poblacionActual
        if cromosoma == mejorPadre then
            encontrado ← encontrado + 1
            encontradoCromo ← cromosoma

if encontrado == 2 then
    poblacionActual ← Eliminar cromosoma encontradoCromo
else
    if encontrado == 1 then
        poblacionActual ← Eliminar un elemento de la población que no sea el mejorPadre
return poblacionActual
```

5.1.2. Estacionario

Algorithm 13 Modelo estacionario

Input: *poblacionIni*: Primer padre.
Input: *distancias*: matriz de distancias entre pares de elementos del conjunto.
Input: *numGenesFactible*: número de elementos necesarios para que la solución sea factible.
Input: *probabilidadCruce*: probabilidad de que una pareja de individuos se cruce.
Input: *probabilidadMutacion*: probabilidad de que un gen mute.
Output: *fitness*: mejor fitness encontrado en la ejecución del algoritmo.

```
poblacionActual ← poblacionIni
iteraciones ← 0
while iteraciones < 100000 do
    nuevaPoblacion ← Vacío
    nuevaPoblacion ← Realizar el operador de selección estacionario
    nuevaPoblacion ← Realizar el operador de cruce
    nuevaPoblacion ← Realizar el operador de mutación
    nuevaPoblacion ← Calcular el fitness de la población
    poblacionActual ← Realizar el operador de reemplazo estacionario
    iteraciones ← iteraciones + número de evaluaciones
return fitness
```

Algorithm 14 Operador de reemplazo estacionario

Input: *poblacionActual*: lista de individuos.
Input: *nuevaPoblacion*: lista de individuos.
Output: *poblacionActual*: lista de individuos después del reemplazo.

```
poblacionActual ← Añadir toda la nuevaPoblacion
poblacionActual ← Ordenar de mayor a menor fitness

/* Eliminar los peores individuos */
tam ← Tamaño de nuevaPoblacion
for i ∈ {0, ..., tam − 1} do
    poblacionActual ← Eliminar el último elemento de la lista
return poblacionActual
```

6. Descripción del Algoritmo Memético

6.1. Esquema de búsqueda

Todos los esquemas implementados se ejecutan cada 10 generaciones del algoritmo genético, y con una intensidad máxima de 400 iteraciones. Lo único que los diferencia es el número de cromosomas sobre los que se realiza la búsqueda local.

6.1.1. Primer esquema

En este esquema se realiza sobre toda la población.

Algorithm 15 Primer esquema de búsqueda

Input: *poblacionIni*: Primer padre.
Input: *distancias*: matriz de distancias entre pares de elementos del conjunto.
Input: *numGenesFactible*: número de elementos necesarios para que la solución sea factible.
Input: *probabilidadCruce*: probabilidad de que una pareja de individuos se cruce.
Input: *probabilidadMutacion*: probabilidad de que un gen mute.
Output: *fitness*: mejor fitness encontrado en la ejecución del algoritmo.

```
poblacionActual ← poblacionIni
iteraciones ← 0
generacion ← 0
while iteraciones < 100000 do
    nuevaPoblacion ← Vacío
    nuevaPoblacion ← Realizar el operador de selección estacionario
    nuevaPoblacion ← Realizar el operador de cruce uniforme
    nuevaPoblacion ← Realizar el operador de mutación
    nuevaPoblacion ← Calcular el fitness de la población
    poblacionActual ← Realizar el operador de reemplazo estacionario
    iteraciones ← iteraciones + número de evaluaciones
    generacion ← generacion + 1
    if generacion == 10 then
        intensidad ← 400
        tam ← Tamaño de la poblacionActual
        for i ∈ {0, ..., tam − 1} do
            cromosoma ← Obtener el cromosoma de la poblacionActual situado en la posición i
            cromosoma ← Realizar una búsqueda local
        generacion ← 0
21: return fitness
```

6.1.2. Segundo esquema

En este esquema se realiza sobre el 10 % de la población.

Algorithm 16 Segundo esquema de búsqueda

Input: *poblacionIni*: Primer padre.
Input: *distancias*: matriz de distancias entre pares de elementos del conjunto.
Input: *numGenesFactible*: número de elementos necesarios para que la solución sea factible.
Input: *probabilidadCruce*: probabilidad de que una pareja de individuos se cruce.
Input: *probabilidadMutacion*: probabilidad de que un gen mute.
Output: *fitness*: mejor fitness encontrado en la ejecución del algoritmo.

```
poblacionActual ← poblacionIni
iteraciones ← 0
generacion ← 0
while iteraciones < 100000 do
    nuevaPoblacion ← Vacío
    nuevaPoblacion ← Realizar el operador de selección estacionario
    nuevaPoblacion ← Realizar el operador de cruce uniforme
    nuevaPoblacion ← Realizar el operador de mutación
    nuevaPoblacion ← Calcular el fitness de la población
    poblacionActual ← Realizar el operador de reemplazo estacionario
    iteraciones ← iteraciones + número de evaluaciones
    generacion ← generacion + 1
    if generacion == 10 then
        intensidad ← 400
        tam ← Tamaño de la poblacionActual
        numCromosomas ← 0.1 * tam
        listaCromosomas ← Vacío
        tamLista ← 0
        while tamLista < numCromosomas do
            nuevo ← Número aleatorio entre 0 y tam − 1
            listaCromosomas ← Añadir nuevo
            tamLista ← tamLista + 1
        for i ∈ {0, ..., tamLista − 1} do
            cromosoma ← Obtener el cromosoma de la poblacionActual situado en la posición i
            cromosoma ← Realizar una búsqueda local
        generacion ← 0
return fitness
```

6.1.3. Tercer esquema

En este esquema se realiza sobre el 10 % de los mejores de la población.

Algorithm 17 Tercer esquema de búsqueda

Input: *poblacionIni*: Primer padre.
Input: *distancias*: matriz de distancias entre pares de elementos del conjunto.
Input: *numGenesFactible*: número de elementos necesarios para que la solución sea factible.
Input: *probabilidadCruce*: probabilidad de que una pareja de individuos se cruce.
Input: *probabilidadMutacion*: probabilidad de que un gen mute.
Output: *fitness*: mejor fitness encontrado en la ejecución del algoritmo.

```
poblacionActual ← poblacionIni
iteraciones ← 0
generacion ← 0
while iteraciones < 100000 do
    nuevaPoblacion ← Vacío
    nuevaPoblacion ← Realizar el operador de selección estacionario
    nuevaPoblacion ← Realizar el operador de cruce uniforme
    nuevaPoblacion ← Realizar el operador de mutación
    nuevaPoblacion ← Calcular el fitness de la población
    poblacionActual ← Realizar el operador de reemplazo estacionario
    iteraciones ← iteraciones + número de evaluaciones
    generacion ← generacion + 1
    if generacion == 10 then
        intensidad ← 400
        tam ← Tamaño de la poblacionActual
        /* Obtener los cromosomas que se van a utilizar */
        numCromosomas ← 0.1 * tam
        /* Se recorre desde el principio porque la poblacionActual se encuentra ordenada de
mayor a menor fitness */
        for i ∈ {0, ..., numCromosomas − 1} do
            cromosoma ← Obtener el cromosoma de la poblacionActual situado en la posición i
            cromosoma ← Realizar una búsqueda local
        generacion ← 0
return fitness
```

7. Descripción del Algoritmo de Enfriamiento Simulado

Para este algoritmo he creado una variante en la que al crear una nueva solución se sustituye siempre por el elemento que menos contribuye a la solución, a diferencia del ES original en el que se sustituye por un elemento aleatorio. Con esta modificación se crea un ES inteligente. También se ha utilizado esta variante para crear otro algoritmo ILS.

7.1. Esquema de búsqueda

Algorithm 18 Esquema de búsqueda del ES

Input: *solucion*: lista con las etiquetas de los elementos que forman la solución actual.
Input: *no_seleccionados*: lista con las etiquetas de los elementos que no forman parte de la solución.
Input: *distancias*: matriz de distancias entre pares de elementos del conjunto.
Input: *iter*: número de evaluaciones actual.
Input: *iter_max*: número máximo de evaluaciones.
Output: *mejor_solucion*: lista con las etiquetas de los elementos que forman la nueva solución actual.
Output: *no_seleccionados*: lista con las etiquetas de los elementos que no forman parte de la nueva solución.
Output: *mejor_fitness*: mejor fitness encontrado en la ejecución del algoritmo.
Output: *evaluaciones*: número de evaluaciones al terminar el algoritmo.

