Metaheurísticas(2019-2020)
Doble Grado en Ingeniería Informática y Matemáticas
Universidad de Granada

Técnicas de Búsqueda basadas en Poblaciones para el Problema de la Máxima Diversidad (MDP)

Alberto Jesús Durán López
DNI: 54142189-M
albduranlopez@gmail.com

Grupo Jueves, 17:30 - 19:30

26 de abril de 2020

Índice

1. Introducción

Durante el transcurso de la asignatura trabajaremos con el problema de la máxima diversidad *(Max Diversity Problem)*.
En particular, en esta práctica estudiaremos técnicas de Búsqueda basadas en Poblaciones para la resolución del problema en cuestión.
Comentaremos todos los pasos y problemas encontrados, detallando minuciosamente todos los detalles y solución a los mismos, comenzando con los dos tipos de Algoritmos Genéticos implementados y terminando con Algoritmos Meméticos.

Además, se incorporarán tablas para mostrar los resultados de todas las ejecuciones y gráficas para contrastar los modelos (optimización, costes y desviación).

2. Problema de la máxima Diversidad

El problema de la máxima diversidad (*maximum diversity problem*, MDP) es un problema de optimización combinatoria consistente en seleccionar un subconjunto de m elementos ($|M| = m$) de un conjunto inicial N de n elementos (con $n > m$) de forma que se maximice la diversidad entre los elementos escogidos.

El **MDP** se puede formular como:

$$\text{Maximizar: } z_{MS}(x) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{ij} x_i x_j$$

$$\text{Sujeto a: } \sum_{i=1}^{n} x_i = m \quad \text{con } x_i = \{0,1\},\ i = 1, ..., n \quad \text{donde:}$$

- x 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

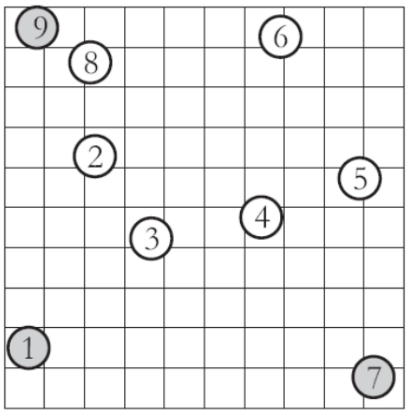


Figura 2.1: MDP-Maximum Diversity Problem

3. Datos y casos considerados

Las ejecuciones se han realizado en un ordenador `Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz, 16GB RAM, 512 SSD`.

Se utilizarán **30 casos** seleccionados de varios de los conjuntos de instancias disponible en la MDPLIB, 10 pertenecientes al grupo `GKD` con distancias Euclideas, n=500 y m=50 *(GKD-c_11_n500_m50 a GKD- c_20_n500_m50)*, 10 del grupo `MDG` con distancias reales en [0,1000], n=500 y m=50 *(MDG-b_1_n500_m50 a MDG-b_10_n500_m50)*; y otras 10 del grupo `MDG` con distancias enteras en 0,10, n=2000 y m=200 *(MDG-a_31_n2000_m200 a MDG- a_40_n2000_m200)*.

Para la realización de la práctica usaremos el lenguaje de programación `C++` ya que se deben probar muchos ejemplos y la ejecución es más rápida al ser un lenguaje de programación compilado.
Todos los archivos se han compilado con la opción de optimización `-O2` y, además, la semilla usada para todas las ejecuciones ha sido 54142189, pudiéndose ésta indicar en la ejecución del archivo en cuestión.

4. Algoritmos Genéticos

4.1. Representación de las Poblaciones

Para la programación de nuestros Algoritmos Genéticos se han necesitado 2 poblaciones que hemos ido actualizando, la población actual y la población correspondiente a la siguiente generación. Por ello, se ha optado por su encapsulación en una clase que hemos llamado *población*. En ella, se han añadido las variables referentes a la mutación y al cruce, el tamaño de la población o número de cromosomas (que hemos llamado **n**), el índice del mejor cromosoma de la población (**best**) y, por último, una matriz de booleanos para representar las distintas soluciones del problema (cada una de tamaño m).

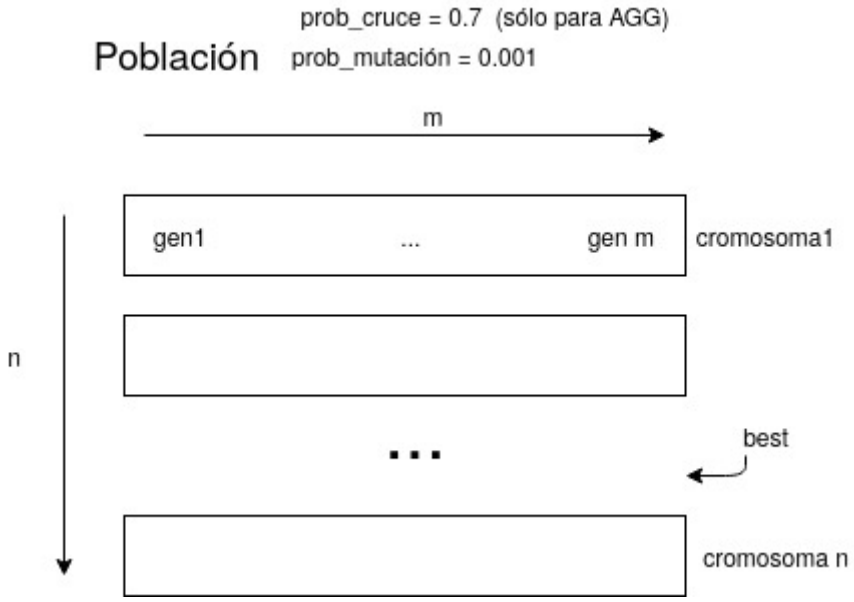


Figura 4.1: Class poblacion

4.2. Generación de Soluciones Iniciales

El esquema de la solución inicial varía respecto a la práctica anterior ya que para los Algoritmos Genéticos es necesaria la codificación binaria. Por tanto, cada cromosoma estará formado por valores booleanos 1 (cuando se toma el gen) o 0 (cuando no lo tomamos). Mostramos a continuación el algoritmo para la generación aleatoria de cada cromosoma de nuestra población, teniendo en cuenta que cada uno de los cromosomas debe cumplir las restricciones del problema MDP.

Datos: Solucion Inicial(n_1, m_1)
inicio
  vector<bool> cromosoma ;
  mientras *not salir* **hacer**
    $cromosoma[random\,\%n_1] \leftarrow true$;
    si n^o *de true de cromosoma* $= m_1$ **entonces**
      $salir \leftarrow true$;
    fin
  fin
  devolver *cromosoma*
fin

Algoritmo 1: Solución inicial

n_1 y m_1 son los índices del problema MDP obtenidos tras leer cada una de las instancias y necesarios para el cumplimiento de las restricciones.

4.3. Descripción de la Función Objetivo

Todos los AG implementados hacen uso de la misma función objetivo, la comentada en el punto (2) anterior y cuyo pseudocódigo mostramos a continuación. Bien es cierto que en el caso de los algoritmos meméticos se tiene que pasar a codificación entera, sin embargo, dicha función no se ve modificada.

