# PRÁCTICA 2
# METAHEURÍSTICAS

## PROBLEMA 1: MDD


UNIVERSIDAD DE GRANADA

**David Muñoz Sánchez 07256819C**

dmunozs14@correo.ugr.es **Grupo A3**

**(Martes)**

**Algoritmos considerados:** AGG_CruceUni, AGG_CrucePos, AGE_CruceUni, AGE_CrucePos.

# Índice

# Descripción del problema

El problema es el MDD (*Minimum Differential Dispersion Problem)*, que consiste en seleccionar un conjunto de m elementos de un conjunto inicial de n elementos de forma que se minimice la dispersión entre los elementos escogidos, de los que tenemos las distancias entre ellos.

La dispersión es la diferencia entre la distancia acumulada máxima y la mínima de un conjunto de m elementos. La distancia acumulada de un elemento es la suma de distancias de ese elemento a todos los demás.

A partir de una solución inicial aleatoria, o, en el caso de esta práctica, de una población de soluciones aleatorias, se tiene que conseguir un conjunto de soluciones aplicando algoritmos genéticos.

# Descripción de los algoritmos empleados y consideraciones comunes a estos

Antes de comenzar con la explicación de los algoritmos y las consideraciones comunes a estos es necesario indicar el esquema de representación de soluciones. Los algoritmos genéticos trabajan sobre una población, que ha sido representada con un conjunto de conjuntos binarios. Cada cromosoma representa una solución de esa población y los 1's que tengan indican que el número real que representa a esa posición está en la solución, por lo que no puede haber más de siete 1's, por ejemplo, en un conjunto de soluciones n=100 m=7, y por tanto, cada cromosoma no puede tener más de 100 genes. Es decir, tiene que haber exactamente siete 1's y noventa y tres 0's en ese caso.

Se han usado varias versiones de los algoritmos genéticos para intentar obtener soluciones lo más cercanas posibles a las óptimas. La primera versión desarrollada es el AGG_Uni, que comparte la estructura con la segunda versión, el AGG_Pos. Estos algoritmos se diferencian en el operador de cruce, que será explicado posteriormente. En general, estos dos algoritmos seleccionan 50 padres al azar de una población inicial de 50 cromosomas de la que previamente se ha obtenido la mejor solución. Los 50 padres seleccionados por torneo binario, se cruzan según una determinada esperanza de cruce (dada la probabilidad de cruce de dos padres). Cada padre tiene dos hijos. La población de hijos

se rellena aleatoriamente con cromosomas de la población intermedia para completar los 50 cromosomas. Esta población de hijos puede sufrir mutaciones según una probabilidad de mutación por individuo, así que se hacen exactamente las mutaciones que indique la esperanza de mutación (los genes a mutar se deciden de forma aleatoria). Si el mejor padre que guardamos anteriormente es mejor que el peor hijo de la nueva población de hijos, lo sustituimos y directamente esta población de hijos sustituye a la de padres.

Las otras dos versiones son AGE_Uni, AGE_Pos, que igualmente se diferencian tan solo en el operador de cruce. Los AGE en general toman 4 padres al azar entre los que hacen un torneo binario 2 a 2 y seleccionan los padres ganadores. Estos padres se cruzan con probabilidad 1 y tienen dos hijos, que pueden mutar según la probabilidad de mutación de cada individuo. Los dos hijos nuevos se comparan con los dos peores padres de la población inicial y se sustituyen exclusivamente los hijos que mejoren a alguno de los dos o a ambos.

En cuanto a los operadores de cruce, ambos toman lo común de los dos cromosomas a cruzar y el de posición, introduce lo que falte en los hijos a partir de lo que restaba en uno de los padres tras desestimar lo que tiene en común con el otro padre, tomando los elementos de forma aleatoria. El de cruce uniforme toma lo restante de forma aleatoria de un padre o de otro, por lo que, puede que la solución que nos proporcione no sea factible (tenga más 1's de la cuenta o menos de los necesarios). Es por ello que es necesario implementar un reparador que tenga en cuenta esto y que añada o quite 1's según afecte menos o disminuya más la media de distancias acumuladas, respectivamente.

A continuación, se adjunta pseudocódigo de la función objetivo y de los operadores comunes a los Algoritmos Genéticos:

## Función objetivo

```
maximo ← 0
minimo ← 0
suma ← 0
elegidos[] //Enteros
elegidos ← devolverEntero(cromosoma, n)
//Estos parámetros se han pasado como argumento a la función objetivo
//DevolverEntero devuelve un vector de enteros a partir de un cromosoma que es como se representa nuestra solución
para i ← 0 hasta TAMAÑO de elegidos[]
```

```
[incremento 1] hacer
      para j ← 0 hasta TAMAÑO de elegidos[]
      [incremento 1] hacer
            Si elegidos[i] <> elegidos[j] entonces
                  suma ← suma +
            distancias[elegidos[i]][elegidos[j]]
                  //La matriz de distancias se pasa como
            parámetro a la función objetivo
      fin_para
      Si suma > maximo entonces
            maximo ← suma
      Fin_si
      Si minimo = 0 entonces
            minimo ← suma
      Sino entonces
            Si suma < minimo entonces
                  minimo ← suma
            Fin_si
      Fin_si

      suma ← 0

fin_para

Devolver maximo - minimo
```