```
/* Valores de los parámetros usados en el enfriamiento */
fi ← 0.3
mu ← 0.3
temp_final ← 0.001

/* Evaluación de la solución actual */
coste_actual ← Obtener el fitness de la solución actual

/* Fijar la mejor solución */
mejor_solucion ← solucion
mejor_fitness ←coste_actual

/* Cálculo de la temperatura inicial */
temp_ini ← Cálculo de la temperatura inicial

/* Fijar temperatura actual */
temp ←temp_ini

/* Se suma uno para tener en cuenta la evaluación de la solución actual */
evaluaciones ← iter + 1
```

```
iteraciones ← 0
num_exitos ← 0
num_vec_generador ← Vacío
tam_sol ← Tamañodelasoluciónactual
num_max_vec_generador ← 10∗tam_sol
num_max_exitos ←tam_sol
while evaluaciones != iter_max && num_exitos > 0 do
    num_exitos ← 0
    num_vec_generador ← 0
    while num_exitos != num_max_exitos &&
            num_vec_generador != num_max_vec_generador do

/* Generar un vecino */
        tam_sol ← Tamaño de la solución actual
        elem_a_sustituir ← Número aleatorio entre 0 y tam_sol
        tam_no_seleccionados ← Tamaño de los no seleccionados de la soluciónactual
        elem_a_incluir ← Número aleatorio entre 0 y tam_no_seleccionados

/* Evaluación del nuevo vecino */
        coste_vecino ← Obtener el fitness de la solución con el nuevo vecino
        evaluaciones ← Incrementar en uno las evaluaciones
        num_vec_generador ← Incrementar en uno los vecinos generados

/* Comprobar si se sustituye la mejor solución */
        diferencia ←coste_actual − coste_vecino

/* Si la nueva solución es mejor que la actual */
        if diferencia < 0 then
            solucion ← Realizar el intercambio en la solución actual del nuevo vecino
            coste_actual ←coste_vecino
            if coste_actual > mejor_fitness then
                mejor_fitness ←coste_actual
                mejor_solucion ← solucion
            num_exitos ← Incrementar en uno los números de éxitos
        else