Datos: ContribucionIndep(*integer ind, vector sel, matriz distancias)*
inicio
  $suma \leftarrow 0$;
  para *j in sel* **hacer**
    $suma \leftarrow sel[j] \cdot (distancias[ind][j] + suma)$;
  fin
  devolver *suma*
fin

Datos: CosteEstimado(*vector sel, matriz distancias)*
inicio
  $suma \leftarrow 0$;
  para *i in* $|sel|$ **hacer**
    si $sel[i] = true$ **entonces**
      $suma \leftarrow ContribucionIndep(i, sel, distancias) + suma$;
    fin
  fin
  devolver $suma/2$
fin

Algoritmo 2: Evaluar solución

4.4. Esquema de Evolución y de Reemplazamiento

Se han desarrollado un total de 4 algoritmos genéticos, dos variantes generacionales elitistas (AGG) y otras dos variantes estacionarias(AGE):

- **AGG**: En los Algoritmos Genéticos Generacionales se selecciona una población de padres del mismo tamaño que la población genética usando el operador de Selección, donde cruzaremos los distintos cromosomas mediantes torneos binarios.

  Por otro lado, están caracterizados por su esquema elitista, es decir, en cada generación se mantiene el mejor elemento de la población.

  Datos: ReemplazamientoAGG
  inicio
  | **si** $poblacion_{antigua}[best] \notin poblacion_{nueva}$ **entonces**
  | | $poblacion_{nueva}[peor] \leftarrow poblacion_{antigua}[best]$;
  | **fin**
  fin

  Algoritmo 3: Operador de Reemplazamiento para AGG

- **AGE**: Se caracterizan por tener un esquema de evolución donde únicamente se seleccionan dos padres que compiten con los 2 peores de la población.

  Los dos descendientes generados tras el cruce y la mutación sustituyen a los dos peores de la población actual (en caso de ser mejores que ellos), tal y como se muestra a continuación:

  Datos: ReemplazamientoAGE
  inicio
  | $peor_1 \leftarrow min\_element(poblacion)$;
  | $peor_2 \leftarrow next\_min\_element(poblacion)$;
  | **si** $Coste\ descendiente_1 > peor_1$ **entonces**
  | | *Intercambiarlos* ;
  | **fin**
  | **si** $Coste\ descendiente_2 > peor_2$ **entonces**
  | | *Intercambiarlos* ;
  | **fin**
  fin

  Algoritmo 4: Operador de Reemplazamiento para AGE

Por orden de llamada, los distintos operadores son: *Selección, Cruce, Mutación y Reemplazamiento*.
Tanto para los AGG como AGE se han desarrollado dos variantes: una con un cruce basado en posición y otro uniforme. Las 4 versiones implementadas comparten estos operadores, sin embargo, comentaremos todos los detalles en las siguientes secciones.

4.4.1. Operador de Selección AGG

Comenzamos comentando el operador de Selección de los AGG, caracterizado por realizar tantos torneos binarios como individuos existan en la población genética. Para ello, generamos índices aleatorios referentes a distintos cromosomas y los hacemos competir llamando a la función **Torneo Binario**. Posteriormente, se va añadiendo el ganador de cada torneo a una nueva población.

Datos: Torneo Binario(int r, int s)
  devolver $coste[r] > coste[s]$? r : s ;

Datos: Seleccion
inicio
  poblacion nueva ;
  para *i in* |*poblacion*| **hacer**
    ganador ← **TorneoBinario***(rand,rand)*;
    cromosoma ← *poblacion*[*ganador*] ;
    nueva ← *nueva* ∪ *cromosoma* ;
  fin
  devolver *nueva*
fin

Algoritmo 5: Operador de Selección

4.5. Operador de Selección AGE

En este operador, en vez de realizar tantos torneos binarios como individuos haya en la población, se aplicará únicamente 2 veces para elegir los dos padres que posteriormente serán cruzados. Hacemos uso de la función Torneo Binario y devolvemos la pareja de padres obtenidos:

Datos: Torneo Binario(int r, int s)
devolver $coste[r] > coste[s]$? r : s ;

Datos: Seleccion
inicio
  pair padres ;
  hacer{
    padres.first ← **TorneoBinario***(rand,rand)* ;
    padres.second ← **TorneoBinario***(rand,rand)* ;
  }**mientras** *padres.first* = *padres.second*
  devolver *padres*
fin

Algoritmo 6: Operador de Selección

4.6. Operador de Cruce Posicional

En este Operador tenemos una probabilidad de cruce de 0.7, por ello, hemos decidido implementar dos versiones.

- La primera, mediante una función que simula una distribución uniforme, generamos un número aleatorio para cada pareja y las cruzamos con probabilidad 0.7
- En la segunda, estimamos a priori el número de cruces haciendo uso de la *Esperanza Matemática*, donde dicho valor se calculará como $prob\_cruce \cdot |poblacion|$. Los cromosomas a cruzar serán obtenidos de forma aleatoria.

  En ambas versiones el cruce se realiza siempre por parejas consecutivas, el 1º con el 2º, el 3º con el 4º, tal y como se ha descrito en el Seminario 3.

Se han hecho pruebas con ambas versiones y se han obtenido unos ligeros mejores resultados en la primera versión ya que forzamos el recorrido de todos los cromosomas y no dependemos de la aleatoriedad en su elección, sin embargo perdemos eficiencia, ya que no hacemos uso de la *Esperanza Matemática*. Mostramos por ello el pseudocódigo asociado a la segunda versión: (la implementación de ambas versiones está íntegra en el proyecto entregado).

Datos: Cruce Posicional
inicio
  $cruces \leftarrow prob_{cruce} \cdot N/2$
  mientras $cruces > 0$ **hacer**
    elegir pareja (i, i+1) para cruzarlos
    $padre_{restos} \leftarrow random(Padre_i, Padre_{i+1})$;
    para *j in* $|padre_i|$ **hacer**
      si $Padre_i[j] = Padre_{i+1}[j]$ **entonces**
        $hijo_1[j] \leftarrow padre_i[j]$;
        $hijo_2[j] \leftarrow padre_i[j]$
      fin
      en otro caso
        $restos \leftarrow restos \cup padre_{restos}[j]$;
      fin
    fin
    rellenar aleat. hijo1 con restos ;
    rellenar aleat. hijo2 con restos ;
    cruces - - ;
  fin
fin

Algoritmo 7: Cruce Posicional

Destacamos que en cada cruce elegimos un padre aleatorio para después rellenar las posiciones de ambos hijos de forma aleatoria. En este operador se generan hijos factibles si los dos padres son factibles luego no se necesita reparar las soluciones. Sin embargo, es más disruptivo, comparte menos información de los padres y la convergencia puede ser más costosa.

4.7. Operador de cruce Uniforme

En el cruce Uniforme no tenemos el problema descrito anteriormente ya que la probabilidad de cruce es de 1, es decir, siempre se cruzan los dos padres. Sin embargo, este operador requiere reparador ya que las soluciones obtenidas pueden no ser factibles. Cruzamos siempre parejas de cromosomas consecutivas, el 1^o con el 2^o, 3^o con el 4^o ...etc. El reparador aplicado es el descrito en las diapositivas del Seminario 3.
Hay que tener en cuenta las siguientes consideraciones:

- Por motivos de eficiencia y para no generar un exceso de números aleatorios, cruzamos parejas de cromosomas consecutivas.
- Las dos variantes de nuestros Algoritmos Genéticos hacen uso de este cruce. En el caso de los AGG, se realiza un cruce uniforme por parejas de toda la población, mientras que en los AGE únicamente se cruzan dos padres. El cruce que mostramos es el que se realiza en los AGG, para los AGE se realiza el mismo cruce pero únicamente para la pareja que se cruza, no para toda la población.