## Operadores comunes

### Generación de soluciones aleatorias

```
seleccionables[] //Este vector está previamente rellenado con
su posición, es decir seleccionables[0] tiene el 0 y así hasta
seleccionables [n-1] que tiene n-1
numeros_cromosoma[] //Vector vacío para añadir los números que
serán la solución, es decir, los genes que tendrán que estar
marcados con 1 según nuestro esquema de representación
primer_sel ← Número random entre 0 y n-1 //Primer cromosoma
seleccionado
seleccionado ← primer_sel
seleccionables[primer_sel] ← -1 //Este gen no se puede volver
a tomar

numeros_cromosoma[] ← añadir seleccionado

//cromosomas[][] es un vector previamente definido que
representa nuestra población
Mientras TAMAÑO de cromosomas[] < TOTAL CROMOSOMAS hacer
```

```
        Mientras TAMAÑO de numeros_cromosomas[] < m hacer
            Mientras seleccionables[seleccionado] = -1 hacer
                seleccionado ← Núm. random entre 0 y n-1
            Fin_mientras
            seleccionables[seleccionado] ← -1
            numeros_cromosoma[] ← añadir seleccionado
        Fin_Mientras

        //cromosoma es un vector vacío binario previamente
definido
        cromosoma[] <- deolverBinario(numeros_cromosoma, n, m)
        //Es necesario pasar nuestro esquema de representación
binario los numeros de genes que representan nuestra solucion
        cromosomas[][] ← añadir cromosoma[]
        cromosoma[] ← Se deja con 0 elementos
        numeros_cromosoma[] ← Se deja con 0 elementos
    Fin_Mientras
```

## Mecanismo de selección

Para el mecanismo de selección, se usa el torneo binario y se ejecuta tantas veces como padres queramos seleccionar con cromosomas aleatorios de la población.

### Torneo binario

```
    //Para el torneo binario necesitamos los fitness de los
cromosomas que van a competir y los índices de los fitness de los
cromosomas que van a competir
    Si fitness[indice_cromosoma1] < fitness[indice_cromosoma2]
entonces
        Devuelve indice_cromosoma1
    Sino
        Devuelve indice_cromosoma2
    Fin_si
```

## Operador de cruce por posición

Para este operador necesitamos dos padres (cromosomas) y el tamaño de la solución (n).

```
    solucion[] //Binario del tamaño de uno de los padres
    solucion_escritos[] //Del tamaño de uno de los padres, binario
y con todas sus posiciones inicializadas a false
    restantes_padre2[] //Binario
    hijo[] //Binario
```

```
hijos[][] //Binario
continuar ← true
contador_hijos ← 0
valor

Para i ← 0 hasta TAMAÑO de uno de los padres
[incremento 1] hacer
      Si padre1[i] = padre2[i] entonces
            solucion[i] ← padre1[i]
            solucion_escritos[i] ← true
      Sino entonces
            restantes_padre2[] ← añadir padre2[i]
      Fin_si
Fin_para

solucion_anterior[] ← solucion[]
solucion_escritos_anterior[] ← solucion_escritos
tamanio_restantes ← TAMAÑO de restantes_padre2[]

Mientras contador_hijos < 2 hacer
      restantes_padre2[] ← se baraja
      tomados_restantes[] //Se inicializa a false y con tamaño
= tamanio_restantes

      Para i ← 0 hasta TAMAÑO de restantes_padre2[]
      [incremento 1] hacer
            tomados_restantes[i] = true;
            hijo[] ← añadir restantes_padre2[i]
      Fin_para

      Para j ← 0 hasta TAMAÑO de hijo[]
      [incremento 1] hacer
            continuar ← true

            Para i ← 0 hasta TAMAÑO de solucion[] y mientras
continuar = true
            [incremento 1] hacer
                  Si solucion_escritos[i] = false entonces
                        continuar ← false
                        solucion[i] ← hijo[j]
                        solucion_escritos[i] ← true
                  Fin_si
            Fin_para
      Fin_para
      hijos[][] ← añadir solucion[]
      contador_hijos ← contador_hijos + 1
      hijo[] ← Se limpia y se deja vacío
      solucion[] ← solucion_anterior[]
```

```
            solucion_escritos[] ← solucion_escritos_anterior[]
      Fin_mientras
      Devuelve hijos[][]
```

## Operador de cruce uniforme

Para el operador de cruce uniforme será necesario disponer de los dos padres a cruzar, el tamaño del cromosoma y el tamaño de la solución, además de la matriz de distancias para después ejecutar el reparador que se comentará al final.

```
      prob_padre
      posicion
      padre[] //Binario
      solucion[] //Binario con tamaño n
      solucion_escritos[] //Binario con TAMAÑO de uno de los padres
e inicializado a false
      retorno[] //Binario

      Para i ← 0 hasta TAMAÑO de unos de los padres
      [incremento 1] hacer
            Si padre1[i] = padre2[i] entonces
                  solucion[i] ← padre1[i]
                  solucion_escritos[i] ← true
            Fin_si
      Fin_para
      Para i ← 0 hasta TAMAÑO de solucion[]
      [incremento 1] hacer
            prob_padre ← float random entre 0 y 1
            Si prob_padre < 0.5 entonces
                  padre ← padre1
            Sino entonces
                  padre ← padre2
            Fin_si

            posicion ← entero random entre 0 y n-1
            Si solucion_escritos[i] = false entonces
                  solucion_escritos[i] ← true
                  solucion[i] ← padre[posicion]
            Fin_Si
      Fin_para

      retorno[] ← reparador(solucion, n, m, matriz_dist)**

      Devolver retorno[]
```