/* Si la nueva solución no es mejor que la actual */
            probabilidad ← Número aleatorio entre 0 y 1
            k ← 1
            limite ← exp((−diferencia)/(k ∗ temp))
            if probabilidad <= limite then
                solucion ← Realizar el intercambio en la solución actual del nuevo vecino
                coste_actual ←coste_vecino
                num_exitos ← Incrementar en uno los números de éxitos
        if evaluaciones == iter_max then
            Terminar bucle
    iteraciones ← Incrementar en uno las iteraciones
    temp ← Realizar el enfriamiento de la temperatura actual
no_seleccionados ← Obtener los elementos no seleccionados a partir de la mejor solución
return mejor_fitness, mejor_solucion, no_seleccionados, evaluaciones
```

7.2. Cálculo de la temperatura inicial

Algorithm 19 Cálculo de la temperatura inicial

Input: *mu*: porcentaje de empeoramiento de la solución inicial.
Input: *coste_actual*: lista con las etiquetas de los elementos que no forman parte de la solución.
Input: *fi*: es la probabilidad de aceptar una solución.
Output: *temp_ini*: temperatura inicial.

$temp\_ini \leftarrow (mu*\text{coste\_actual}) \ / \ (-log(fi))$
return *temp_ini*

7.3. Esquema de enfriamiento

Algorithm 20 Esquema de enfriamiento

Input: *temp*: temperatura actual.
Input: *temp_ini*: temperatura inicial.
Input: $temp\_final$: temperatura final.
Input: *iteraciones*: número de iteraciones realizadas actualmente.
Output: *nueva_temp*: nueva temperatura.

/* Si se supera la temperatura final, esta se disminuye */
if $temp\_final >= temp\_ini$ **then**
    $temp_final \leftarrow$temp_final / 100

/* Se calcula la constante beta */
beta $\leftarrow$ (temp_ini − temp_final) / (*iteraciones*∗temp_ini∗temp_final)
$nueva\_temp \leftarrow temp \ / \ (1 \ + \ beta * temp)$
return *nueva_temp*

8. Descripción del Algoritmo de Búsqueda Multiarranque Básica

8.1. Esquema de búsqueda

Algorithm 21 Esquema de búsqueda del BMB

Input: *solucion*: lista con las etiquetas de los elementos que forman la solución actual.
Input: *distancias*: matriz de distancias entre pares de elementos del conjunto.
Input: m: número de elementos en la solución.
Output: *mejor_solucion*: lista con las etiquetas de los elementos que forman la nueva solución actual.
Output: $mejor\_fitness$: mejor fitness encontrado en la ejecución del algoritmo.

$iteraciones\_max \leftarrow 10$
iteraciones $\leftarrow 0$
$mejor\_solucion \leftarrow Vacío$
$mejor\_fitness \leftarrow 0$
$iter\_fin \leftarrow 100000$ / iteraciones_max
while iteraciones != $iteraciones\_max$ **do**
  $solucion\_aleatoria \leftarrow Vacío$
  $no\_seleccionados\_aleatoria \leftarrow Vacío$
  $solucion\_aleatoria$, $no\_seleccionados\_aleatoria \leftarrow Generar\ solución\ aleatoria$
  $iter\_ini \leftarrow 0$
  $nuevo\_fitness$, $iter\_ini \leftarrow Realizar\ BL$
  if $nuevo\_fitness > mejor\_fitness$ **then**
    $mejor_fitness \leftarrow$nuevo_fitness
    $mejor_solucion \leftarrow$solucion_aleatoria
  iteraciones $\leftarrow Incrementar\ en\ uno\ las\ iteraciones$
return $mejor\_fitness$, $mejor\_solucion$

9. Descripción del Algoritmo de Búsqueda Local Reiterada

9.1. Esquema de búsqueda

Algorithm 22 Esquema de búsqueda del ILS

Input: *solucion*: lista con las etiquetas de los elementos que forman la solución actual.
Input: *distancias*: matriz de distancias entre pares de elementos del conjunto.
Input: m: número de elementos en la solución.
Output: *mejor_solucion*: lista con las etiquetas de los elementos que forman la nueva solución actual.
Output: $mejor\_fitness$: mejor fitness encontrado en la ejecución del algoritmo.

$iteraciones\_max \leftarrow 10$
iteraciones $\leftarrow 0$
$mejor\_fitness \leftarrow Vacío$

/* Cálculo del número de mutaciones */
t $\leftarrow 0.1 * m$
$iter\_fin \leftarrow$ 100000 / iteraciones_max
$mejor\_solucion \leftarrow Vacío$
$no\_seleccionados \leftarrow Vacío$
$fitness\_actual \leftarrow Vacío$
$mejor\_solucion$, $no\_seleccionados \leftarrow$ *Generar solución aleatoria*
$iter\_ini \leftarrow 0$
$mejor\_solucion$, $no\_seleccionados$, $mejor\_fitness \leftarrow$ *Realizar BL*
iteraciones != $iteraciones\_max$
while evaluaciones != $iter\_max$ && $num\_exitos > 0$ **do**
    $nueva_solucion \leftarrow$mejor_solucion
    $nuevo_no_seleccionados \leftarrow$no_seleccionados
    $nueva\_solucion$, $nuevo\_no\_seleccionados \leftarrow$ *Realizar mutación de la nueva solución*
    $iter\_ini \leftarrow 0$
    $nuevo\_fitness$, $nueva\_solucion$, $nuevo\_no\_seleccionados \leftarrow$ *Realizar BL*
    if $nuevo\_fitness > mejor\_fitness$ **then**
        $mejor_fitness \leftarrow$nuevo_fitness
        $mejor_solucion \leftarrow$nueva_solucion
        $no_seleccionados \leftarrow$nuevo_no_seleccionados
    iteraciones $\leftarrow$ *Incrementar en uno las iteraciones*
return $mejor\_fitness$, $mejor\_solucion$

Algorithm 23 Esquema de búsqueda del ILS con ES

Input: *solucion*: lista con las etiquetas de los elementos que forman la solución actual.
Input: *distancias*: matriz de distancias entre pares de elementos del conjunto.
Input: m: número de elementos en la solución.
Output: *mejor_solucion*: lista con las etiquetas de los elementos que forman la nueva solución actual.
Output: $mejor\_fitness$: mejor fitness encontrado en la ejecución del algoritmo.

$iteraciones\_max \leftarrow 10$
iteraciones $\leftarrow 0$
$mejor\_fitness \leftarrow Vacío$

/* Cálculo del número de mutaciones */
t $\leftarrow 0.1 * m$
$iter\_fin \leftarrow$ 100000 / iteraciones_max
$mejor\_solucion \leftarrow Vacío$
$no\_seleccionados \leftarrow Vacío$
$fitness\_actual \leftarrow Vacío$
$mejor\_solucion$, $no\_seleccionados \leftarrow$ *Generar solución aleatoria*
$iter\_ini \leftarrow 0$
$mejor\_solucion$, $no\_seleccionados$, $mejor\_fitness \leftarrow$ *Realizar ES*
iteraciones != $iteraciones\_max$