Datos: Cruce Uniforme*(integer m)*
inicio
  $cruces \leftarrow prob_{cruce} \cdot N/2$;
  mientras *cruces*>0 **hacer**
    Elegir pareja de Cromosomas $(i, i+1)$;
    $padre_r = random(padre_i, padre_{i+1})$;
    $padre_s = random(padre_i, padre_{i+1})$;
    para j *in* $|padre_i|$ **hacer**
      si $Padre_i[j] = Padre_{i+1}[j]$ **entonces**
        $hijo_1[j] \leftarrow padre_i[j]$;
        $hijo_2[j] \leftarrow padre_i[j]$
      fin
      en otro caso
        //Rellenamos los hijos con los restos de un padre aleat. ;
        $hijo_1[j] \leftarrow padre_r[j]$;
        $hijo_2[j] \leftarrow padre_s[j]$;
      fin
    fin
    reparar*(*$hijo_1$*,m)*;
    reparar*(*$hijo_2$*,m)*;
    cruces - - ;
  fin
fin

Algoritmo 8: Cruce Uniforme

Recorremos ambos padres. En aquellas posiciones que contengan el mismo valor en ambos padres se mantienen en el hijo para así preservar las selecciones prometedores. En otro caso, las selecciones restantes serán rellenadas por los valores de uno de los padres.

4.8. Operador de Mutación

Para el operador de mutación consideramos una probabilidad de mutación por gen de 0.001. Para que nuestro problema siga cumpliendo las restricciones, intercambiaremos un gen x_i por otro x_j dentro de un mismo cromosoma/solución de nuestra población, comprobando que los valores a intercambiar sean opuestos, es decir, un 0 por un 1. Por tanto, las soluciones seguirán siendo factibles y no será necesaria ninguna comprobación ni reparación posterior.

Al igual que ocurre en el operador de cruce, uno de los problemas más costosos y más ineficientes de los algoritmos genéticos es la generación de números aleatorios. Como en este operador este problema está intensificado (ya que la prob_cruce es de 0.001), volvemos a diseñar una implementación basada en la *Esperanza Matemática* para así reducir en gran medida la cantidad de números aleatorios generados y, como consecuencia, reducir el tiempo de ejecución lo máximo posible.

El pseudocódigo correspondiente a dicho operador es el siguiente:

Datos: Mutacion
inicio
  total $\leftarrow$ *prob_mut* $\cdot$ n^o*genes total*;
  suma $\leftarrow$ *0* ;
  mientras *total* > 0 **hacer**
    $n_1 \leftarrow$ *índice para elegir cromosoma* ;
    $m_1 \leftarrow$ *índice para elegir gen* ;
    hacer {
     $m_2 \leftarrow$ *índice para elegir gen distinto a* m_1 ;
    } **mientras**(*los dos genes sean iguales*) ;
    Intercambiar genes ;
    total - - ;
  fin
fin

Algoritmo 9: Operador de Mutación

Generamos 3 números aleatorios. El primero n_1 para elegir el cromosoma y, los dos posteriores m_1 y m_2, para intercambiar dos genes dentro de éste. Para el intercambio de los genes comprobamos que sean del mismo cromosoma (para no violar las restricciones) y que dichos genes sean distintos (uno cero y el otro uno), como hemos comentado anteriormente.
El pseudocódigo anterior es el de los AGG ya que involucra a toda la población de cromosomas. El referente a AGE es una pequeña variante del anterior en el que únicamente se toman los dos padres.

5. Algoritmos Meméticos

Nuestro Algoritmo Memético consistirá en una técnica de hibridación de un Algoritmo Genético Generacional con una BL (la desarrollada en la práctica 1).
Nuestro problema MDP se complica ya que para aplicar la búsqueda local es necesario pasar de codificación binaria a codificación de conjuntos enteros, sin embargo, esto no supone ninguna modificación en el resto de funciones implementadas.

Se han realizado tres posibilidades de hibridación, combinando y manteniendo un equilibrio distinto entre exploración y explotación. Adelantamos que en todas ellas se usa la estrategia con el operador de cruce basado en posición ya que se han obtenido mejores resultados que con el uniforme.

El tamaño de la población del AGG cambia y pasará a tener 10 cromosomas. Sin embargo, las probabilidades de cruce y mutación se mantienen a 0.7 y 0.001, respectivamente.
Por otro lado, la búsqueda local se detendrá cuando realice 400 evaluaciones de vecinos distintos o no encuentre mejora en el entorno. El criterio de parada del AM consistirá en realizar 100.000 evaluaciones de la función objetivo, incluyendo además las de la BL.

- `AM-(10,1.0)`: Cada **10** generaciones aplicaremos Búsqueda Local sobre **todos los cromosomas** de la población. Predomina la exploración sobre la explotación, ya que la BL consume las evaluaciones muy rápido, sin embargo, no se deja a ningún cromosoma atrás y favorece la diversidad de las soluciones.
- `AM-(10,0.1)`: Cada **10** generaciones aplicaremos la Búsqueda Local sobre un **subconjunto de cromosomas** de la población seleccionando aleatoriamente con probabilidad de **0.1** para cada uno de ellos. Tando la exploración como la explotación predominan por igual.
- `AM-(10,0.1mej)`: Cada **10** generaciones aplicaremos Búsqueda Local sobre los **0.1**· **N mejores** cromosomas de la población actual. (N es el tamaño de ésta). Predomina más la explotación que la exploración, se busca a toda costa mejorar el resultado de la función objetivo aplicando BL sobre los cromosomas más prometedores, dejando atrás los que peores resultados arrojan.

Las tres variantes se han implementado en un mismo archivo `.cpp`. Se ha usado la estructura de control `switch` que ejecutará una u otra según el valor de la variable `tipo`. Como en nuestros algoritmos se ha optado por la encapsulación de la población en una clase, dicha variable se podrá modificar en el constructor de la misma.

Mostramos el pseudocódigo asociado a estas tres variantes:

```
Datos: Algoritmos Meméticos
inicio
    Pasar Población a Codificación Entera ;
    switch (tipo)
      case  AM-(10,1.0):
        para i in |poblacion|
            cromosoma ← poblacion[i] ;
            poblacion[i] ← BusquedaLocal(cromosoma , eval);
        break ;

      case  AM-(10,0.1):
        total ← prob_ls · n ;
        mientras total > 0 {
            r ← rand() %n ;
            cromosoma ← poblacion[r] ;
            poblacion[r] ← BusquedaLocal(cromosoma , eval);
            total - - ;
        }
        break ;

      case  AM-(10,0.1mej):
        n_mejores ← prob_ls · n ;
        aux ← Índices de los n_ mejores cromosomas;
        para i in aux hacer{
            cromosoma ← poblacion[i] ;
            poblacion[i] ← BusquedaLocal(cromosoma , eval);
        }
        break ;
    Pasar Población a Codificación Binaria ;
fin
```

