UNIVERSIDAD DE JAÉN
ESCUELA POLITÉCNICA SUPERIOR DE JAÉN

Práctica 2:
Metaheurísticas basadas en poblaciones y algoritmos genéticos para el Problema de Asignación de Frecuencias con Mínimas Interferencias (Mi-Fap)

Curso 2017-2018

Grado de Ingeniería Informática

Algoritmos considerados:

Algoritmo Greedy

Algoritmo genético estacionario (Cruce dos puntos)

Algoritmo genético estacionario (Cruce BLX)

Algoritmo genético generacional (Cruce dos puntos)

Algoritmo genético generacional (Cruce BLX)

Realizado por:

Nombre: Alejandro Ureña Marín DNI: 77384109B Correo electrónico: aum00006@red.ujaen.es

Nombre: Manuel García López DNI: 77381751E Correo electrónico: mgl00050@red.ujaen.es

Grupo 1 Prácticas: Martes 12:30-14:30

Índice

1.Descripción y formulación del problema

Descripción

El problema del MI-FAP consiste en asignar frecuencias a una serie de transmisores (cada uno de los cuales posee un rango de frecuencias disponibles en las que emitir datos). Sea T ={t1, t2, … tn } un conjunto de n transmisores (TRX) y sea Fi = {fi1, fi2,… f ik}} ⊂ N un conjunto de frecuencias válidas que pueden ser asignadas a los transmisores t i ∈T, i=1.. , n. Nótese que el cardinal de k de Fi no es necesariamente el mismo para todos los transmisores. Se define una matriz de interferencias entre las frecuencias (incluida esa información en el fichero ctr.txt de cada instancia).
La función objetivo consiste es que el coste de la suma de las interferencias existentes en los transmisores instalados, sea el mínimo posible, buscando llegar a el mínimo global de coste 0 como caso ideal.

Función objetivo del Mi-FAP

$$fitness = \sum_{i=0}^{t} \sum_{j=i+1}^{t} MI(TRXi, TRXj)$$

La representación se llevará a cabo mediante un vector solución de enteros del tamaño total de x transmisores, en el que cada uno llevará asociada una de las y frecuencias (sin incluir permutaciones, porque dichas frecuencias pueden estar repetidas) correspondientes al rango de transmisión en el que emite cada transmisor.

Sol = {1, ... , y}

Restricciones (indicadas en ctr.txt)

1- Cada transmisor solo puede usar una de las disponibles para este transmisor.
2- Las restricciones tienen la siguiente forma:

-trx1 trx2 C>100 4

(Indica la restricción del transmisor 1 respecto a el 2, donde en caso de ser uno, 100 unidades de frecuencia mayor que 2, o más, se sumará 4 al cálculo del coste del problema)

-No se usan las restricciones que contengan el símbolo D=

2. Descripción de la aplicación de los algoritmos empleados al problema

Generación de la solución inicial

La solución inicial será creada mediante un algoritmo Greedy, el cual, consistirá en seleccionar un transmisor de manera aleatoria y asignar a los demás transmisores una frecuencia dentro de su dominio realizando una búsqueda local. Todo ello se realizará 50 veces inicialmente.

Desarrollo de la solución

Salvo en el caso del algoritmo Greedy, para el resto de algoritmos, tendremos que realizar el cálculo de una función objetivo. El cálculo de esta función objetivo se realiza minimizando el "Z" o costo de la selección de frecuencias, siempre que se tengan en cuenta las restricciones de todos los transmisores y que no se deje ningún transmisor sin asignar.

Pseudocódigo de la función objetivo (Comprobación de la solución)

En nuestro caso dispondremos de dos funciones de comprobación, una de objetivo inicial para el Greedy, y otra factorial para la comprobación final de los otros algoritmos, que se detallara posteriormente.

```
devolver ← 0
Bucle i ← 0 hasta nº restricciones hacer
        trx1 ← restricciones (i, trx1)
        trx2 ← restricciones (i, trx2)
        a ← frecuencias (transmisores(trx1(sol(trx1))))
        b ← frecuencias (transmisores(trx2(sol(trx2))))
        c ← a - b
        c ← abs(c)
        Si c mayor que diferencia de frecuencias de la restricción
                devolver ← devolver + penalización de la restricción
        Fin_Si
Fin_Bucle
```

Carga y lectura de ficheros

Para proceder a realizar la práctica, primero necesitamos los datos almacenados en los ficheros específicos. Se utilizarán las 10 instancias dadas. Cuyos tamaños oscilan entre 200 a 916 transmisores con entre 1134 a 5273 restricciones cada uno.