while evaluaciones != $iter\_max$ && $num\_exitos > 0$ **do**
    $nueva_solucion \leftarrow$mejor_solucion
    $nuevo_no_seleccionados \leftarrow$no_seleccionados
    $nueva\_solucion$, $nuevo\_no\_seleccionados \leftarrow$ *Realizar mutación de la nueva solución*
    $iter\_ini \leftarrow 0$
    $nuevo\_fitness$, $nueva\_solucion$, $nuevo\_no\_seleccionados \leftarrow$ *Realizar ES*
    if $nuevo\_fitness > mejor\_fitness$ **then**
        $mejor_fitness \leftarrow$nuevo_fitness
        $mejor_solucion \leftarrow$nueva_solucion
        $no_seleccionados \leftarrow$nuevo_no_seleccionados
    iteraciones $\leftarrow$ *Incrementar en uno las iteraciones*
return $mejor\_fitness$, $mejor\_solucion$

Algorithm 24 Esquema de búsqueda del ILS con ES inteligente

Input: *solucion*: lista con las etiquetas de los elementos que forman la solución actual.
Input: *distancias*: matriz de distancias entre pares de elementos del conjunto.
Input: m: número de elementos en la solución.
Output: *mejor_solucion*: lista con las etiquetas de los elementos que forman la nueva solución actual.
Output: $mejor\_fitness$: mejor fitness encontrado en la ejecución del algoritmo.

$iteraciones\_max \leftarrow 10$
iteraciones $\leftarrow 0$
$mejor\_fitness \leftarrow Vacío$

/* Cálculo del número de mutaciones */
t $\leftarrow 0.1 * m$
$iter\_fin \leftarrow$ 100000 / iteraciones_max
$mejor\_solucion \leftarrow Vacío$
$no\_seleccionados \leftarrow Vacío$
$fitness\_actual \leftarrow Vacío$
$mejor\_solucion$, $no\_seleccionados \leftarrow$ *Generar solución aleatoria*
$iter\_ini \leftarrow 0$
$mejor\_solucion$, $no\_seleccionados$, $mejor\_fitness \leftarrow$ *Realizar ES inteligente*
iteraciones != $iteraciones\_max$
while evaluaciones != $iter\_max$ && $num\_exitos > 0$ **do**
    $nueva_solucion \leftarrow$mejor_solucion
    $nuevo_no_seleccionados \leftarrow$no_seleccionados
    $nueva\_solucion$, $nuevo\_no\_seleccionados \leftarrow$ *Realizar mutación de la nueva solución*
    $iter\_ini \leftarrow 0$
    $nuevo\_fitness$, $nueva\_solucion$, $nuevo\_no\_seleccionados \leftarrow$ *Realizar ES inteligente*
    if $nuevo\_fitness > mejor\_fitness$ **then**
        $mejor_fitness \leftarrow$nuevo_fitness
        $mejor_solucion \leftarrow$nueva_solucion
        $no_seleccionados \leftarrow$nuevo_no_seleccionados
    iteraciones $\leftarrow$ *Incrementar en uno las iteraciones*
return $mejor\_fitness$, $mejor\_solucion$

9.2. Operador de mutación

Algorithm 25 Operador de mutación

Input: *solucion*: lista con las etiquetas de los elementos que forman la solución actual.
Input: *no_seleccionados*: lista con las etiquetas de los elementos que no forman parte de la solución.
Input: *distancias*: matriz de distancias entre pares de elementos del conjunto.
Input: t: número de mutaciones.
Output: *nueva_solucion*: lista con las etiquetas de los elementos que forman la nueva solución actual.
Output: *nuevo_no_seleccionados*: lista con las etiquetas de los elementos que no forman parte de la nueva solución.

num_mutaciones $\leftarrow 0$
while *num_mutaciones* != t **do**
tam_sol $\leftarrow$ *Tamaño de la solucion*
elem_a_sustituir $\leftarrow$ *Número aleatorio entre* 0 *y* tam_sol $-$ 1
tam_no_seleccionados $\leftarrow$ *Tamaño de los no seleccionados*
elem_a_incluir $\leftarrow$ *Número aleatorio entre* 0 *y* tam_no_seleccionados $-$ 1
nueva_solucion, *nuevo_no_seleccionados* $\leftarrow$ *Generar nueva solución intercambiando* los elementos seleccionados
num_mutaciones $\leftarrow$ *Incrementar en uno el número de mutaciones realizadas*
return *nueva_solucion*, *nuevo_no_seleccionados*

10. Desarrollo de la práctica

La práctica la he implementado en C++, sin hacer uso de frameworks.

Para tomar los tiempos he utilizado el código del timer que se nos proporciona con la práctica. El resto de código está totalmente implementado por mi.

10.1. Compilación

Para compilar utilizo un archivo Makefile. Al ejecutar "make" en el terminal, se lanza su regla general, que genera todos los ejecutables, los ficheros objeto, y las carpetas necesarias. En el caso de los ejecutables crea dos por cada algoritmo, uno para depurar, y el otro para tomar tiempos con la opción de compilación -O2.

Para más información del Makefile abrir el archivo makefile y ver los comentarios y reglas que hay en él, el makefile se encuentra en la raíz del proyecto.

10.2. Limpiar ficheros derivados de la compilación

Para eliminar todos estos ficheros también hago uso del Makefile con la regla "make clean".

10.3. Obtener resultados

Para obtener los resultados he creado un script bash que ejecuta cada algoritmo con todos los casos y da como resultado un archivo CSV con los costes y tiempos del algoritmo en cada caso.

Para obtener los resultados de todos los algoritmos hay que ejecutar "./script_tiempos.sh" o ejecutar "make tiempos". Recomiendo esta última ya que da permisos al archivo para poder ejecutarse.

Y si se quiere obtener sólo el resultado de un algoritmo en concreto, se puede hacer ejecutando "./script_tiempos.sh <nombre del algoritmo sin extensión>".

Ejemplo:

- ./script_tiempos.sh BL_M
- ./script_tiempos.sh BL_PM
- ./script_tiempos.sh Greedy

Además de los CSV hay un archivo Excel de cada algoritmo que calcular la desviación de cada caso, la desviación media y el tiempo medio. Este archivo hay que modificarlo a mano, es decir, copiar y pegar los resultados del CSV.

Si se quiere ejecutar el archivo ejecutable de alguno de los algoritmos se debe ejecutar de la siguiente forma:

./<nombre del ejecutable> <seed> <nombre del fichero de datos con su ruta>

10.4. Distribución de carpetas

El proyecto está divido en las siguientes carpetas:

- src ← contiene los archivos fuente
- include ← contiene los archivos de cabecera
- data ← contiene los archivos de datos de los distintos casos
- tablas ← contiene los CSV y Excel con los resultados