Algoritmo 10: Algoritmos Meméticos

6. Resultados

6.1. Algoritmos Genéticos Generacionales

Instancia	Mejor coste	AGGp	Desv	Time	AGGu	Desv	Time
GKD-c_11_n500_m50	19587,13	19586,2	0,00	16,91	19561,4	0,13	69,98
GKD-c_12_n500_m50	19360,23	19359,6	0,00	16,05	19352,3	0,04	76,26
GKD-c_13_n500_m50	19366,69	19361,0	0,03	15,91	19361	0,03	78,76
GKD-c_14_n500_m50	19458,56	19458,2	0,00	16,30	19445,5	0,07	80,16
GKD-c_15_n500_m50	19422,15	19422,1	0,00	16,00	19412,7	0,05	80,44
GKD-c_16_n500_m50	19680,20	19679,5	0,00	16,27	19668,1	0,06	75,37
GKD-c_17_n500_m50	19331,38	19331,4	0,00	16,06	19325,1	0,03	79,72
GKD-c_18_n500_m50	19461,39	19460	0,01	16,18	19446,7	0,08	73,45
GKD-c_19_n500_m50	19477,32	19477,3	0,00	16,01	19464,4	0,07	72,37
GKD-c_20_n500_m50	19604,84	19604,8	0,00	15,65	19588,4	0,08	72,15
MDG-b_1_n500_m50	778030,62	755790	2,86	15,86	760558	2,25	71,94
MDG-b_2_n500_m50	779963,68	762735	2,21	16,15	756991	2,95	72,62
MDG-b_3_n500_m50	776768,43	759655	2,20	16,07	767226	1,23	72,32
MDG-b_4_n500_m50	775394,62	752398	2,97	16,69	746584	3,72	72,93
MDG-b_5_n500_m50	775611,06	759879	2,03	19,11	750147	3,28	79,06
MDG-b_6_n500_m50	775153,68	758308	2,17	17,87	748763	3,40	77,12
MDG-b_7_n500_m50	777232,87	759429	2,29	16,92	753540	3,05	73,13
MDG-b_8_n500_m50	779168,75	759832	2,48	17,38	757009	2,84	72,09
MDG-b_9_n500_m50	774802,18	760125	1,89	17,14	757970	2,17	73,37
MDG-b_10_n500_m50	774961,31	763852	1,43	15,71	754409	2,65	72,74
MDG-a_31_n2000_m200	114139	108682	4,78	523,88	108813	4,67	931,76
MDG-a_32_n2000_m200	114092	108946	4,51	580,85	109176	4,31	780,13
MDG-a_33_n2000_m200	114124	108659	4,79	557,01	108687	4,76	792,33
MDG-a_34_n2000_m200	114203	108478	5,01	601,11	109207	4,37	901.01
MDG-a_35_n2000_m200	114180	108858	4,66	605,42	108186	5,25	781.72
MDG-a_36_n2000_m200	114252	108797	4,77	608,99	109011	4,59	795,34
MDG-a_37_n2000_m200	114213	108626	4,89	566,00	108615	4,90	915,67
MDG-a_38_n2000_m200	114378	109186	4,54	557,96	108592	5,06	759.13
MDG-a_39_n2000_m200	114201	108556	4,94	591,34	108347	5,13	741.71
MDG-a_40_n2000_m200	114191	108866	4,66	631,10	108857	4,67	863.45

Tabla 6.1: Tabla Resultados Algoritmos Genéticos Generacionales

6.2. Algoritmos Genéticos Estacionarios

Instancia	Mejor coste	AGEp	Desv	Time	AGEu	Desv	Time
GKD-c_11_n500_m50	19587,13	19577,7	0,05	26,98	19582,9	0,02	29,55
GKD-c_12_n500_m50	19360,23	19355	0,03	26,84	19353,7	0,03	28,63
GKD-c_13_n500_m50	19366,69	19349,5	0,09	26,88	19357,1	0,05	29,53
GKD-c_14_n500_m50	19458,56	19443,3	0,08	27,48	19453,7	0,03	29,46
GKD-c_15_n500_m50	19422,15	19416	0,03	29,81	19418,7	0,02	27,96
GKD-c_16_n500_m50	19680,20	19678,2	0,01	31,99	19678,3	0,01	29,21
GKD-c_17_n500_m50	19331,38	19320,1	0,06	30,97	19325,5	0,03	27,06
GKD-c_18_n500_m50	19461,39	19444,4	0,09	29,51	19453,9	0,04	27,98
GKD-c_19_n500_m50	19477,32	19471,5	0,03	28,80	19477,3	0,00	28,64
GKD-c_20_n500_m50	19604,84	19598,4	0,03	29,57	19599,9	0,03	27,61
MDG-b_1_n500_m50	778030,62	757086	2,69	30,89	760952	2,20	28,09
MDG-b_2_n500_m50	779963,68	751029	3,71	27,01	763328	2,13	28,04
MDG-b_3_n500_m50	776768,43	755045	2,80	26,53	757693	2,46	27,81
MDG-b_4_n500_m50	775394,62	754901	2,64	29,94	767379	1,03	26,71
MDG-b_5_n500_m50	775611,06	750179	3,28	30,30	760322	1,97	27,02
MDG-b_6_n500_m50	775153,68	748749	3,41	28,28	754484	2,67	28,19
MDG-b_7_n500_m50	777232,87	750968	3,38	31,67	763788	1,73	27,13
MDG-b_8_n500_m50	779168,75	755859	2,99	30,29	764510	1,88	26,40
MDG-b_9_n500_m50	774802,18	751285	3,04	28,62	758980	2,04	29,22
MDG-b_10_n500_m50	774961,31	749695	3,26	30,39	765325	1,24	28,58
MDG-a_31_n2000_m200	114139	110589	3,11	1321,67	111428	2,38	890,51
MDG-a_32_n2000_m200	114092	110609	3,05	1367,44	111276	2,47	1106,02
MDG-a_33_n2000_m200	114124	110972	2,76	1313.23	111251	2,52	912,13
MDG-a_34_n2000_m200	114203	110548	3,20	1218.74	112094	1,85	875,89
MDG-a_35_n2000_m200	114180	110712	3,04	1217.12	111615	2,25	856,71
MDG-a_36_n2000_m200	114252	110813	3,01	1271.19	111544	2,37	911,05
MDG-a_37_n2000_m200	114213	110657	3,11	1119.46	111312	2,54	792,13
MDG-a_38_n2000_m200	114378	110782	3,14	1256.32	111932	2,14	854,81
MDG-a_39_n2000_m200	114201	110814	2,97	1303.41	111873	2,04	867,90
MDG-a_40_n2000_m200	114191	110748	3,02	1181.33	111549	2,31	832,35