**El reparador se encarga de (si la solución tiene un número de 1’s menor que m) seleccionar los nuevos teniendo en cuenta que

tiene que aumentar lo menos posible la media de las distancias acumuladas que haya en ese momento. En caso de que haya más 1's de la cuenta, tiene que ir deshaciéndose de ellos de manera que reduzca la media lo máximo posible. Si tiene los 1's necesarios se devuelve igual.

### Operador de mutación

Primero, expongo el pseudocódigo del método encargado de intercambiar los genes dentro de un cromosoma. En este caso, se necesita un cromosoma y dos posiciones a intercambiar. Dicho cromosoma debe quedar modificado o podemos alternativamente devolver una copia de ese cromosoma modificado.

```
intercambio ← cromosoma[pos1]
cromosoma[pos1] ← cromosoma[pos2]
cromosoma[pos2] ← intercambio
```

Para garantizar que la mutación se hace de forma correcta, hemos de comprobar que los genes que intercambiamos tienen valores necesariamente distintos. Esto se hace generando dos posiciones aleatorias hasta que sean distintas o representen a genes distintos. Una vez hecho esto, se aplica la mutación a los cromosomas según el algoritmo.

## Componentes particulares de cada algoritmo

Los dos AGG y los AGE comparten el esquema de evolución y de reemplazamiento.

En el caso de los AGG, se eligen 50 padres porque la población genética es de 50 cromosomas, este sería el esquema de evolución. En cuanto al reemplazamiento, la población de hijos sustituye a la actual con la salvedad de que si el mejor padre es mejor que el peor hijo, se cambia el peor hijo por el mejor padre.

### Esquema de evolución AGG

```
Para j ← 0 hasta TOTAL CROMOSOMAS
[incremento 1] hacer
      indices_seleccionables[] ← mezclar
      indice_padre1_AGG ← último elemento de
indices_seleccionables[]
      indices_seleccionables[] ← mezclar
```

```
        indice_padre2_AGG ← último elemento de
    indices_seleccionables[]
        indice ← torneoBinario(indice_padre1_AGG,
    indice_padre2_AGG, fitness_cromosomas[])
        intermedia_AGG[] ← añadir cromosomas[indice]

        //Este si sino no seria necesario puesto que los padres
    ganadores se pueden repetir
        Si indice = indice_padre1_AGG entonces
            indices_seleccionables [] ← añadir indice_padre2_AGG
        Sino entonces
            indices_seleccionables [] ← añadir indice_padre1_AGG
        Fin_si
            indices_seleccionables [] ←
        indices_seleccionables_anterior[]

    Fin_para
```

## Esquema reemplazamiento AGG

Tras rellenar los hijos faltantes con cromosomas aleatorios de la población intermedia, se aplican las mutaciones convenientes y posteriormente se comprueba si el mejor de los cromosomas de la población actual es mejor que el peor de los hijos de la nueva población, en cuyo caso se intercambian. Además, se actualizan los fitness de la población actual.

En el caso de los AGE, se seleccionan 2 padres aleatorios para lo cual se aplican 2 torneos binarios. El esquema de reemplazamiento compara 2 a 2 las peores soluciones de la población actual con los dos hijos generados tras el cruce de los dos padres.

## Esquema de evolución AGE

```
    indice_padre1_AGE ← núm. random entre 0 y TOTAL CROMOSOMAS-1
    indice2_padre1_AGE ← núm. random entre 0 y TOTAL CROMOSOMAS-1
    Mientras indice_padre1_AGE = indice2_padre1_AGE hacer
        indice2_padre1_AGE ← núm random entre 0 y TOT. CRO. -1
    Fin_mientras

    padre1_AGE ← cromosomas[torneoBinario(indice_padre1_AGE,
indice2_padre1_AGE, fitness_cromosomas[])]

    indice_padre2_AGE ← núm. random entre 0 y TOTAL CROMOSOMAS-1
```

```
     Mientras indice_padre2_AGE = indice_padre1_AGE or
indice_padre2_AGE = indice2_padre1_AGE hacer
          indice_padre2_AGE ← núm random entre 0 y TOT. CRO. -1
     Fin_mientras

     indice2_padre2_AGE <- núm. random entre 0 y TOTAL CROMOSOMAS-1
     Mientras indice_padre2_AGE = indice2_padre2_AGE or
indice2_padre2_AGE = indice2_padre1_AGE hacer
          indice2_padre2_AGE ← núm random entre 0 y TOT. CRO. -1
     Fin_mientras

     padre2_AGE    ←    cromosomas[torneoBinario(indice_padre2_AGE,
indice2_padre2_AGE, fitness_cromosomas[])]
```