- bin ← contiene los ejecutables
- obj ← contiene los ficheros objeto

Además en la raíz del proyecto se encuentra el Makefile y el script de bash.

11. Experimentos y Análisis de resultados

11.1. Descripción de los casos y valores de parámetros

Los casos utilizados se agrupan por diferente cantidad de elementos para el conjunto y cantidad de elementos para la solución. Para obtener los tiempos se han utilizado casos con los siguientes tamaños:

- 500 elementos en el conjunto, y 50 elementos para la solución
- 2000 elementos en el conjunto, y 200 elementos para la solución
- 3000 elementos en el conjunto, y 300, 400, 500, ó 600 elementos para la solución

Los argumentos pasados a los algoritmos son los siguientes:

- Nombre del caso
- Semilla

Para todas las ejecuciones de los algoritmos con los distintos casos se ha utilizado la semilla con valor 0.

11.2. Resultados obtenidos

Todos los tiempos están expresados en segundos.

11.2.1. Búsqueda Local

Algoritmo Búsqueda Local del Primero Mejor		
Caso	**Desv**	**Tiempo**
MDG-a_1_n500_m50	3,23	0,009096
MDG-a_2_n500_m50	2,25	0,009857
MDG-a_3_n500_m50	2,32	0,008824
MDG-a_4_n500_m50	1,53	0,009175
MDG-a_5_n500_m50	2,13	0,008875
MDG-a_6_n500_m50	1,88	0,009707
MDG-a_7_n500_m50	1,62	0,010312
MDG-a_8_n500_m50	1,79	0,009197
MDG-a_9_n500_m50	2,22	0,008761
MDG-a_10_n500_m50	1,93	0,008628
MDG-b_21_n2000_m200	0,73	0,217935
MDG-b_22_n2000_m200	0,88	0,220469
MDG-b_23_n2000_m200	1,20	0,221893
MDG-b_24_n2000_m200	0,89	0,228545
MDG-b_25_n2000_m200	0,87	0,224658
MDG-b_26_n2000_m200	1,22	0,216504
MDG-b_27_n2000_m200	1,31	0,219684
MDG-b_28_n2000_m200	1,05	0,215919
MDG-b_29_n2000_m200	1,21	0,221140
MDG-b_30_n2000_m200	0,86	0,221838
MDG-c_1_n3000_m300	0,68	0,434002
MDG-c_2_n3000_m300	0,77	0,435650
MDG-c_8_n3000_m400	0,41	0,617974
MDG-c_9_n3000_m400	0,56	0,602094
MDG-c_10_n3000_m400	0,60	0,590469
MDG-c_13_n3000_m500	0,53	0,836042
MDG-c_14_n3000_m500	0,34	0,805850
MDG-c_15_n3000_m500	0,25	0,797442
MDG-c_19_n3000_m600	0,39	1,009410
MDG-c_20_n3000_m600	0,49	1,010500

Tabla 1: Resultados BL_PM

11.2.2. Greedy

Algoritmo Greedy		
Caso	**Desv**	**Tiempo**
MDG-a_1_n500_m50	24,63	0,00283
MDG-a_2_n500_m50	21,77	0,00279
MDG-a_3_n500_m50	23,50	0,00268
MDG-a_4_n500_m50	23,39	0,00279
MDG-a_5_n500_m50	24,98	0,00276
MDG-a_6_n500_m50	22,74	0,00274
MDG-a_7_n500_m50	24,36	0,00286
MDG-a_8_n500_m50	24,61	0,00263
MDG-a_9_n500_m50	19,67	0,00258
MDG-a_10_n500_m50	26,99	0,00272
MDG-b_21_n2000_m200	12,81	0,23355
MDG-b_22_n2000_m200	12,53	0,23452
MDG-b_23_n2000_m200	12,01	0,25347
MDG-b_24_n2000_m200	12,48	0,24368
MDG-b_25_n2000_m200	12,63	0,23630
MDG-b_26_n2000_m200	12,21	0,26197
MDG-b_27_n2000_m200	12,12	0,24787
MDG-b_28_n2000_m200	12,58	0,22852
MDG-b_29_n2000_m200	13,10	0,24468
MDG-b_30_n2000_m200	12,53	0,22586
MDG-c_1_n3000_m300	10,22	0,86770
MDG-c_2_n3000_m300	10,55	0,82770
MDG-c_8_n3000_m400	8,42	1,50415
MDG-c_9_n3000_m400	8,47	1,53032
MDG-c_10_n3000_m400	8,11	1,49027
MDG-c_13_n3000_m500	7,15	2,24823
MDG-c_14_n3000_m500	7,10	2,38319
MDG-c_15_n3000_m500	7,21	2,18022
MDG-c_19_n3000_m600	6,13	3,21624
MDG-c_20_n3000_m600	6,24	3,22367

Tabla 2: Resultados Greedy

11.2.3. Algoritmo Genético

Algoritmo Genético Generacional Uniforme		
Caso	**Desv**	**Tiempo**
MDG-a_1_n500_m50	2,86	2,05134
MDG-a_2_n500_m50	2,29	2,42187
MDG-a_3_n500_m50	0,97	2,40216
MDG-a_4_n500_m50	2,26	2,27867
MDG-a_5_n500_m50	1,62	2,08868
MDG-a_6_n500_m50	2,02	2,54159
MDG-a_7_n500_m50	1,49	2,28424
MDG-a_8_n500_m50	1,76	2,36072
MDG-a_9_n500_m50	0,80	2,68403
MDG-a_10_n500_m50	2,55	2,36472
MDG-b_21_n2000_m200	1,27	52,48460
MDG-b_22_n2000_m200	1,55	49,41550
MDG-b_23_n2000_m200	1,14	55,40780
MDG-b_24_n2000_m200	1,24	55,32120
MDG-b_25_n2000_m200	1,53	44,71500
MDG-b_26_n2000_m200	1,64	59,70840
MDG-b_27_n2000_m200	1,29	55,97080
MDG-b_28_n2000_m200	1,24	57,95830
MDG-b_29_n2000_m200	1,92	49,19690
MDG-b_30_n2000_m200	1,28	60,27940
MDG-c_1_n3000_m300	1,15	169,7140
MDG-c_2_n3000_m300	1,29	180,8270
MDG-c_8_n3000_m400	0,87	262,0220
MDG-c_9_n3000_m400	0,94	248,8020
MDG-c_10_n3000_m400	1,28	295,1720
MDG-c_13_n3000_m500	0,78	374,8730
MDG-c_14_n3000_m500	0,96	376,7850
MDG-c_15_n3000_m500	0,82	381,4910
MDG-c_19_n3000_m600	0,70	555,2500
MDG-c_20_n3000_m600	0,79	541,8560

Tabla 3: Resultados AGG-uniforme

Algoritmo Genético Generacional Posición		
Caso	**Desv**	**Tiempo**
MDG-a_1_n500_m50	3,12	0,68471
MDG-a_2_n500_m50	3,90	0,68979
MDG-a_3_n500_m50	2,89	0,69802
MDG-a_4_n500_m50	2,84	0,69004
MDG-a_5_n500_m50	3,18	0,69149
MDG-a_6_n500_m50	2,48	0,68629
MDG-a_7_n500_m50	1,79	0,68981
MDG-a_8_n500_m50	2,57	0,68805
MDG-a_9_n500_m50	1,41	0,68522
MDG-a_10_n500_m50	3,46	0,69386
MDG-b_21_n2000_m200	2,20	6,94798
MDG-b_22_n2000_m200	2,78	6,86697
MDG-b_23_n2000_m200	2,37	6,66475
MDG-b_24_n2000_m200	2,22	5,96563
MDG-b_25_n2000_m200	2,19	5,97550
MDG-b_26_n2000_m200	2,40	5,96973
MDG-b_27_n2000_m200	2,31	5,96078
MDG-b_28_n2000_m200	2,53	5,99388
MDG-b_29_n2000_m200	2,76	5,90010
MDG-b_30_n2000_m200	2,19	7,05282
MDG-c_1_n3000_m300	2,18	13,40250
MDG-c_2_n3000_m300	2,45	13,64720
MDG-c_8_n3000_m400	1,74	26,38530
MDG-c_9_n3000_m400	1,81	23,31890
MDG-c_10_n3000_m400	2,05	23,98740
MDG-c_13_n3000_m500	1,63	38,72420
MDG-c_14_n3000_m500	1,64	39,36490
MDG-c_15_n3000_m500	1,56	40,33130
MDG-c_19_n3000_m600	1,42	58,33060
MDG-c_20_n3000_m600	1,48	57,41410

Tabla 4: Resultados AGG-posicion

Algoritmo Genético Estacionario Uniforme		
Caso	**Desv**	**Tiempo**
MDG-a_1_n500_m50	5,19	1,24700
MDG-a_2_n500_m50	5,13	1,23490
MDG-a_3_n500_m50	6,18	1,31716
MDG-a_4_n500_m50	4,12	1,23839
MDG-a_5_n500_m50	3,72	1,23140
MDG-a_6_n500_m50	3,10	1,22887
MDG-a_7_n500_m50	6,81	1,17513
MDG-a_8_n500_m50	4,88	1,22157
MDG-a_9_n500_m50	3,26	1,22962
MDG-a_10_n500_m50	4,75	1,21637
MDG-b_21_n2000_m200	5,60	12,09380
MDG-b_22_n2000_m200	4,90	14,16980
MDG-b_23_n2000_m200	4,25	12,09540
MDG-b_24_n2000_m200	4,35	11,74830
MDG-b_25_n2000_m200	4,00	13,95500