Tabla 6.2: Tabla Resultados Algoritmos Genéticos Estacionarios

6.3. Algoritmos Meméticos

Instancia	Mejor coste	AM(1.0)	Desv	Time	AM(0.1)	Desv	Time	AM(mej)	Desv	Time
GKD-c_11_n500_m50	19587,13	19587,1	0,00	0,77	19587,1	0,00	5,79	19587,1	0,00	6,60
GKD-c_12_n500_m50	19360,23	19360,2	0,00	0,76	19360,2	0,00	5,17	19360,2	0,00	5,99
GKD-c_13_n500_m50	19366,69	19366,7	0,00	0,95	19366,7	0,00	5,96	19366,7	0,00	6,24
GKD-c_14_n500_m50	19458,56	19458,6	0,00	0,62	19458,6	0,00	5,68	19458,6	0,00	6,12
GKD-c_15_n500_m50	19422,15	19422,1	0,00	0,70	19422,1	0,00	6,29	19422,1	0,00	6,70
GKD-c_16_n500_m50	19680,20	19680,2	0,00	0,68	19680,2	0,00	6,25	19680,2	0,00	6,28
GKD-c_17_n500_m50	19331,38	19331,4	0,00	0,68	19331,4	0,00	6,12	19331,4	0,00	6,45
GKD-c_18_n500_m50	19461,39	19461,4	0,00	0,63	19461,4	0,00	5,73	19461,4	0,00	6,06
GKD-c_19_n500_m50	19477,32	19477,3	0,00	0,70	19477,3	0,00	5,70	19477,3	0,00	6,58
GKD-c_20_n500_m50	19604,84	19604,8	0,00	0,66	19604,8	0,00	6,19	19604,8	0,00	6,85
MDG-b_1_n500_m50	778030,62	761787	2,09	0,77	758166	2,55	6,12	763462	1,87	59,96
MDG-b_2_n500_m50	779963,68	774782	0,66	0,72	762255	2,27	5,99	772275	0,99	6,26
MDG-b_3_n500_m50	776768,43	772620	0,53	0,66	774668	0,27	6,46	768366	1,08	6,16
MDG-b_4_n500_m50	775394,62	774821	0,07	0,65	764028	1,47	7,00	771515	0,50	5,85
MDG-b_5_n500_m50	775611,06	765608	1,29	0,68	763078	1,62	6,68	773229	0,31	6,56
MDG-b_6_n500_m50	775153,68	768277	0,89	0,71	766400	1,13	6,43	765215	1,28	5,71
MDG-b_7_n500_m50	777232,87	764203	1,68	0,70	766893	1,33	6,20	768267	1,15	6,04
MDG-b_8_n500_m50	779168,75	769304	1,27	0,66	764302	1,91	6,46	773702	0,70	6,05
MDG-b_9_n500_m50	774802,18	770145	0,60	0,68	762961	1,53	6,52	761828	1,67	5,79
MDG-b_10_n500_m50	774961,31	773959	0,13	0,92	765402	1,23	6,58	771548	0,44	6,06
MDG-a_31_n2000_m200	114139	112774	1,20	23,15	113135	0,88	176,08	112687	1,27	183,03
MDG-a_32_n2000_m200	114092	112590	1,32	21,85	113110	0,86	177,47	112731	1,19	180,65
MDG-a_33_n2000_m200	114124	112874	1,10	22,45	112541	1,39	187,93	112928	1,05	183,04
MDG-a_34_n2000_m200	114203	112437	1,55	23,51	112646	1,36	189,08	112540	1,46	186.18
MDG-a_35_n2000_m200	114180	112819	1,19	21,89	113165	0,89	178,57	112841	1,17	175,43
MDG-a_36_n2000_m200	114252	112540	1,50	22,12	112863	1,22	184,16	112918	1,17	180,10
MDG-a_37_n2000_m200	114213	112767	1,27	22,88	112462	1,53	187,30	113069	1,00	174,18
MDG-a_38_n2000_m200	114378	112920	1,27	21,11	112917	1,28	188,18	112933	1,26	173,04
MDG-a_39_n2000_m200	114201	112582	1,42	24,56	112996	1,06	199,16	112579	1,42	172,14
MDG-a_40_n2000_m200	114191	112894	1,14	23,60	112977	1,06	186,45	112659	1,34	177,32

Tabla 6.3: Tabla Resultados Algoritmos Meméticos

6.4. Resultados Globales

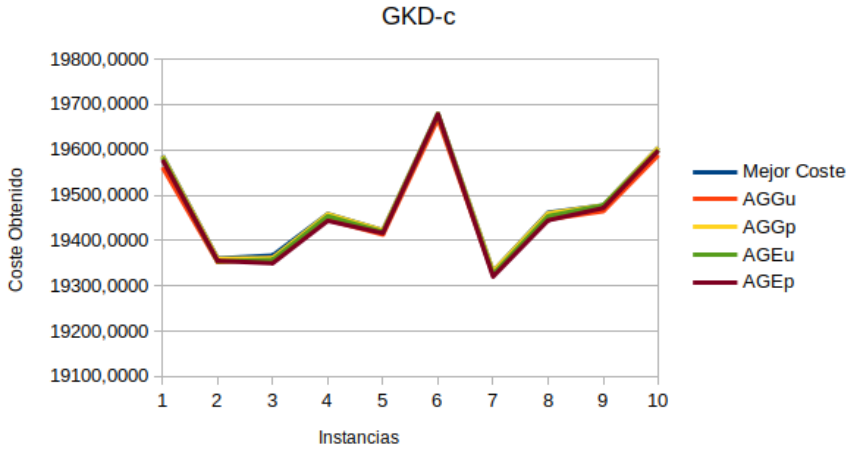
Algoritmo	*Desv*	*Tiempo*
Greedy	1,37	0,46
BL	1,07	0,49
$AGG - posicion$	2,34	205,13
$AGG - uniforme$	2,53	190,37
$AGE - posicion$	2,07	109,06
$AGE - uniforme$	1,42	315,41
$AM - (10,1,0)$	0,74	8,05
$AM - (10,0,1)$	0,89	65,92
$AM - (10,0,1mej)$	0,74	59,24

7. Comparación y Análisis de costes obtenidos

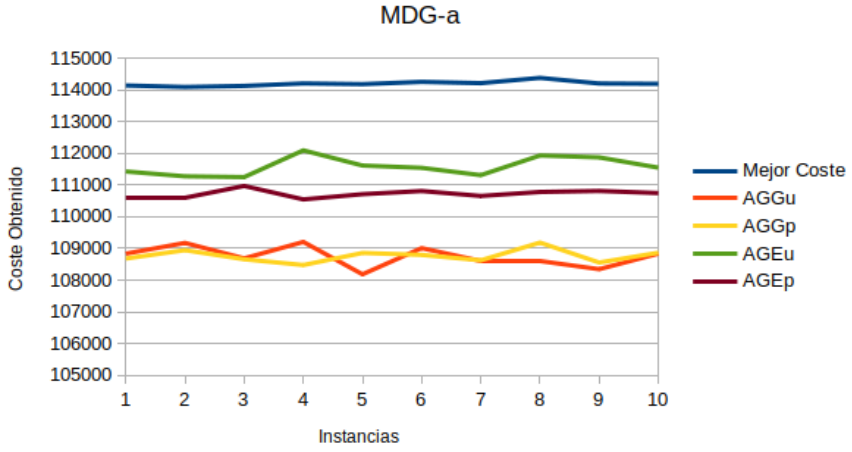
7.1. Algoritmos Genéticos

Recordemos los casos seleccionados para la ejecución de los algoritmos:
Tomamos 10 pertenecientes al grupo `GKD` con distancias Euclideas, 10 del grupo `MDG` con distancias reales y otras 10 del grupo `MDG` con distancias enteras. Mostramos los resultados obtenidos en 3 gráficas para reflejar mejor las diferencias:

- `GKD-c`: Conjunto de datos, referentes al grupo de distancias Euclideas con $n = 500$, $m = 50$, en el que mejor resultados se han obtenido, tanto el AGG como la AGE llegan al resultado óptimo en sus dos variantes. Hay convergencia con ambos algoritmos y con todas las instancias, de hecho, se aprecia que las tres líneas están superpuestas, obteniendo prácticamente una desviación nula.