### Esquema reemplazamiento AGE

Mediante un método, se obtienen los dos peores cromosomas de la población actual. Sus índices se guardan en minimos[]. En hijos_AGE únicamente hay dos hijos. Primero se compara el primer hijo con el primer peor. Si es mejor el hijo pasamos a la única comparación que podemos hacer. Si no fuera mejor, comparamos el primer hijo con el segundo peor y, lo mejore o no, pasamos a comparar el segundo hijo con el primer peor.

```
     Si fitness_cromosomas[minimos[0]] > fitness_hijo1 entonces
          cromosomas[minimos[0]] ← hijos_AGE[0]
          fitness_cromosomas[minimos[0]] ← fitness_hijo1

          Si fitness_cromosomas[minimos[1]] > fitness_hijo2
     entonces
          cromosomas[minimos[1]] ← hijos_AGE[1]
          fitness_cromosomas[minimos[1]] ← fitness_hijo2
          Fin_si

     Sino entonces
          Si fitness_cromosomas[minimos[1]] > fitness_hijo1
     entonces
          cromosomas[minimos[1]] ← hijos_AGE[0]
          fitness_cromosomas[minimos[1]] ← fitness_hijo1
          Fin_si

          Si fitness_cromosomas[minimos[0]] > fitness_hijo2
     entonces
          cromosomas[minimos[0]] ← hijos_AGE[1]
          fitness_cromosomas[minimos[0]] ← fitness_hijo2
          Fin_si
     Fin_si
```

# Procedimiento desarrollo

La práctica ha sido desarrollada haciendo uso de C++. Para la implementación de la práctica, se ha tomado el código para leer los archivos con las distancias de la Práctica 1. Para leer los archivos, hago uso de las principales funciones de **fstream**, para el control de flujo y de errores. Se sigue la misma política que en la práctica anterior de introducir la semilla (entre 1 y 5) para inicializar los números aleatorios como argumento de la ejecución. Además, ahora se incluye otro argumento (entre 1 y 4) para seleccionar qué versión del algoritmo genético queremos ejecutar **(1 significa AGG_CruceUni, 2 significa AGG_CrucePos, 3 significa AGE_CruceUni y 4 significa AGE_CrucePos).**

Todo lo demás se ha hecho desde cero haciendo uso sobre todo del tipo de dato **vector** de la *STL* y, por consiguiente, de los métodos asociados a este así como de iteradores y sus métodos para borrar valores concretos de los vectores.

# Experimentos y análisis de resultados

Es importante resaltar antes de proceder con el análisis, que todos los experimentos se han hecho introduciendo como argumento la semilla **1.**