El formato de los ficheros es el siguiente:
• var.txt: la lista de los enlaces de comunicación (transmisores), con una columna que indica el número de transmisor y otra el rango de frecuencias de emisión.
• dom.txt: la lista de dominios de frecuencia, en la que por cada rango aparecen la cantidad de frecuencias posibles y los valores de las mismas.
• ctr.txt: la lista de todas las restricciones, explicado anteriormente.

Pseudocódigo de las funciones carga de ficheros

-Carga del fichero de transmisores (var.txt)

Procedimiento loadVar(nombre: cadena, v: vector, t: vector)

```
Coger cadenas de memoria auxiliares dato1 y dato2
Comprobar apertura del fichero nombre
Mientras que no se llegue a el fin del fichero
        t  ← insertar dato1
        v ← insertar dato2
Fin_Mientras
Cerrar fichero
```

-Carga del fichero de dominio de frecuencias de los transmisores (dom.txt)

Procedimiento loadDom(nombre: cadena, v: matriz)

```
Coger memoria vector auxiliar w y memoria para datoVar
Comprobar apertura del fichero nombre
Mientras que no se llegue a el fin de fichero
        Si no hay cadenas vacías
                w ← insertar datoVar
                v ← insertar w
        Fin_Si
Fin_Mientras
Cerrar fichero
```

-Carga del fichero de las restricciones de los transmisores (ctr.txt)

Procedimiento loadCtr (nombre: cadena, w: vector, v: matriz, t: vector)

```
Coger memoria para cadenas a, b, c, d, e y f; dato dato
Comprobar apertura del fichero nombre
Mientras que no se llegue a el fin de fichero
        Si c distinto de 'D'
                Bucle i ← 0 hasta tamaño de t hacer
                        Si t(i) es igual que a y a no se habia encontrado
                                dato(trx1) igual a i
                        Fin_Si
                        Si t(i) es igual que b y b no se habia encontrado
                                dato(trx2) igual a i
                        Fin_Si
                        Si a y b han sido encontrados
                                salir del bucle
                        Fin_Si
                Fin_Bucle
                dato(diferencia frecuencias) igual a e
                dato(penalización) igual a f
                w ← insertar dato
        Fin_Si
Fin_Mientras
Cierre fichero

Bucle i ← 0 hasta tamaño de w hacer
        v(i(trx1)) ← insertar w(i)
        v(i(trx2)) ← insertar w(i)
Fin_Bucle
```

Inicialización

Procedimiento Inicialización (var: vector, dom: matriz, ctr: vector, ctr2: matriz, poblacion: vector (individuo), reiniciar: bool)

Creamos un individuo
pos ← 0
Si reiniciar es cierto entonces
    pos ← pos +1
    individuo ← primer elemento de población
    limpiamos población y le añadimos individuo
Fin_Si
Bucle desde i← pos hasta 50 hacer
    Llamar a Greedy e incluir la solución del greedy en la población
Fin_Bucle
Ordenar población

3. Descripción en pseudocódigo de la estructura del método de búsqueda y de las operaciones relevantes de cada algoritmo

Componentes de los algoritmos (Esquema general):

Los AGs de esta práctica presentarán las siguientes componentes:

• Esquema de representación: En este caso vamos a usar un esquema de representación basado en un vector solución de enteros, de nuestra población.

• Función objetivo: Minimizar interferencias

$$fitness = \sum_{i=0}^{t} \sum_{j=i+1}^{t} MI(TRXi, TRXj)$$

• Generación de la población inicial: Todos los cromosomas se generarán usando un greedy simple similar al del grasp pero no aleatorio y sin sesgo.

• Esquema de evolución: Variará en función de si se va a hacer uso del AGE o esquema estacionario, o del AGG o esquema generacional elitista. En el primero se seleccionarán únicamente dos padres, mientras que en el segundo seleccionaremos una población de padres del mismo tamaño que la población genética.