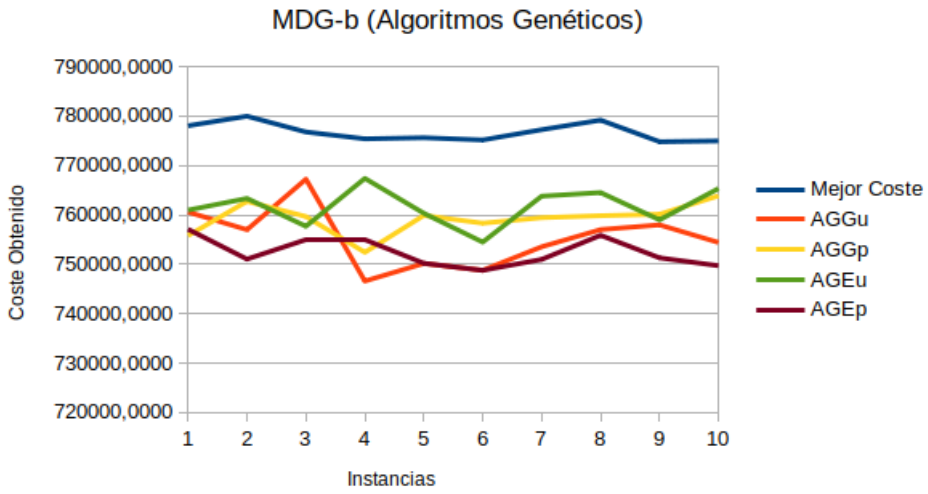


- `MDG-a`: Conjunto de datos correspondiente al grupo de distancias enteras y tamaño de $n = 2000$ y $m = 200$.



En todas las muestras los costes obtenidos en el AGE se encuentra sobre AGG, en sus dos variantes. En el AGE se aprecia que el que usa cruce uniforme tiene una mayor convergencia y esto es debido a que el cruce posicional es más disruptivo y comparte menos información de los padres. Esto no ocurre en el caso del AGG, donde se ve que no hay un claro ganador.

- `MDG-b`: Conjunto de datos correspondiente al grupo de distancias reales y tamaño $n = 500$ y $m = 50$.

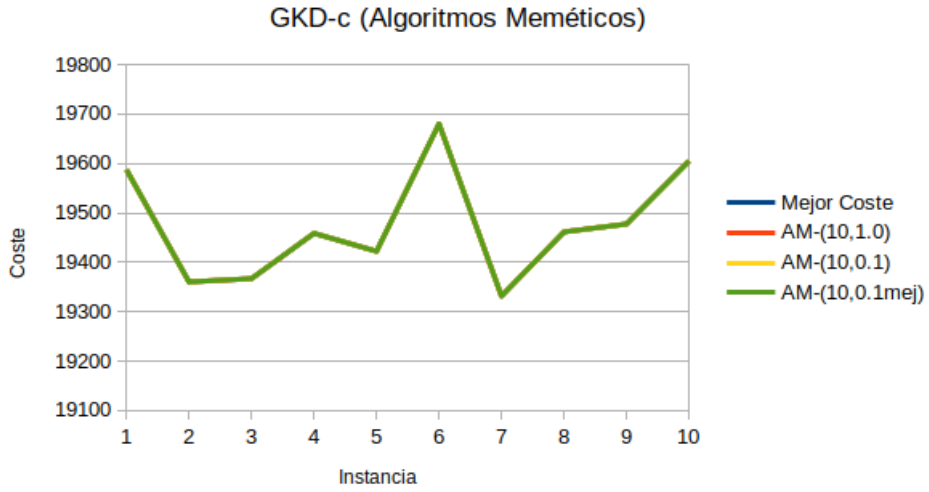


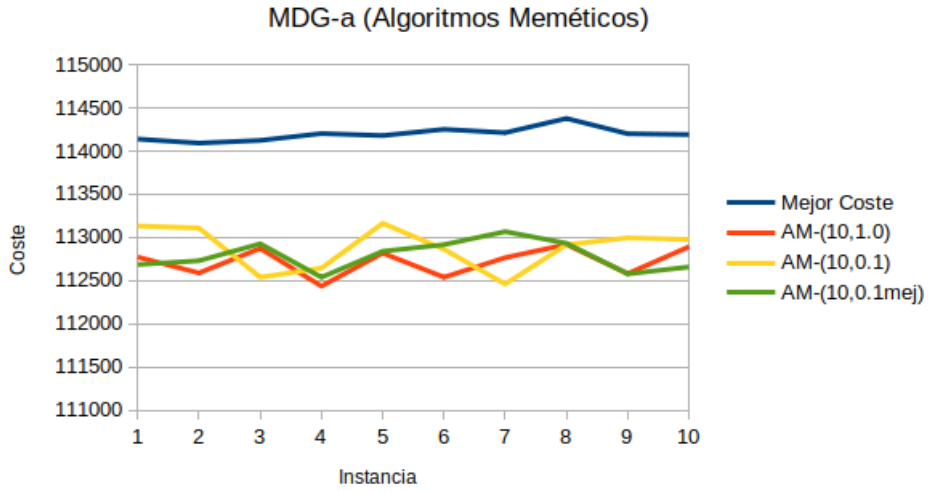
Se puede ver que la línea verde mayoritariamente está por encima del resto, es decir, al igual que en el caso anterior, el algoritmo de AGE con la estrategia de cruce uniforme mejora los costes de los otros algoritmos. Sin embargo, lo que se gana en eficacia se pierde en eficiencia, ya que si observamos la tabla de resultados globales 8.1, el AGEu es el que tiene un mayor tiempo de ejecución de media, en total 315,41 segundos (Casi duplicando el resto de tiempos de los AG).

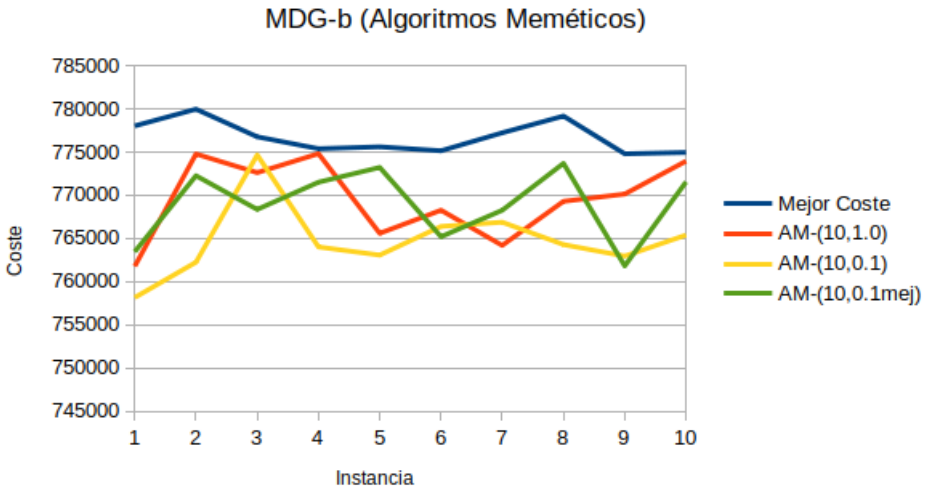
Por otro lado y en segundo lugar se encuentra el AGG con cruce posicional que mejora respecto a su variante de cruce uniforme y será entonces el elegido para la implementación de los Algoritmos Meméticos.