MDG-b_26_n2000_m200	3,94	14,34370
MDG-b_27_n2000_m200	4,02	12,72640
MDG-b_28_n2000_m200	4,87	11,03590
MDG-b_29_n2000_m200	3,98	12,71510
MDG-b_30_n2000_m200	4,38	12,70990
MDG-c_1_n3000_m300	4,98	25,88460
MDG-c_2_n3000_m300	4,78	30,54290
MDG-c_8_n3000_m400	3,47	45,42600
MDG-c_9_n3000_m400	4,03	48,68960
MDG-c_10_n3000_m400	3,65	47,76910
MDG-c_13_n3000_m500	3,24	63,59920
MDG-c_14_n3000_m500	3,30	62,87610
MDG-c_15_n3000_m500	3,73	66,43930
MDG-c_19_n3000_m600	2,97	88,10410
MDG-c_20_n3000_m600	2,99	89,40250

Tabla 5: Resultados AGE-uniforme

Algoritmo Genético Estacionario Posición		
Caso	**Desv**	**Tiempo**
MDG-a_1_n500_m50	12,05	0,821339
MDG-a_2_n500_m50	13,38	0,783013
MDG-a_3_n500_m50	11,73	0,793680
MDG-a_4_n500_m50	12,92	0,790551
MDG-a_5_n500_m50	10,63	0,794508
MDG-a_6_n500_m50	11,26	0,803923
MDG-a_7_n500_m50	12,01	0,784719
MDG-a_8_n500_m50	10,06	0,787745
MDG-a_9_n500_m50	10,77	0,777115
MDG-a_10_n500_m50	12,31	0,791747
MDG-b_21_n2000_m200	8,37	6,986230
MDG-b_22_n2000_m200	8,52	5,979240
MDG-b_23_n2000_m200	8,33	6,991070
MDG-b_24_n2000_m200	8,31	5,985400
MDG-b_25_n2000_m200	8,18	5,953450
MDG-b_26_n2000_m200	7,77	6,550930
MDG-b_27_n2000_m200	8,05	6,354130
MDG-b_28_n2000_m200	8,25	5,980130
MDG-b_29_n2000_m200	8,82	5,952390
MDG-b_30_n2000_m200	7,98	6,944240
MDG-c_1_n3000_m300	7,19	15,23940
MDG-c_2_n3000_m300	7,06	14,49210
MDG-c_8_n3000_m400	6,42	24,42240
MDG-c_9_n3000_m400	6,18	25,50000
MDG-c_10_n3000_m400	5,95	25,19830
MDG-c_13_n3000_m500	5,22	36,73110
MDG-c_14_n3000_m500	5,22	38,16990
MDG-c_15_n3000_m500	5,45	39,71850
MDG-c_19_n3000_m600	4,86	52,45980
MDG-c_20_n3000_m600	4,78	54,92730

Tabla 6: Resultados AGE-posicion

11.2.4. Algoritmo Memético

Algoritmo Memético (10,1.0)		
Caso	**Desv**	**Tiempo**
MDG-a_1_n500_m50	5,73	0,9600
MDG-a_2_n500_m50	6,17	0,9432
MDG-a_3_n500_m50	4,92	0,9652
MDG-a_4_n500_m50	5,28	0,9502
MDG-a_5_n500_m50	6,12	0,9410
MDG-a_6_n500_m50	5,15	0,9308
MDG-a_7_n500_m50	5,52	0,9514
MDG-a_8_n500_m50	4,76	0,9153
MDG-a_9_n500_m50	5,83	0,9425
MDG-a_10_n500_m50	6,50	0,9690
MDG-b_21_n2000_m200	4,20	10,6955
MDG-b_22_n2000_m200	4,23	10,1560
MDG-b_23_n2000_m200	4,08	10,3686
MDG-b_24_n2000_m200	4,40	10,5511
MDG-b_25_n2000_m200	3,93	10,2370
MDG-b_26_n2000_m200	4,07	10,3353
MDG-b_27_n2000_m200	3,63	10,6893
MDG-b_28_n2000_m200	3,92	10,7476
MDG-b_29_n2000_m200	4,10	10,4051
MDG-b_30_n2000_m200	4,28	10,5951
MDG-c_1_n3000_m300	3,38	22,1768
MDG-c_2_n3000_m300	3,35	22,4184
MDG-c_8_n3000_m400	3,04	29,3499
MDG-c_9_n3000_m400	2,96	29,6037
MDG-c_10_n3000_m400	3,03	29,7376
MDG-c_13_n3000_m500	2,39	37,9994
MDG-c_14_n3000_m500	2,20	37,3767
MDG-c_15_n3000_m500	2,38	37,3277
MDG-c_19_n3000_m600	2,11	46,6057
MDG-c_20_n3000_m600	2,11	45,9050

Tabla 7: Resultados AM-(10,1.0)

Algoritmo Memético (10,0.1)		
Caso	**Desv**	**Tiempo**
MDG-a_1_n500_m50	2,42	1,3902
MDG-a_2_n500_m50	1,72	1,5007
MDG-a_3_n500_m50	2,85	1,2058
MDG-a_4_n500_m50	1,42	1,6619
MDG-a_5_n500_m50	2,39	1,3691
MDG-a_6_n500_m50	1,51	1,8160
MDG-a_7_n500_m50	1,36	1,4832
MDG-a_8_n500_m50	1,81	1,4420
MDG-a_9_n500_m50	1,71	1,8915
MDG-a_10_n500_m50	2,54	1,5691
MDG-b_21_n2000_m200	1,88	26,7251
MDG-b_22_n2000_m200	2,03	27,3707
MDG-b_23_n2000_m200	2,00	20,7778
MDG-b_24_n2000_m200	1,84	26,9987
MDG-b_25_n2000_m200	2,15	27,5935
MDG-b_26_n2000_m200	1,79	24,4421
MDG-b_27_n2000_m200	1,63	26,1138
MDG-b_28_n2000_m200	2,06	25,9475
MDG-b_29_n2000_m200	1,93	28,6723
MDG-b_30_n2000_m200	1,75	30,0896
MDG-c_1_n3000_m300	1,31	63,6325
MDG-c_2_n3000_m300	1,61	68,9938
MDG-c_8_n3000_m400	1,22	116,2590
MDG-c_9_n3000_m400	1,63	100,8560
MDG-c_10_n3000_m400	1,46	85,2201
MDG-c_13_n3000_m500	1,38	132,2540
MDG-c_14_n3000_m500	1,00	163,7750
MDG-c_15_n3000_m500	1,07	120,2420
MDG-c_19_n3000_m600	0,91	189,8850
MDG-c_20_n3000_m600	1,09	162,3430

Tabla 8: Resultados AM-(10,0.1)

Algoritmo Memético (10,0.1) Mejores		
Caso	**Desv**	**Tiempo**
MDG-a_1_n500_m50	0,19	2,1148
MDG-a_2_n500_m50	1,26	2,4121
MDG-a_3_n500_m50	2,02	1,8862
MDG-a_4_n500_m50	1,77	2,4470
MDG-a_5_n500_m50	1,57	2,5367
MDG-a_6_n500_m50	1,64	2,0857
MDG-a_7_n500_m50	1,08	2,0600
MDG-a_8_n500_m50	1,59	2,2310
MDG-a_9_n500_m50	2,00	2,2897
MDG-a_10_n500_m50	1,86	1,5884
MDG-b_21_n2000_m200	2,06	32,2627
MDG-b_22_n2000_m200	1,81	31,7925
MDG-b_23_n2000_m200	1,92	38,8069
MDG-b_24_n2000_m200	1,62	31,8356
MDG-b_25_n2000_m200	1,74	39,0579
MDG-b_26_n2000_m200	2,13	42,0403
MDG-b_27_n2000_m200	1,65	38,5389
MDG-b_28_n2000_m200	1,54	35,7289
MDG-b_29_n2000_m200	1,10	50,1483
MDG-b_30_n2000_m200	1,56	34,0905
MDG-c_1_n3000_m300	1,09	131,4760
MDG-c_2_n3000_m300	1,40	83,4747
MDG-c_8_n3000_m400	1,35	142,3760
MDG-c_9_n3000_m400	1,05	188,6070
MDG-c_10_n3000_m400	1,42	134,5760
MDG-c_13_n3000_m500	1,13	178,8560
MDG-c_14_n3000_m500	0,79	189,8600
MDG-c_15_n3000_m500	1,05	182,9620
MDG-c_19_n3000_m600	0,91	234,7920
MDG-c_20_n3000_m600	1,56	360,2740

Tabla 9: Resultados AM-(10,0.1,mej)

11.2.5. Algoritmo Enfriamiento Simulado

Algoritmo Enfriamiento Simulado		
Caso	**Desv**	**Tiempo**
MDG-a_1_n500_m50	3,46	0,004115
MDG-a_2_n500_m50	3,75	0,001848
MDG-a_3_n500_m50	2,89	0,002818
MDG-a_4_n500_m50	2,33	0,003210
MDG-a_5_n500_m50	2,24	0,003399
MDG-a_6_n500_m50	2,75	0,003260
MDG-a_7_n500_m50	4,31	0,002906
MDG-a_8_n500_m50	3,48	0,001874
MDG-a_9_n500_m50	3,07	0,003941
MDG-a_10_n500_m50	3,65	0,002082
MDG-b_21_n2000_m200	1,60	0,107855
MDG-b_22_n2000_m200	1,46	0,119202
MDG-b_23_n2000_m200	1,47	0,132320
MDG-b_24_n2000_m200	1,86	0,082142
MDG-b_25_n2000_m200	1,31	0,150983
MDG-b_26_n2000_m200	1,74	0,109508
MDG-b_27_n2000_m200	1,77	0,211285
MDG-b_28_n2000_m200	1,32	0,161895
MDG-b_29_n2000_m200	1,83	0,107835
MDG-b_30_n2000_m200	1,64	0,104068
MDG-c_1_n3000_m300	1,10	0,403152
MDG-c_2_n3000_m300	1,31	0,418322
MDG-c_8_n3000_m400	0,64	0,532294
MDG-c_9_n3000_m400	0,89	0,574967
MDG-c_10_n3000_m400	0,93	0,605257
MDG-c_13_n3000_m500	0,78	0,750593
MDG-c_14_n3000_m500	0,58	0,843429
MDG-c_15_n3000_m500	0,47	0,747395
MDG-c_19_n3000_m600	0,54	0,897482