## Resultados AGG_Uni

| Algoritmo AGG_Uni | | | |
|---|---|---|---|
| **Caso** | **Coste medio obtenido** | **Desv** | **Tiempo (ms)** |
| GKD-b_1_n25_m2 | 0,0000 | 0,00 | No medido |
| GKD-b_2_n25_m2 | 0,0000 | 0,00 | No medido |
| GKD-b_3_n25_m2 | 0,0000 | 0,00 | No medido |
| GKD-b_4_n25_m2 | 0,0000 | 0,00 | No medido |
| GKD-b_5_n25_m2 | 0,0000 | 0,00 | No medido |
| GKD-b_6_n25_m7 | 12,7180 | 0,00 | 5,10E+03 |
| GKD-b_7_n25_m7 | 14,0988 | 0,00 | 5,09E+03 |
| GKD-b_8_n25_m7 | 20,9588 | 20,03 | 5,07E+03 |
| GKD-b_9_n25_m7 | 21,7391 | 21,48 | 5,01E+03 |
| GKD-b_10_n25_m7 | 26,2380 | 11,33 | 5,10E+03 |
| GKD-b_11_n50_m5 | 3,6741 | 47,58 | 1,08E+04 |
| GKD-b_12_n50_m5 | 2,9138 | 27,21 | 1,08E+04 |
| GKD-b_13_n50_m5 | 5,2999 | 55,43 | 1,08E+04 |
| GKD-b_14_n50_m5 | 3,7904 | 56,12 | 1,08E+04 |
| GKD-b_15_n50_m5 | 6,2471 | 54,33 | 1,07E+04 |
| GKD-b_16_n50_m15 | 111,2440 | 61,57 | 1,34E+04 |
| GKD-b_17_n50_m15 | 68,3292 | 29,59 | 1,35E+04 |
| GKD-b_18_n50_m15 | 114,8120 | 62,38 | 1,35E+04 |
| GKD-b_19_n50_m15 | 102,1450 | 54,56 | 1,36E+04 |
| GKD-b_20_n50_m15 | 128,5530 | 62,88 | 1,35E+04 |
| GKD-b_21_n100_m10 | 63,7992 | 78,32 | 2,02E+04 |
| GKD-b_22_n100_m10 | 64,7718 | 78,90 | 2,03E+04 |
| GKD-b_23_n100_m10 | 56,5065 | 72,84 | 2,04E+04 |
| GKD-b_24_n100_m10 | 52,0748 | 83,41 | 2,03E+04 |
| GKD-b_25_n100_m10 | 60,9892 | 71,80 | 2,06E+04 |
| GKD-b_26_n100_m30 | 482,5350 | 65,03 | 3,42E+04 |
| GKD-b_27_n100_m30 | 421,8630 | 69,87 | 3,42E+04 |
| GKD-b_28_n100_m30 | 420,2390 | 74,69 | 3,40E+04 |
| GKD-b_29_n100_m30 | 395,8960 | 65,28 | 3,38E+04 |
| GKD-b_30_n100_m30 | 381,8420 | 66,61 | 3,39E+04 |
| GKD-b_31_n125_m12 | 106,8400 | 89,01 | 3,32E+04 |
| GKD-b_32_n125_m12 | 58,6067 | 67,94 | 3,34E+04 |
| GKD-b_33_n125_m12 | 102,7150 | 81,96 | 3,32E+04 |
| GKD-b_34_n125_m12 | 84,3774 | 76,90 | 3,36E+04 |
| GKD-b_35_n125_m12 | 98,7289 | 81,65 | 3,34E+04 |
| GKD-b_36_n125_m37 | 562,9120 | 72,39 | 5,93E+04 |
| GKD-b_37_n125_m37 | 696,4890 | 71,44 | 5,97E+04 |
| GKD-b_38_n125_m37 | 728,2560 | 74,19 | 5,94E+04 |
| GKD-b_39_n125_m37 | 530,7810 | 68,24 | 5,95E+04 |
| GKD-b_40_n125_m37 | 617,2000 | 71,13 | 5,92E+04 |
| GKD-b_41_n150_m15 | 154,7310 | 84,91 | 3,27E+04 |
| GKD-b_42_n150_m15 | 157,0580 | 82,94 | 3,26E+04 |
| GKD-b_43_n150_m15 | 139,5470 | 80,83 | 3,26E+04 |
| GKD-b_44_n150_m15 | 132,3940 | 80,41 | 3,27E+04 |
| GKD-b_45_n150_m15 | 114,7550 | 75,80 | 3,26E+04 |
| GKD-b_46_n150_m45 | 752,4850 | 69,73 | 7,38E+04 |
| GKD-b_47_n150_m45 | 836,7290 | 72,68 | 7,46E+04 |
| GKD-b_48_n150_m45 | 769,8220 | 70,55 | 7,46E+04 |
| GKD-b_49_n150_m45 | 750,1640 | 69,82 | 7,46E+04 |
| GKD-b_50_n150_m45 | 818,2830 | 69,59 | 7,38E+04 |