7.2. Algorítmos Meméticos

Como hemos comentado en el punto anterior, el AGG que se ha usado para la población ha sido el del cruce basado en posición, aunque por muy pocas décimas. 8.1. Los resultados en general han sido muy satisfactorios, mejorando como era de esperar los resultados de BL y AGG por separado.
La hibridación de los Algoritmos Meméticos no solo ha mejorado en eficacia sino que su eficiencia también se ha visto favorecida. Ésta ha aumentado, obteniendo un tiempo de ejecución bastante inferior respecto al AGG original, coincidiendo con el caso teórico esperado. Esto es debido a que la BL implementada en la práctica 1 realizaba una factorización de la función objetivo, no siendo necesario recalcular el coste de ls solución obtenida (bastaba con sumar y restar la contribución del elemento añadido y quitado, respectivamente).
Mostramos para cada grupo `(GKD-c, MDG-a y MDG-b)` el coste obtenido para cada una de sus instancias:



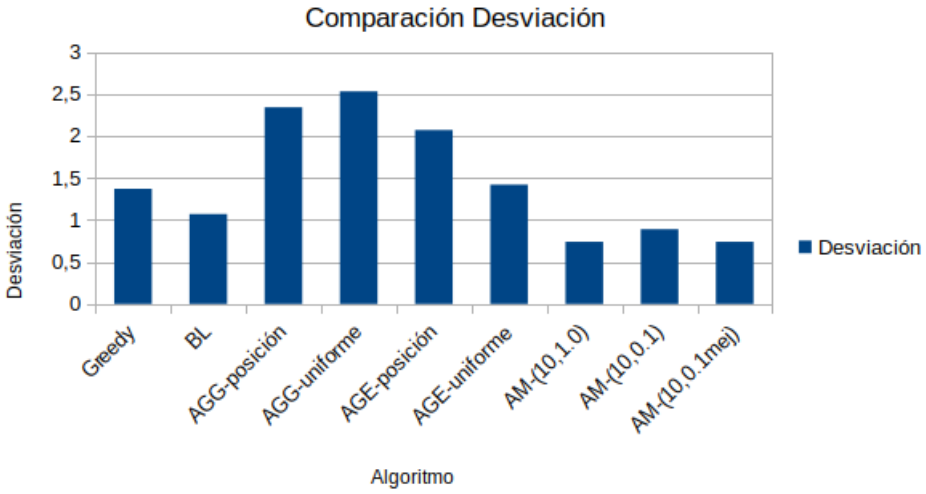




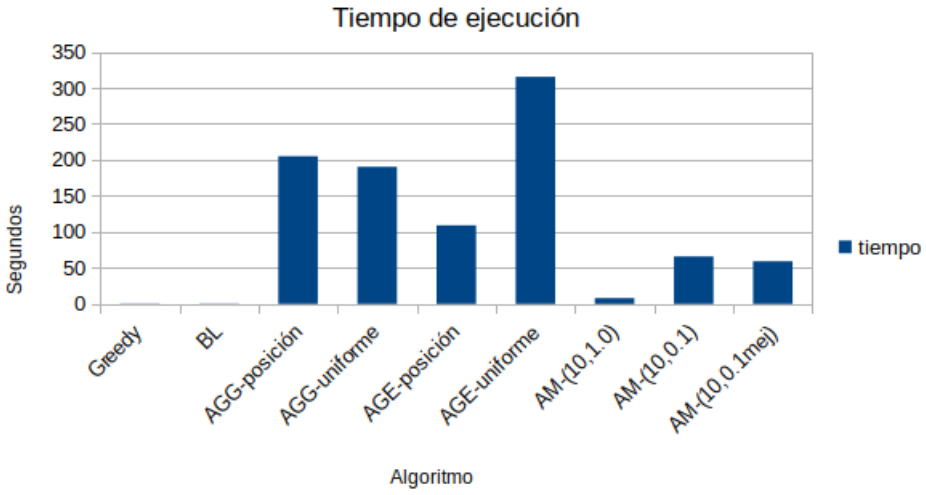
A simple vista, observando las gráficas anteriores, no hay un claro vencedor. Sin embargo, en la gráfica de `MDG-b`, la línea amarilla referente a `AM-(10,0.1)` se encuentra mayoritariamente por debajo del resto lo que se ve reflejado en una leve peor desviación en la tabla 8.1.
Si recordamos esta variante, consistía en aplicar BL a un determinado número de cromosomas elegidos de forma aleatoria. Como era de esperar, la aleatoriedad no se lleva bien (en este caso) con los algoritmos y, de hecho, los mejores resultados se obtienen en `AM-(10,1.0)` y `AM-(10,0.1mej)` donde no hay un factor aleatorio sino que se aplica la BL sobre todos los cromosomas y sobre los mejores cromosomas, respectivamente.

7.3. Desviación y tiempo

Por último, reflejamos en un par de gráficos la desviación y el tiempo obtenidos de media para cada uno de los algoritmos implementados tanto en la práctica 1 como en la 2.
Conviene destacar que los algoritmos que peores resultados arrojan son los Algoritmos Genéticos Generacionales, obteniendo una desviación incluso peor que el algoritmo Greedy. Como norma general, el Greedy suele ser uno de los algoritmos que peor se adapta en cuanto a resultados al problema, sin embargo, en nuestro caso no es así.



En lo referente al tiempo, los algoritmos genéticos se llevan la derrota. Como se ha comentado anteriormente, la mejor variante en coste ha sido AGEu, pero también ha sido la peor en tiempo.
Por otro lado, la técnica BL tanto individual como en sus versiones hibridadas de los Algoritmos Meméticos en nuestro problema MDP ha sido un gran logro tanto en eficacia como en eficiencia obteniendo unos resultados casi óptimos y ejecutando casi todas las instancias por debajo de los 50 segundos.



8. Ejercicio extra

Durante la realización de esta práctica me ha parecido muy interesante el equilibrio entre exploración y explotación llevado a cabo en las variantes de nuestro algoritmo memético y cómo aumentando un enfoque u otro se pueden obtener unos mejores resultados.

Recordando la sección 5, la variante `AM-(10,1.0)` aplicaba BL sobre todos los cromosomas, `AM-(10,0.1)` aplicaba BL a un subconjunto de cromosomas de la población elegido de forma aleatoria y `AM-(10,0.1mej)` aplicaba BL sobre los **0.1·mejores** cromosomas de la población actual. El problema de estos casos es que la exploración era algo débil ya que la BL consumía todas las evaluaciones de la función objetivo. Por ello, establecemos un nuevo criterio de parada de 600 generaciones (en torno al doble) para olvidarnos del problema de la exploración y así establecer una heurística de explotación para obtener un resultado óptimo para nuestro problema MDP.

Para ello, se han programado tres nuevas variantes:

- `AM-(greedy1)`: Es el Algoritmo Memético en su versión más greedy. A la hora de aplicar BL, se explora el cromosoma que mejor coste tiene de toda la población, es decir, se elige la opción óptima en cada paso local, con la esperanza de llegar a una solución general óptima.

  Nos podemos encontrar situaciones en las que uno de los cromosomas sea bueno en comparación con el resto de cromosomas de la población pero no lo sea tanto para el problema MDP. Si este cromosoma se encuentra en un óptimo local, se aplicará BL sobre éste pero no saldrá de este óptimo, obstaculizando así la exploración del resto de cromosomas.

- `AM-(greedy2)`: Versión greedy en la que se intentan evitar óptimos locales. En vez de explorar los cromosomas que más coste tienen, se exploran los que menos. Con esto, se evitarán más óptimos locales ya que se explora un abanico más grande de cromosomas. Sin embargo, nos encontramos en el dilema de estar explorando cromosomas no prometedores para evitar óptimos locales.

- `AM-(worst)`: Para la implementación de esta versión me he basado en la idea del operador de reparación del operador de cruce Uniforme usado en nuestros Algoritmos Genéticos y propuesto en el Seminario 3. Cuando corresponda, se aplica BL sobre el subconjunto de $n/2$ **peores** cromosomas (donde n es el tamaño de la población). Con esto, ganamos diversidad ya que exploramos también las soluciones menos prometedoras y, además, el problema de los óptimos locales deja de serlo ya que el subconjunto donde se aplica BL es suficientemente grande.