MDG-c_20_n3000_m600	0,58	0,878835

Tabla 10: Resultados ES

11.2.6. Algoritmo Enfriamiento Simulado inteligente

Algoritmo Enfriamiento Simulado inteligente		
Caso	**Desv**	**Tiempo**
MDG-a_1_n500_m50	2,19	0,001577
MDG-a_2_n500_m50	1,96	0,002493
MDG-a_3_n500_m50	1,27	0,001479
MDG-a_4_n500_m50	2,56	0,001835
MDG-a_5_n500_m50	2,40	0,002344
MDG-a_6_n500_m50	4,92	0,001188
MDG-a_7_n500_m50	2,49	0,001808
MDG-a_8_n500_m50	2,66	0,001884
MDG-a_9_n500_m50	3,34	0,001643
MDG-a_10_n500_m50	1,81	0,002509
MDG-b_21_n2000_m200	1,10	0,047460
MDG-b_22_n2000_m200	0,80	0,071476
MDG-b_23_n2000_m200	0,92	0,053613
MDG-b_24_n2000_m200	1,12	0,061628
MDG-b_25_n2000_m200	1,26	0,049121
MDG-b_26_n2000_m200	1,06	0,071879
MDG-b_27_n2000_m200	1,29	0,056513
MDG-b_28_n2000_m200	0,81	0,064387
MDG-b_29_n2000_m200	0,90	0,074661
MDG-b_30_n2000_m200	1,13	0,058523
MDG-c_1_n3000_m300	0,55	0,175006
MDG-c_2_n3000_m300	0,63	0,276090
MDG-c_8_n3000_m400	0,48	0,365444
MDG-c_9_n3000_m400	0,73	0,277408
MDG-c_10_n3000_m400	0,74	0,194103
MDG-c_13_n3000_m500	0,42	0,429891
MDG-c_14_n3000_m500	0,38	0,705409
MDG-c_15_n3000_m500	0,29	0,385588
MDG-c_19_n3000_m600	0,35	0,933100
MDG-c_20_n3000_m600	0,43	0,722338

Tabla 11: Resultados ES-inte

11.2.7. Algoritmo de Búsqueda Multiarranque Básica

Algoritmo Búsqueda Multiarranque Básica		
Caso	**Desv**	**Tiempo**
MDG-a_1_n500_m50	1,31	0,047933
MDG-a_2_n500_m50	1,18	0,049365
MDG-a_3_n500_m50	0,80	0,047563
MDG-a_4_n500_m50	1,48	0,048293
MDG-a_5_n500_m50	1,67	0,047551
MDG-a_6_n500_m50	0,96	0,047142
MDG-a_7_n500_m50	1,00	0,048774
MDG-a_8_n500_m50	0,48	0,049169
MDG-a_9_n500_m50	0,88	0,049144
MDG-a_10_n500_m50	1,01	0,055212
MDG-b_21_n2000_m200	0,90	0,524166
MDG-b_22_n2000_m200	0,93	0,531071
MDG-b_23_n2000_m200	0,98	0,499756
MDG-b_24_n2000_m200	1,10	0,516646
MDG-b_25_n2000_m200	0,92	0,492388
MDG-b_26_n2000_m200	1,01	0,553096
MDG-b_27_n2000_m200	0,96	0,508929
MDG-b_28_n2000_m200	0,92	0,581489
MDG-b_29_n2000_m200	1,08	0,529014
MDG-b_30_n2000_m200	0,98	0,543041
MDG-c_1_n3000_m300	0,98	1,164430
MDG-c_2_n3000_m300	1,10	1,021500
MDG-c_8_n3000_m400	0,78	1,395820
MDG-c_9_n3000_m400	0,74	1,438060
MDG-c_10_n3000_m400	0,92	1,412480
MDG-c_13_n3000_m500	0,60	1,884740
MDG-c_14_n3000_m500	0,62	1,875270
MDG-c_15_n3000_m500	0,53	1,943040
MDG-c_19_n3000_m600	0,52	2,338360
MDG-c_20_n3000_m600	0,55	2,654130

Tabla 12: Resultados BMB

11.2.8. Algoritmo de Búsqueda Local Reiterada

Algoritmo Búsqueda Local Reiterada		
Caso	**Desv**	**Tiempo**
MDG-a_1_n500_m50	2,03	0,033767
MDG-a_2_n500_m50	0,90	0,034301
MDG-a_3_n500_m50	1,27	0,033301
MDG-a_4_n500_m50	1,38	0,034187
MDG-a_5_n500_m50	1,59	0,033240
MDG-a_6_n500_m50	1,20	0,034124
MDG-a_7_n500_m50	1,07	0,035086
MDG-a_8_n500_m50	1,29	0,034505
MDG-a_9_n500_m50	0,26	0,035406
MDG-a_10_n500_m50	1,24	0,034614
MDG-b_21_n2000_m200	0,66	0,316103
MDG-b_22_n2000_m200	0,64	0,393328
MDG-b_23_n2000_m200	0,89	0,339765
MDG-b_24_n2000_m200	0,51	0,352158
MDG-b_25_n2000_m200	0,70	0,322055
MDG-b_26_n2000_m200	0,52	0,289770
MDG-b_27_n2000_m200	1,08	0,292923
MDG-b_28_n2000_m200	0,56	0,294567
MDG-b_29_n2000_m200	0,61	0,296172
MDG-b_30_n2000_m200	0,72	0,291735
MDG-c_1_n3000_m300	0,25	0,725300
MDG-c_2_n3000_m300	0,55	0,606528
MDG-c_8_n3000_m400	0,34	0,995152
MDG-c_9_n3000_m400	0,43	0,864283
MDG-c_10_n3000_m400	0,46	0,819254
MDG-c_13_n3000_m500	0,35	1,066870
MDG-c_14_n3000_m500	0,19	1,195110
MDG-c_15_n3000_m500	0,10	1,101180
MDG-c_19_n3000_m600	0,26	1,358000
MDG-c_20_n3000_m600	0,33	1,507410

Tabla 13: Resultados ILS

11.2.9. Algoritmo de Búsqueda Local Reiterada con ES

Algoritmo Búsqueda Local Reiterada con ES		
Caso	**Desv**	**Tiempo**
MDG-a_1_n500_m50	0,77	0,028409
MDG-a_2_n500_m50	2,44	0,027653
MDG-a_3_n500_m50	2,11	0,028456
MDG-a_4_n500_m50	1,58	0,027744
MDG-a_5_n500_m50	1,93	0,024637
MDG-a_6_n500_m50	2,27	0,026710
MDG-a_7_n500_m50	1,46	0,024261
MDG-a_8_n500_m50	1,63	0,026385
MDG-a_9_n500_m50	2,18	0,024964
MDG-a_10_n500_m50	1,40	0,024995
MDG-b_21_n2000_m200	2,38	0,221873
MDG-b_22_n2000_m200	1,90	0,216851
MDG-b_23_n2000_m200	2,18	0,217467
MDG-b_24_n2000_m200	2,22	0,216852
MDG-b_25_n2000_m200	2,17	0,217344
MDG-b_26_n2000_m200	2,25	0,217916
MDG-b_27_n2000_m200	2,14	0,216792
MDG-b_28_n2000_m200	1,82	0,215707
MDG-b_29_n2000_m200	2,20	0,216092
MDG-b_30_n2000_m200	1,95	0,216012
MDG-c_1_n3000_m300	2,09	0,371441
MDG-c_2_n3000_m300	2,31	0,379389
MDG-c_8_n3000_m400	1,69	0,521789
MDG-c_9_n3000_m400	1,89	0,522062
MDG-c_10_n3000_m400	1,96	0,528119
MDG-c_13_n3000_m500	1,52	0,696165
MDG-c_14_n3000_m500	1,35	0,696808
MDG-c_15_n3000_m500	1,50	0,695371
MDG-c_19_n3000_m600	1,36	0,876721
MDG-c_20_n3000_m600	1,35	0,880387

Tabla 14: Resultados ILS-ES

11.2.10. Algoritmo de Búsqueda Local Reiterada con ES inteligente

Algoritmo Búsqueda Local Reiterada con ES inteligente		
Caso	**Desv**	**Tiempo**
MDG-a_1_n500_m50	1,81	0,009949
MDG-a_2_n500_m50	0,62	0,013450
MDG-a_3_n500_m50	0,59	0,010232
MDG-a_4_n500_m50	1,04	0,012122
MDG-a_5_n500_m50	1,90	0,012053
MDG-a_6_n500_m50	0,69	0,013853
MDG-a_7_n500_m50	1,54	0,012055
MDG-a_8_n500_m50	2,23	0,010995
MDG-a_9_n500_m50	0,43	0,013792
MDG-a_10_n500_m50	1,35	0,010735
MDG-b_21_n2000_m200	0,44	0,259702
MDG-b_22_n2000_m200	0,50	0,268967
MDG-b_23_n2000_m200	0,69	0,269102
MDG-b_24_n2000_m200	0,80	0,257674
MDG-b_25_n2000_m200	0,67	0,254422
MDG-b_26_n2000_m200	0,82	0,265124
MDG-b_27_n2000_m200	0,75	0,258520
MDG-b_28_n2000_m200	0,63	0,257008
MDG-b_29_n2000_m200	0,75	0,255919
MDG-b_30_n2000_m200	0,39	0,255056
MDG-c_1_n3000_m300	0,43	0,520209
MDG-c_2_n3000_m300	0,55	0,527669
MDG-c_8_n3000_m400	0,39	0,788177
MDG-c_9_n3000_m400	0,51	0,781022
MDG-c_10_n3000_m400	0,48	0,866956
MDG-c_13_n3000_m500	0,28	1,149970
MDG-c_14_n3000_m500	0,25	1,103060
MDG-c_15_n3000_m500	0,19	1,094690
MDG-c_19_n3000_m600	0,26	1,484650
MDG-c_20_n3000_m600	0,30	1,442260

Tabla 15: Resultados ILS-inte

11.2.11. Resultados globales

Algoritmo Greedy	**Desv**	**Tiempo**
Greedy	14,71	0,73
BL_PM	1,2	0,31
AGG-uniforme	1,41	131,69