**Media Desv:** **56,07**
**Media Tiempo:** **2,83E+04**

## Resultados AGG_Pos

| Algoritmo AGG_Pos | | | |
|---|---|---|---|
| **Caso** | **Coste medio obtenido** | **Desv** | **Tiempo (ms)** |
| GKD-b_1_n25_m2 | 0,0000 | 0,00 | No medido |
| GKD-b_2_n25_m2 | 0,0000 | 0,00 | No medido |
| GKD-b_3_n25_m2 | 0,0000 | 0,00 | No medido |
| GKD-b_4_n25_m2 | 0,0000 | 0,00 | No medido |
| GKD-b_5_n25_m2 | 0,0000 | 0,00 | No medido |
| GKD-b_6_n25_m7 | 12,7180 | 0,00 | 2,36E+03 |
| GKD-b_7_n25_m7 | 14,0988 | 0,00 | 2,36E+03 |
| GKD-b_8_n25_m7 | 16,7612 | 0,00 | 2,39E+03 |
| GKD-b_9_n25_m7 | 21,7391 | 21,48 | 2,38E+03 |
| GKD-b_10_n25_m7 | 28,2510 | 17,65 | 2,35E+03 |
| GKD-b_11_n50_m5 | 4,4756 | 56,96 | 3,44E+03 |
| GKD-b_12_n50_m5 | 5,6086 | 62,18 | 3,46E+03 |
| GKD-b_13_n50_m5 | 6,1648 | 61,68 | 3,45E+03 |
| GKD-b_14_n50_m5 | 5,4695 | 69,59 | 3,46E+03 |
| GKD-b_15_n50_m5 | 5,3404 | 46,57 | 3,51E+03 |
| GKD-b_16_n50_m15 | 119,5080 | 64,23 | 4,51E+03 |
| GKD-b_17_n50_m15 | 103,5320 | 53,53 | 4,51E+03 |
| GKD-b_18_n50_m15 | 97,2842 | 55,60 | 4,52E+03 |
| GKD-b_19_n50_m15 | 89,6576 | 48,23 | 4,54E+03 |
| GKD-b_20_n50_m15 | 112,5570 | 57,61 | 4,57E+03 |
| GKD-b_21_n100_m10 | 55,0130 | 74,86 | 5,68E+03 |
| GKD-b_22_n100_m10 | 50,0706 | 72,71 | 5,74E+03 |
| GKD-b_23_n100_m10 | 57,8306 | 73,46 | 5,70E+03 |
| GKD-b_24_n100_m10 | 41,4895 | 79,17 | 5,71E+03 |
| GKD-b_25_n100_m10 | 67,4166 | 74,49 | 5,75E+03 |
| GKD-b_26_n100_m30 | 496,0240 | 65,98 | 9,51E+03 |
| GKD-b_27_n100_m30 | 452,6080 | 71,92 | 9,58E+03 |
| GKD-b_28_n100_m30 | 415,9600 | 74,43 | 9,59E+03 |
| GKD-b_29_n100_m30 | 456,6830 | 69,90 | 9,55E+03 |
| GKD-b_30_n100_m30 | 470,0070 | 72,88 | 9,49E+03 |
| GKD-b_31_n125_m12 | 87,5115 | 86,58 | 7,85E+03 |
| GKD-b_32_n125_m12 | 93,1430 | 79,83 | 7,73E+03 |
| GKD-b_33_n125_m12 | 79,4948 | 76,69 | 7,74E+03 |
| GKD-b_34_n125_m12 | 106,9970 | 81,79 | 7,78E+03 |
| GKD-b_35_n125_m12 | 94,3068 | 80,79 | 7,77E+03 |
| GKD-b_36_n125_m37 | 531,1120 | 70,73 | 1,38E+04 |
| GKD-b_37_n125_m37 | 686,5330 | 71,03 | 1,36E+04 |
| GKD-b_38_n125_m37 | 532,0260 | 64,67 | 1,37E+04 |
| GKD-b_39_n125_m37 | 598,8320 | 71,85 | 1,36E+04 |
| GKD-b_40_n125_m37 | 658,0960 | 72,92 | 1,38E+04 |
| GKD-b_41_n150_m15 | 147,2290 | 84,14 | 9,13E+03 |
| GKD-b_42_n150_m15 | 160,1080 | 83,27 | 9,06E+03 |
| GKD-b_43_n150_m15 | 147,4990 | 81,86 | 9,15E+03 |
| GKD-b_44_n150_m15 | 140,0360 | 81,48 | 9,05E+03 |
| GKD-b_45_n150_m15 | 113,3020 | 75,49 | 9,04E+03 |
| GKD-b_46_n150_m45 | 897,6910 | 74,63 | 1,75E+04 |
| GKD-b_47_n150_m45 | 820,0990 | 72,12 | 1,74E+04 |
| GKD-b_48_n150_m45 | 688,1550 | 67,05 | 1,74E+04 |
| GKD-b_49_n150_m45 | 836,9160 | 72,95 | 1,75E+04 |
| GKD-b_50_n150_m45 | 819,0170 | 69,62 | 1,74E+04 |

**Media Desv:** **57,29**
**Media Tiempo:** **7,36E+03**

## Resultados AGE_Uni

| Algoritmo AGE_Uni | | | |
|---|---|---|---|
| **Caso** | **Coste medio obtenido** | **Desv** | **Tiempo (ms)** |
| GKD-b_1_n25_m2 | 0,0000 | 0,00 | No medido |
| GKD-b_2_n25_m2 | 0,0000 | 0,00 | No medido |
| GKD-b_3_n25_m2 | 0,0000 | 0,00 | No medido |
| GKD-b_4_n25_m2 | 0,0000 | 0,00 | No medido |
| GKD-b_5_n25_m2 | 0,0000 | 0,00 | No medido |
| GKD-b_6_n25_m7 | 20,0439 | 36,55 | 4,09E+03 |
| GKD-b_7_n25_m7 | 27,1901 | 48,15 | 4,15E+03 |
| GKD-b_8_n25_m7 | 36,4758 | 54,05 | 4,13E+03 |
| GKD-b_9_n25_m7 | 25,0145 | 31,76 | 4,09E+03 |
| GKD-b_10_n25_m7 | 38,4036 | 39,42 | 4,05E+03 |
| GKD-b_11_n50_m5 | 12,8333 | 84,99 | 9,89E+03 |
| GKD-b_12_n50_m5 | 17,8438 | 88,11 | 9,89E+03 |
| GKD-b_13_n50_m5 | 14,0029 | 83,13 | 9,91E+03 |
| GKD-b_14_n50_m5 | 14,0864 | 88,19 | 9,97E+03 |
| GKD-b_15_n50_m5 | 8,3574 | 65,86 | 9,94E+03 |
| GKD-b_16_n50_m15 | 87,4344 | 51,11 | 1,09E+04 |
| GKD-b_17_n50_m15 | 80,8277 | 40,48 | 1,09E+04 |
| GKD-b_18_n50_m15 | 96,3155 | 55,15 | 1,09E+04 |
| GKD-b_19_n50_m15 | 165,2900 | 71,92 | 1,08E+04 |
| GKD-b_20_n50_m15 | 90,3847 | 47,21 | 1,09E+04 |
| GKD-b_21_n100_m10 | 28,9044 | 52,15 | 1,52E+04 |
| GKD-b_22_n100_m10 | 35,0319 | 60,99 | 1,50E+04 |
| GKD-b_23_n100_m10 | 47,2557 | 67,53 | 1,47E+04 |
| GKD-b_24_n100_m10 | 32,2502 | 73,21 | 1,51E+04 |
| GKD-b_25_n100_m10 | 35,7630 | 51,90 | 1,49E+04 |
| GKD-b_26_n100_m30 | 419,9450 | 59,82 | 1,74E+04 |
| GKD-b_27_n100_m30 | 257,5450 | 50,65 | 1,72E+04 |
| GKD-b_28_n100_m30 | 303,5030 | 64,95 | 1,71E+04 |
| GKD-b_29_n100_m30 | 317,1270 | 56,66 | 1,71E+04 |
| GKD-b_30_n100_m30 | 366,9210 | 65,26 | 1,75E+04 |
| GKD-b_31_n125_m12 | 44,6115 | 73,67 | 2,56E+04 |
| GKD-b_32_n125_m12 | 50,6613 | 62,91 | 2,56E+04 |
| GKD-b_33_n125_m12 | 105,9550 | 82,51 | 2,60E+04 |
| GKD-b_34_n125_m12 | 51,9658 | 62,50 | 2,58E+04 |
| GKD-b_35_n125_m12 | 31,8860 | 43,20 | 2,57E+04 |
| GKD-b_36_n125_m37 | 407,1830 | 61,83 | 2,96E+04 |
| GKD-b_37_n125_m37 | 343,4840 | 42,09 | 2,98E+04 |
| GKD-b_38_n125_m37 | 635,2430 | 70,41 | 2,81E+04 |
| GKD-b_39_n125_m37 | 338,5760 | 50,21 | 2,92E+04 |
| GKD-b_40_n125_m37 | 408,4160 | 56,37 | 2,93E+04 |
| GKD-b_41_n150_m15 | 74,4400 | 68,64 | 1,72E+04 |
| GKD-b_42_n150_m15 | 67,3933 | 60,25 | 1,72E+04 |
| GKD-b_43_n150_m15 | 87,7245 | 69,50 | 1,69E+04 |
| GKD-b_44_n150_m15 | 76,6583 | 66,17 | 1,76E+04 |
| GKD-b_45_n150_m15 | 82,3657 | 66,28 | 1,74E+04 |
| GKD-b_46_n150_m45 | 582,0290 | 60,87 | 2,09E+04 |
| GKD-b_47_n150_m45 | 430,0430 | 46,84 | 2,13E+04 |
| GKD-b_48_n150_m45 | 688,0990 | 67,05 | 2,08E+04 |
| GKD-b_49_n150_m45 | 506,1030 | 55,26 | 2,17E+04 |
| GKD-b_50_n150_m45 | 537,7350 | 53,72 | 2,04E+04 |