Mostramos a continuación el pseudocódigo de las tres versiones:

Datos: Algoritmos Meméticos
inicio

case `AM-(greedy1):`
$n\_mejores \leftarrow prob\_ls \cdot n$;
mientras $n\_mejores > 0$ {
$i \leftarrow best\_element(poblacion)$;
$cromosoma \leftarrow poblacion[i]$;
$poblacion[i] \leftarrow$ **BusquedaLocal***(cromosoma , eval)*;
$n\_mejores$ - - ;
}
break ;

case `AM-(greedy1):`
$n\_peores \leftarrow prob\_ls \cdot n$;
mientras $n\_peores > 0$ {
$i \leftarrow worst\_element(poblacion)$;
$cromosoma \leftarrow poblacion[i]$;
$poblacion[i] \leftarrow$ **BusquedaLocal***(cromosoma , eval)*;
n_peores - - ;
}
break ;

case `AM-(worst):`
$n\_peores \leftarrow n/2$;
$aux \leftarrow$ *Índices de los n_peores cromosomas*;
para *i in aux* **hacer**{
$cromosoma \leftarrow poblacion[i]$;
$poblacion[i] \leftarrow$ **BusquedaLocal***(cromosoma , eval)*;
}
break ;

fin

Algoritmo 11: Algoritmos Meméticos

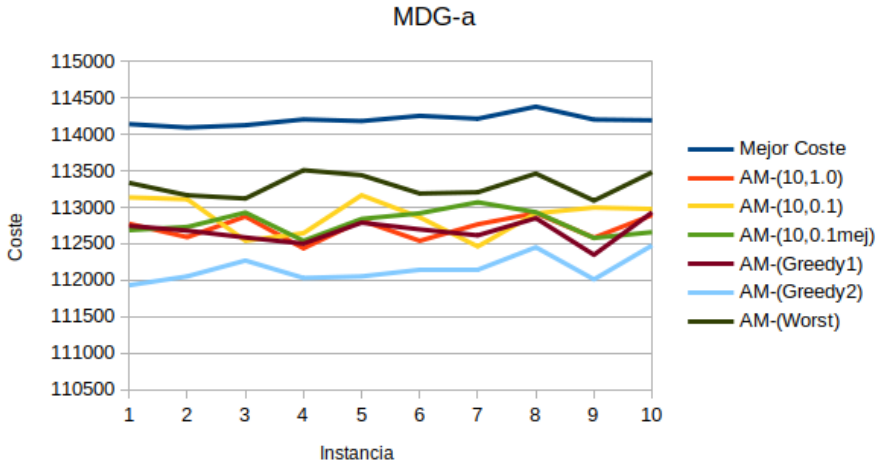
8.1. Resultados

Instancia	Mejor Coste	AM-Greedy1	Desv	AM-Greedy2	Desv	AM-Worst	Desv
GKD-c_11_n500_m50	19587,13	19587,1	0,00	19587,1	0,00	19587,1	0,00
GKD-c_12_n500_m50	19360,24	19360,2	0,00	19360,2	0,00	19360,2	0,00
GKD-c_13_n500_m50	19366,70	19366,7	0,00	19366,7	0,00	19366,7	0,00
GKD-c_14_n500_m50	19458,56	19458,6	0,00	19458,6	0,00	19458,6	0,00
GKD-c_15_n500_m50	19422,15	19422,1	0,00	19422,1	0,00	19422,1	0,00
GKD-c_16_n500_m50	19680,21	19680,2	0,00	19680,2	0,00	19680,2	0,00
GKD-c_17_n500_m50	19331,39	19331,4	0,00	19328,4	0,02	19331,4	0,00
GKD-c_18_n500_m50	19461,39	19461,4	0,00	19461,4	0,00	19461,4	0,00
GKD-c_19_n500_m50	19477,39	19477,3	0,00	19477,3	0,00	19477,3	0,00
GKD-c_20_n500_m50	19604,84	19604,8	0,00	19604,8	0,00	19604,8	0,00
MDG-b_1_n500_m50	778030,62	766611	1,47	767828	1,31	768717	1,20
MDG-b_2_n500_m50	779963,69	762531	2,24	766806	1,69	764921	1,93
MDG-b_3_n500_m50	776768,44	774819	0,25	770896	0,76	773699	0,40
MDG-b_4_n500_m50	775394,62	767390	1,03	767084	1,07	770724	0,60
MDG-b_5_n500_m50	775611,06	769639	0,77	767524	1,04	774024	0,20
MDG-b_6_n500_m50	775153,69	768731	0,83	769179	0,77	768685	0,83
MDG-b_7_n500_m50	777232,87	765342	1,53	766411	1,39	769942	0,94
MDG-b_8_n500_m50	779168,75	762033	2,20	773812	0,69	772572	0,85
MDG-b_9_n500_m50	774802,19	765032	1,26	768830	0,77	770562	0,55
MDG-b_10_n500_m50	774961,31	764799	1,31	767798	0,92	766434	1,10
MDG-a_31_n2000_m200	114139	112746	1,22	111932	1,93	113334	0,71
MDG-a_32_n2000_m200	114092	112680	1,24	112054	1,79	113164	0,81
MDG-a_33_n2000_m200	114124	112586	1,35	112270	1,62	113120	0,88
MDG-a_34_n2000_m200	114203	112502	1,49	112032	1,90	113508	0,61
MDG-a_35_n2000_m200	114180	112790	1,22	112053	1,86	113439	0,65
MDG-a_36_n2000_m200	114252	112698	1,36	112148	1,84	113189	0,93
MDG-a_37_n2000_m200	114213	112615	1,40	112142	1,81	113205	0,88
MDG-a_38_n2000_m200	114378	112851	1,34	112452	1,68	113462	0,80
MDG-a_39_n2000_m200	114201	112348	1,62	112012	1,92	113091	0,97
MDG-a_40_n2000_m200	114191	112931	1,10	112477	1,50	113479	0,62

Tabla 8.1: Resultados AM-Extra

Algoritmo	*Desv*	*Tiempo*
$AM-(10,1,0)$	$0,74$	$8,05$
$AM-(10,0,1)$	$0,89$	$65,92$
$AM-(10,0,1mej)$	$0,74$	$59,24$
$AM-(Greedy1)$	$0,87$	$19,48$
$AM-(Greedy2)$	$0,94$	$19,95$
$AM-(Worst)$	$0,55$	$19,71$

Como era de esperar por lo comentado anteriormente, tanto la versión `Greedy1` como `Greedy2` no mejoran respecto a las versiones implementadas (a pesar de tener un criterio de parada menos restrictivo). Por otro lado, el mejor algoritmo implementado hasta ahora ha sido `AM-(Worst)`, con un total de 0.55 de desviación media total y con un tiempo de ejecución relativamente bajo, 19,71 segundos de media.



En las instancias de `GKD-c` todos los algoritmos llegan al óptimo global y en las de `MDG-b` se producen demasiados solapamientos y no permiten diferenciar muy bien los colores referentes a cada algoritmo. Sin embargo, en `MDG-a` la visualización es clara ya que nuestro problema MDP trabaja con una matriz de distancias de mayor tamaño.
Como podemos observar, `AM-(Worst)` es el claro ganador en términos de convergencia y `AM-Greedy` en sus dos versiones implementadas son los peores. Esto nos deja como conclusión que no por una exploración más profunda se van a obtener mejores resultado sino que lo que realmente importa es la forma de hacerlo.

Referencias

-Seminario 1
-Seminario 2
-Seminario 3
-Guión de prácticas