AGG-posicion	2,32	13,50
AGE-uniforme	4,29	23,62
AGE-posicion	8,6	13,28
AM-(10,1.0)	4,13	15,09
AM-(10,0.1)	1,72	49,45
AM-(10,0.1,mej)	1,46	74,11
ES	1,86	0,27
ES-inte	1,37	0,17
BMB	0.93	0,76
ILS	0.75	0,46
ILS-ES	1,87	0,29
ILS-ES-inte	0.74	0,42

Tabla 16: Resultados Globales

11.3. Análisis de resultados

11.3.1. Algoritmos sin múltiples trayectorias

Para este apartado se van a utilizar las siguientes gráficas:

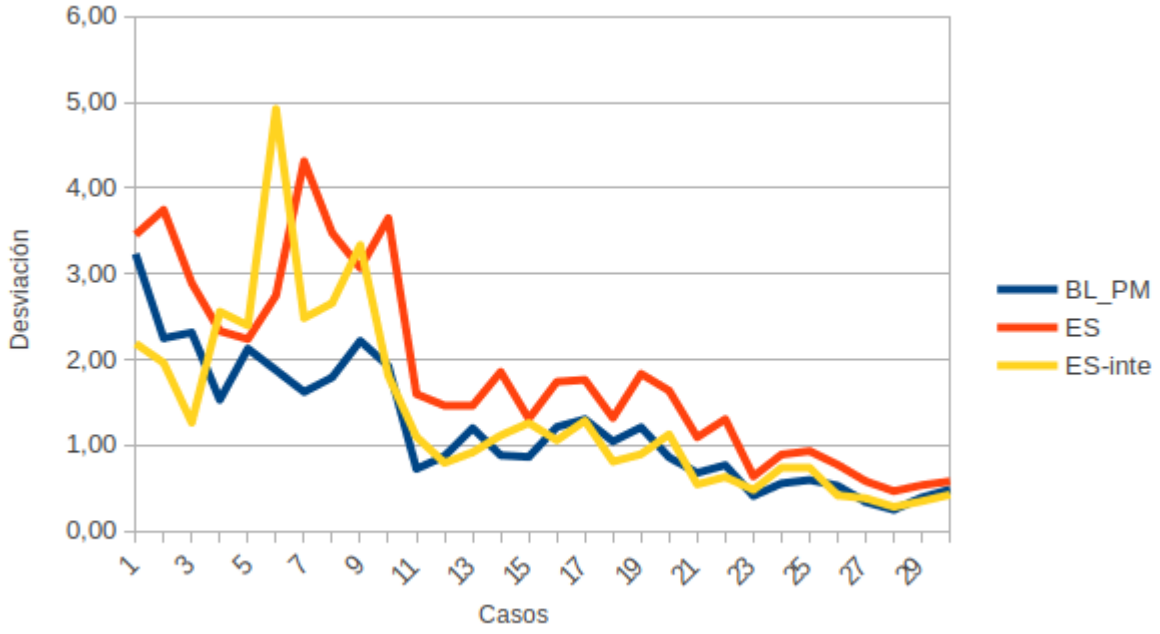

Figura 1: Evolución de la desviación - Algoritmos sin múltiples trayectorias

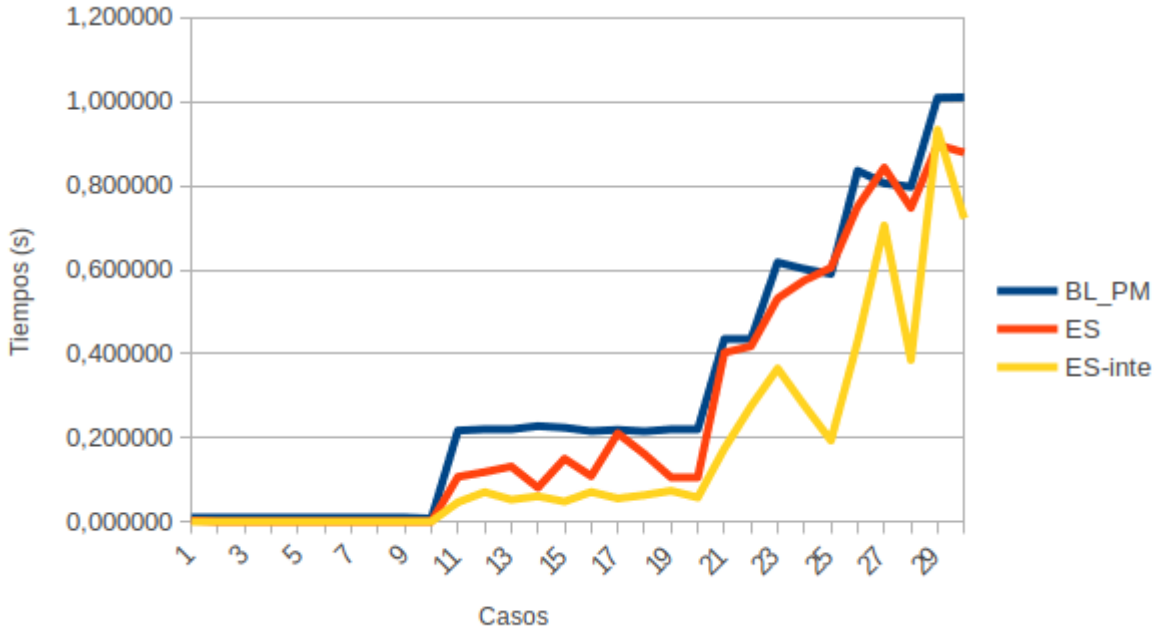

Figura 2: Evolución del tiempo - Algoritmos sin múltiples trayectorias

En este punto se va a analizar el comportamiento de los algoritmos que no utilizan múltiples

trayectorias.

En cuanto a la desviación, con la gráfica [1] se puede deducir que no hay grandes cambios entre los distintos algoritmos. El algoritmo BL_PM es el más estable, a diferencia de los algoritmos de enfriamiento simulado que en algunos casos tiene muchas desviación.

Una vez vista la desviación, ahora nos vamos a centrar en los tiempos obtenidos, con la gráfica [2] se pueden deducir lo siguiente:

Los algoritmos de enfriamiento simulado son más rápidos que los algoritmos de búsqueda local, aunque en el caso del algoritmo de enfriamiento simulado que no es inteligente no hay tanta diferencia con el algoritmo de búsqueda local. Esta diferencia de tiempo se debe a que la creación de las nuevas soluciones es más rápida en el caso de los algoritmos de enfriamiento simulado que en el caso de los algoritmos de búsqueda local.

Viendo los resultados de estos algoritmos en desviación y en tiempos, en mi opinión el mejor algoritmo para los casos utilizados es el algoritmo de enfriamiento simulado inteligente.

11.3.2. Influencia de la posición inicial en los algoritmos de búsqueda local

Para este apartado se van a utilizar las siguientes gráficas:

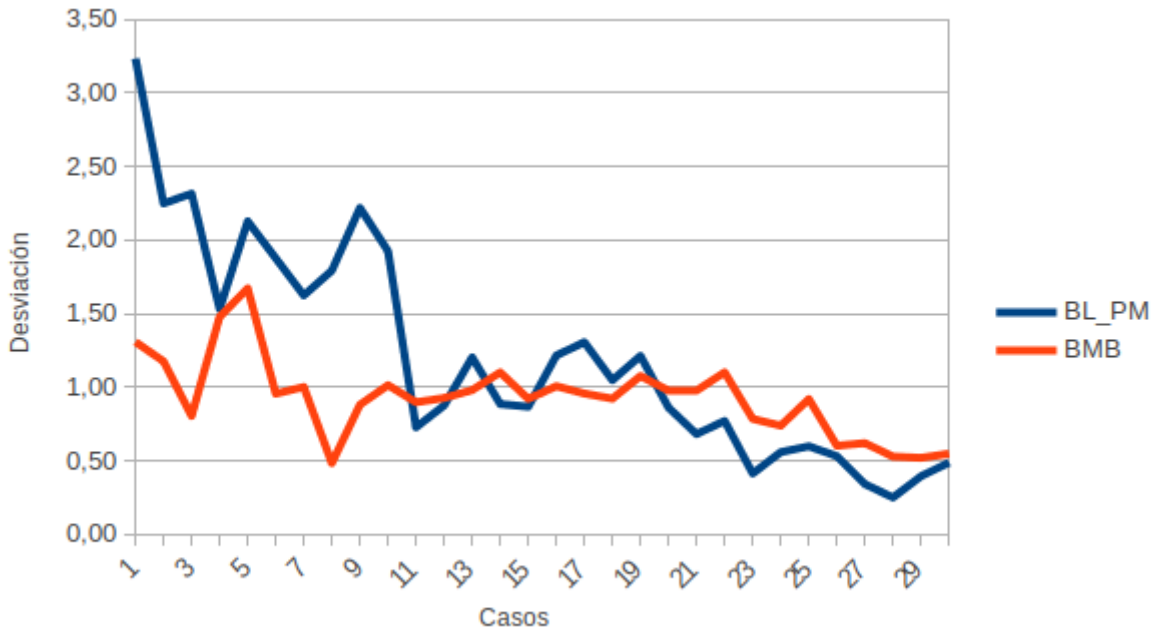

Figura 3: Evolución de la desviación - Influencia de la posición inicial en el BL

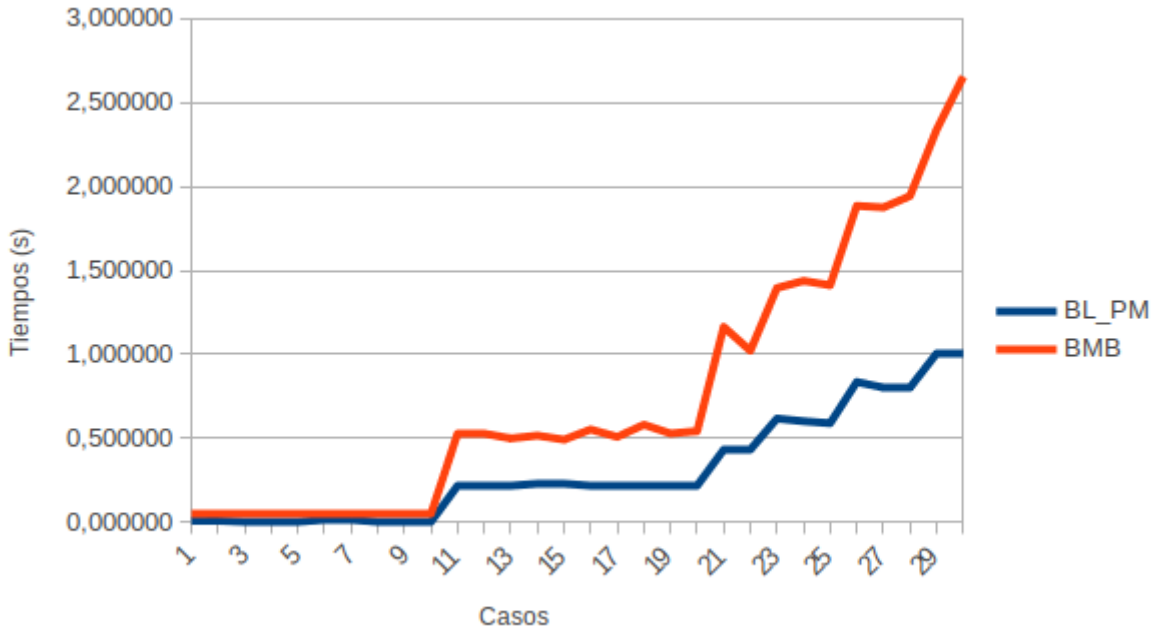

Figura 4: Evolución del tiempo - Influencia de la posición inicial en el BL

En cuanto a la desviación, con la gráfica [3] se puede comprobar como el algoritmo BMB es más estable en los resultados que el algoritmo de BL. Esto se debe a que el algoritmo BMB utiliza varias soluciones iniciales, y esto reduce la dependencia del algoritmo BL de la solución inicial. Y como se puede comprobar el algoritmo BMB obtiene resultados mejores o iguales, dependiendo del caso; que el algoritmo BL, y realizando menos evaluaciones en la ejecución del BL en cada iteración del algoritmo BMB.

Una vez vista la desviación, ahora nos vamos a centrar en los tiempos obtenidos, con la gráfica [4] se puede comprobar como el algoritmo BMB es siempre más lento que el algoritmo BL. Esto es debido a que en cada iteración del algoritmo BMB se deben hacer muchas copias de elementos para guardar la mejor solución, y además se debe iniciar muchas veces el algoritmo BL, por lo que la diferencia de tiempo es debido a esto. Si se consiguiese optimizar esta parte, el algoritmo BMB sería una mejor opción que el algoritmo BL ya que es más fiable al dar unos resultados más estables.

11.3.3. Comparación de los algoritmos de múltiples trayectorias con el mejor algoritmo de única trayectoria

Para este apartado se van a utilizar las siguientes gráficas:

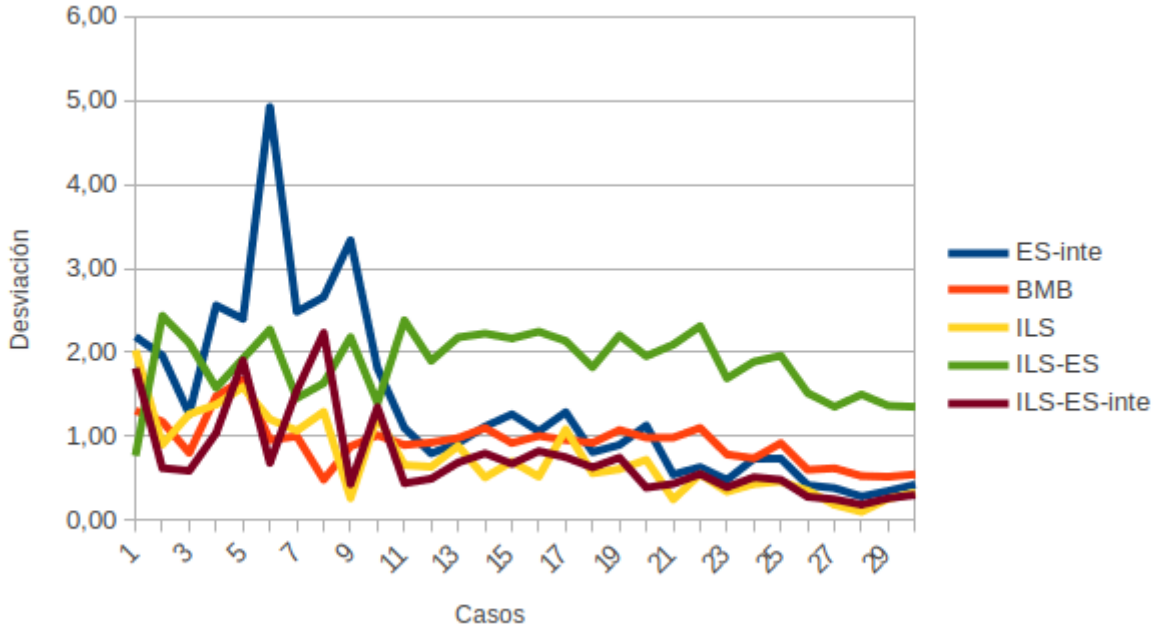

Figura 5: Evolución de la desviación - Comparación de los algoritmos de múltiples y única trayectoria

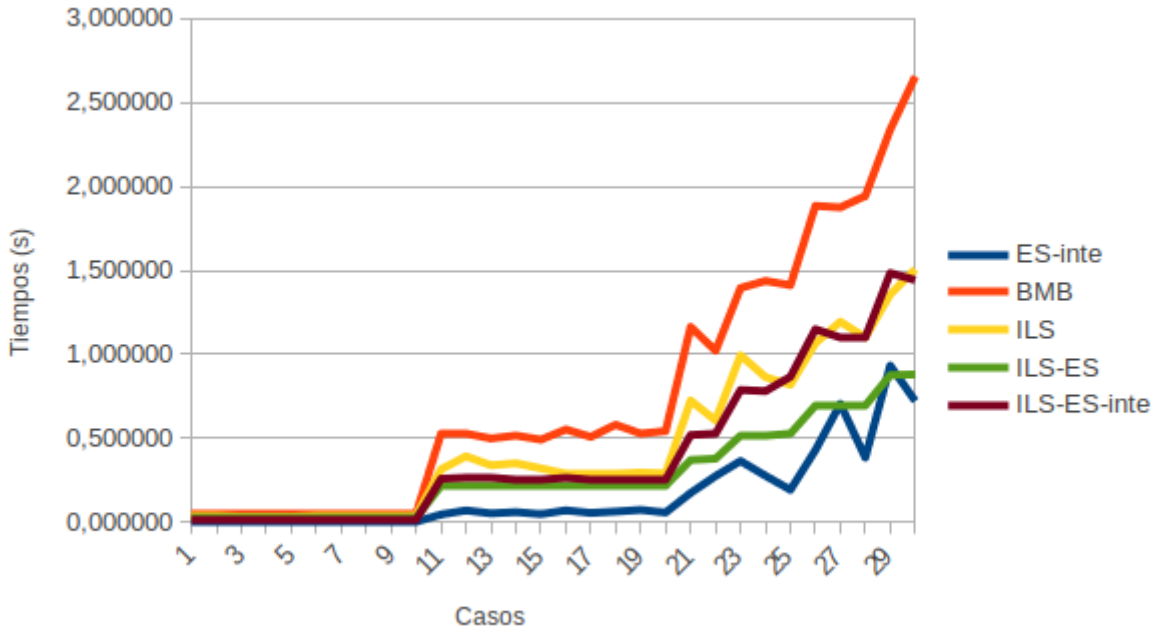

Figura 6: Evolución del tiempo - Comparación de los algoritmos de múltiples y única trayectoria

En cuanto a la desviación, con la gráfica [5] se puede comprobar como todos los algoritmos de múltiple trayectoria son más estables en los resultados que el mejor algoritmo de una única trayectoria. En general también dan mejores resultados salvo el ILS-ES y en algunas ocasiones el BMB.

Una vez vista la desviación, ahora nos vamos a centrar en los tiempos obtenidos, con la gráfica [6] se puede comprobar lo mismo que se vio en la comparación del BMB y el BL, es decir, que los algoritmos de múltiples trayectorias, al tener que guardar la mejor solución y tener que iniciar varias veces los algoritmos de una única trayectoria, tardan siempre más que los algoritmo de un única trayectoria.

11.3.4. Comparación entre los algoritmos de múltiples trayectorias

Para este apartado se van a utilizar las siguientes gráficas:

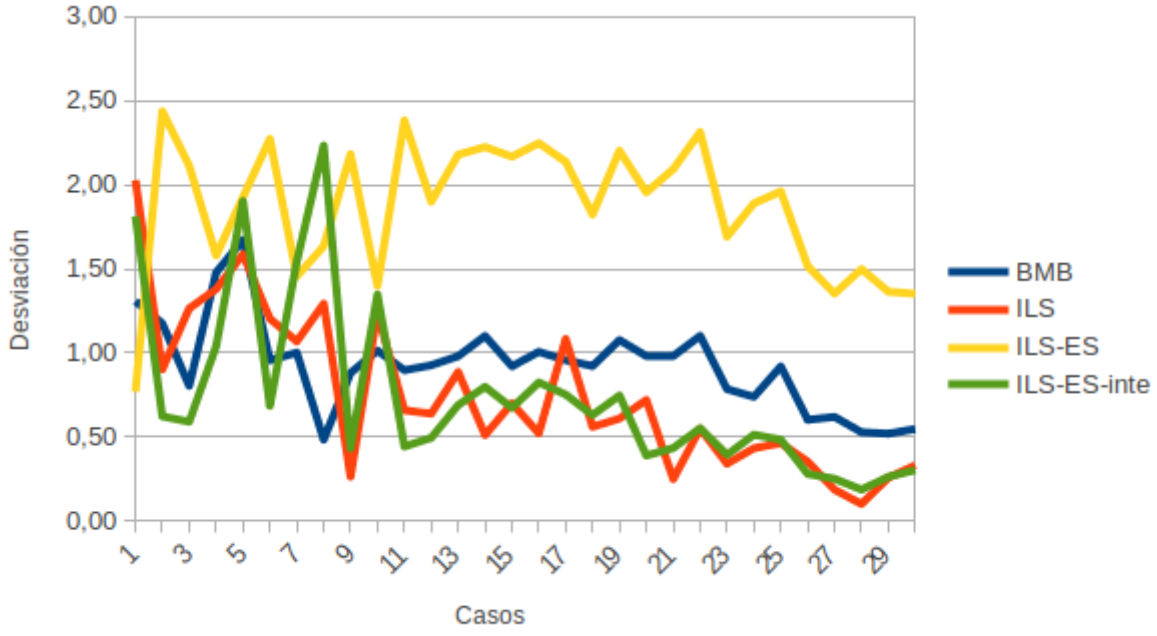

Figura 7: Evolución de la desviación - Comparación de los algoritmos de múltiples trayectorias

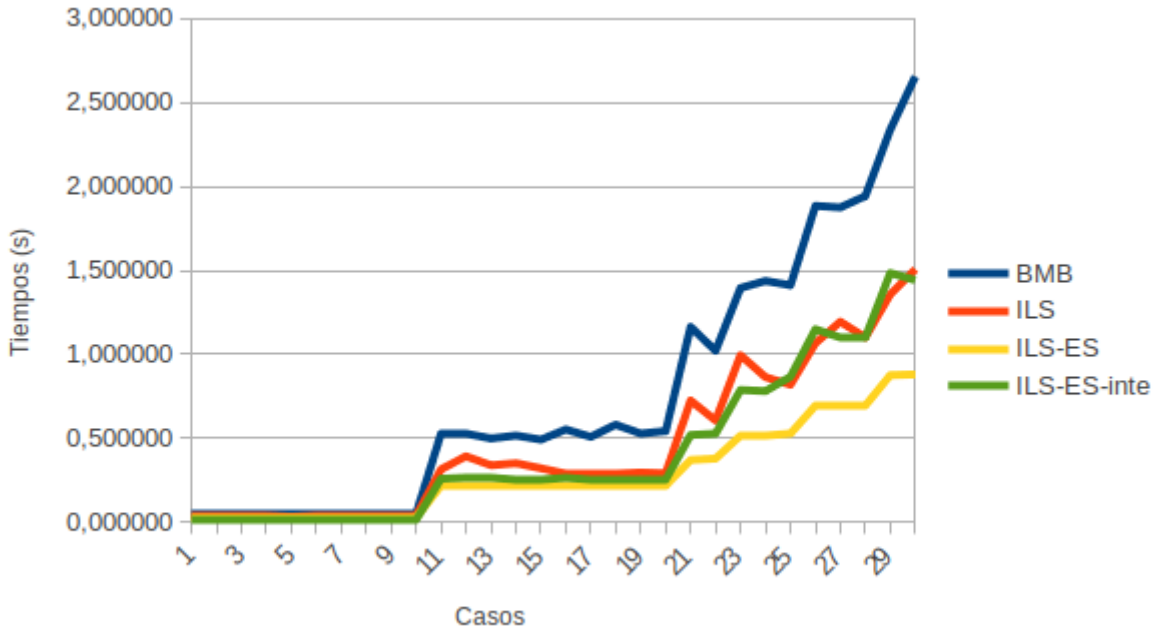

Figura 8: Evolución del tiempo - Comparación de los algoritmos de múltiples trayectorias

En cuanto a la desviación, con la gráfica [7] se puede deducir que los algoritmos de múltiples trayectorias que utilizan en cada iteración un algoritmo inteligente ,como puede ser BL_PM ó ES-inte; obtiene mejor resultados que los algoritmos que utilizan algoritmos no inteligentes, como ES.

Una vez vista la desviación, ahora nos vamos a centrar en los tiempos obtenidos, con la gráfica [8] se puede deducir que los algoritmos de múltiples trayectorias que utilizan el algoritmo ES o alguna de sus variantes son más rápidos que los algoritmos que utilizan el algoritmo BL. Y además se puede deducir que el algoritmo BMB, que en cada iteración ejecuta el algoritmo BL sobre una solución aleatoria, siempre es más lento, que los algoritmos ILS, que en todas sus variantes en cada iteración se ejecuta el algoritmo BL ó ES sobre la mejor solución encontrada hasta el momento.

En resumen, por los resultados obtenidos es mejor opción utilizar algoritmos con múltiples trayectorias, como ILS, y que además utiliza un algoritmo inteligente para obtener la mejor solución.