**Media Desv:** **54,19**
**Media Tiempo:** **1,50E+04**

## Resultados AGE_Pos

| Algoritmo AGE_Pos | | | |
|---|---|---|---|
| **Caso** | **Coste medio obtenido** | **Desv** | **Tiempo (ms)** |
| GKD-b_1_n25_m2 | 0,0000 | 0,00 | No medido |
| GKD-b_2_n25_m2 | 0,0000 | 0,00 | No medido |
| GKD-b_3_n25_m2 | 0,0000 | 0,00 | No medido |
| GKD-b_4_n25_m2 | 0,0000 | 0,00 | No medido |
| GKD-b_5_n25_m2 | 0,0000 | 0,00 | No medido |
| GKD-b_6_n25_m7 | 29,9510 | 57,54 | 1,28E+03 |
| GKD-b_7_n25_m7 | 26,5340 | 46,87 | 1,29E+03 |
| GKD-b_8_n25_m7 | 29,9996 | 44,13 | 1,30E+03 |
| GKD-b_9_n25_m7 | 32,8645 | 48,06 | 1,30E+03 |
| GKD-b_10_n25_m7 | 26,8843 | 13,46 | 1,30E+03 |
| GKD-b_11_n50_m5 | 7,8336 | 75,41 | 1,86E+03 |
| GKD-b_12_n50_m5 | 6,8062 | 68,84 | 1,88E+03 |
| GKD-b_13_n50_m5 | 13,5757 | 82,60 | 1,86E+03 |
| GKD-b_14_n50_m5 | 14,3189 | 88,38 | 1,88E+03 |
| GKD-b_15_n50_m5 | 8,8058 | 67,60 | 1,86E+03 |
| GKD-b_16_n50_m15 | 78,6998 | 45,69 | 2,23E+03 |
| GKD-b_17_n50_m15 | 93,1043 | 48,33 | 2,25E+03 |
| GKD-b_18_n50_m15 | 111,8440 | 61,38 | 2,23E+03 |
| GKD-b_19_n50_m15 | 85,4822 | 45,71 | 2,28E+03 |
| GKD-b_20_n50_m15 | 78,4737 | 39,20 | 2,22E+03 |
| GKD-b_21_n100_m10 | 38,2960 | 63,88 | 2,08E+03 |
| GKD-b_22_n100_m10 | 35,0167 | 60,98 | 2,10E+03 |
| GKD-b_23_n100_m10 | 28,9166 | 46,93 | 2,08E+03 |
| GKD-b_24_n100_m10 | 32,9497 | 73,78 | 2,11E+03 |
| GKD-b_25_n100_m10 | 32,0570 | 46,34 | 2,09E+03 |
| GKD-b_26_n100_m30 | 350,9440 | 51,92 | 3,26E+03 |
| GKD-b_27_n100_m30 | 283,8540 | 55,22 | 3,31E+03 |
| GKD-b_28_n100_m30 | 212,3500 | 49,90 | 3,31E+03 |
| GKD-b_29_n100_m30 | 190,9200 | 28,00 | 3,32E+03 |
| GKD-b_30_n100_m30 | 321,6670 | 60,37 | 3,33E+03 |
| GKD-b_31_n125_m12 | 42,1169 | 72,11 | 2,80E+03 |
| GKD-b_32_n125_m12 | 48,6422 | 61,37 | 2,79E+03 |
| GKD-b_33_n125_m12 | 53,2948 | 65,23 | 2,78E+03 |
| GKD-b_34_n125_m12 | 57,7842 | 66,27 | 2,81E+03 |
| GKD-b_35_n125_m12 | 44,9110 | 59,67 | 2,77E+03 |
| GKD-b_36_n125_m37 | 333,2860 | 53,36 | 4,61E+03 |
| GKD-b_37_n125_m37 | 474,7270 | 58,10 | 4,64E+03 |
| GKD-b_38_n125_m37 | 663,3020 | 71,66 | 4,59E+03 |
| GKD-b_39_n125_m37 | 333,5540 | 49,46 | 4,63E+03 |
| GKD-b_40_n125_m37 | 402,5850 | 55,74 | 4,64E+03 |
| GKD-b_41_n150_m15 | 57,1690 | 59,16 | 2,43E+03 |
| GKD-b_42_n150_m15 | 83,8813 | 68,06 | 1,00E+01 |
| GKD-b_43_n150_m15 | 90,9194 | 70,57 | 2,40E+03 |
| GKD-b_44_n150_m15 | 67,0231 | 61,30 | 2,38E+03 |
| GKD-b_45_n150_m15 | 64,4129 | 56,88 | 2,38E+03 |
| GKD-b_46_n150_m45 | 527,8370 | 56,85 | 4,97E+03 |
| GKD-b_47_n150_m45 | 485,8020 | 52,94 | 4,91E+03 |
| GKD-b_48_n150_m45 | 403,6730 | 43,83 | 4,93E+03 |
| GKD-b_49_n150_m45 | 471,1220 | 51,94 | 4,96E+03 |
| GKD-b_50_n150_m45 | 486,7310 | 48,87 | 4,90E+03 |

**Media Desv:** **51,08**
**Media Tiempo:** **2,51E+03**

Resultados globales (en términos de media)

| RESULTADOS GLOBALES | | |
|---|---|---|
| **Algoritmo** | **Desv** | **Tiempo (ms)** |
| Greedy | 63,99 | 1,15E+01 |
| BL | 54,71 | 3,41E+02 |
| AGG_Uni | 56,07 | 2,83E+04 |
| AGG_Pos | 57,29 | 7,36E+03 |
| AGE_Uni | 54,19 | 1,50E+04 |
| AGE_Pos | 51,08 | 2,51E+03 |

Antes de realizar el análisis de los resultados cabe comentar las diferencias notables de tiempo entre los algoritmos que usan el cruce uniforme y los que usan el cruce por posición.

El método reparador, anteriormente comentado, llama varias veces a la función objetivo, pero a una versión modificada que en vez de devolver el fitness devuelve directamente la lista con las distancias acumuladas. Para intentar rebajar los tiempos tendríamos que enfocarnos en este método y en optimizar lo máximo posible, por ejemplo, haciendo uso de menos llamadas a la función objetivo modificada.

Atendiendo ahora a las desviaciones medias, claramente el peor algoritmo es el Greedy, ya que como vimos en la práctica 1, dependía mucho de la solución inicial aleatoria. Es destacable también que los AG Generacionales nos dan peores resultados en términos generales que los AG Estacionarios, lo cual tiene sentido porque en los estacionarios solo añadimos los dos hijos a la población si mejoran a los dos peores padres con lo cual, nos aseguramos que la población siempre mejora en mayor o menor medida. Sin embargo, en los generacionales, a pesar de que se mantenga al mejor padre si es mejor que el peor hijo, se sustituye toda la población por la de hijos y perfectamente podríamos pasar a un escenario peor del que partíamos. La búsqueda local, a pesar de quedar estancada rápidamente en un mínimo local, logra mejores resultados que los Algoritmos Genéticos Generacionales.

Teóricamente, el Cruce Uniforme debería funcionar mejor que el Cruce por Posición, ya que tiene en cuenta la media de distancias acumuladas para añadir o quitar 1’s, y esto ha sido así en el caso de los generacionales, pero no de los estacionarios. Esto, probablemente, se deba a la semilla elegida para inicializar los números aleatorios.

Por último, comparando las ejecuciones individuales, se puede observar un mejor comportamiento generalizado de los Generacionales con respecto a los estacionarios hasta el caso de prueba 20, es decir, en cromosomas con 25 o 50 genes. Al reducir el número de posibles soluciones, los generacionales obtienen mejores resultados puesto que exploran más cromosomas que los estacionarios, a pesar de que estos siempre mejoran la población.