• Operador de selección: Se usará el torneo binario, consistente en elegir aleatoriamente dos individuos de la población y seleccionar el mejor de ellos. En el AGG, se aplicarán tantos torneos como individuos existan en la población genética, incluyendo los individuos ganadores en la población de padres. En el AGE, se aplicará dos veces el torneo para elegir los dos padres que serán posteriormente recombinados.

• Esquema de reemplazamiento: En el AGG, la población de hijos sustituye automáticamente a la actual. Para conservar el elitismo, si la mejor solución de la generación anterior no sobrevive, sustituye directamente la peor solución de la

nueva población. En el AGE, 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.

• Operadores de cruces: Cruce en dos puntos y Cruce BLX-α.

• Operador de mutación: Selecciona un individuo en función de una probabilidad, y cambia el valor de los genes, según otra probabilidad de mutación por gen.

• Reinicialización: Se aplicará una reinicialización para evitar un estancamiento de la búsqueda. Será dinámico y se podrán hacer 0, 1 o muchas veces, dependerá de la evolución del algoritmo.

“Una vez habiendo tenido en cuenta este esquema general seguido por nuestros algoritmos genéticos, procederemos a realizar una breve explicación, junto con el pseudocódigo de cada uno de los algoritmos.”

Greedy

El algoritmo Greedy, será usado principalmente como algoritmo de comparación de resultados, pues al comenzar el testeo, se tomará una población inicial, que se evaluará tanto en el Greedy, como en el resto de algoritmos, con el fin de obtener una población inicial igual y comparar mejor los resultados.

Pseudocódigo del Greedy

Procedimiento Greedy(var: vector, dom: matriz, ctr: vector, ctr2: matriz, individuo:individuo)

Borrar datos de las soluciones de individuo y reservamos memoria de nuevo
trans ← transistor random de la solución
elegir frecuencia random para dicho transmisor
asignar como posición de partida ese transmisor
Bucle Mientras que trans sea diferente del transmisor inicial
    pos ← trans
    penalización ← máxima
    freq ← 0
    j ← random freq
    aleatorio ← aleatorio igual a 0 o 1
    Si aleatorio igual a 0 entonces aleatorio ← -1 Fin_Si
    Hacer
        j ← j + aleatorio
        Comprobamos que j no se salga del dominio de frecuencias
        Mandamos la nueva solución a ComprobarGrasp
        Si la solución mejora a la mejor solución anterior, la guardamos
        y la incluimos en nuestro vector solución
    Mientras que no sea j de nuevo la frecuencia inicial
Fin_Mientras
Actualizamos el coste de ese individuo tras comprobar rellenar todas sus soluciones

Algoritmo genético generacional

Se trata de un algoritmo de carácter elitista, en el que los hijos de cada generación, reemplazan a los padres, salvo, al mejor de todos ellos, que por el hecho de serlo, es respetado durante la siguiente generación. Es aquí cuando se verá el esquema de evolución de los generacionales con los distintos operadores de cruce.

Pseudocódigo del AGG

Procedimiento AGG (iteraciones: int, población: vector(individuo))
cont ← 0
Mientras que cont sea distinto de iteraciones
    MejorPadre ← Guardamos el mejor padre de la población actual
    Realizamos un torneo binario, y lo asignamos a un vector de padres
    Bucle desde ← 0 hasta número de padres
        Aplicamos el operador de cruce correspondiente, bien el Cruce 2 puntos o bien el cruce BLX
        Comprobamos el coste de las soluciones y los añadimos a un vector de hijos
    Fin_Bucle
    Mandamos a los hijos a una competición con los padres, a fin de seleccionar a los mejores.
    Mandamos a un elemento aleatorio de la población al operador de mutación
    Comprobamos si se ha de reinicializar por estancamiento
    Una vez hecho esto mantenemos el elitismo y ordenamos el vector solución definitivo
Fin_Mientras

Algoritmo genético estacionario

Se trata de un algoritmo basado en la selección únicamente de dos cromosomas para las operaciones de cruce y mutación. Los nuevos hijos, sustituyen a los padres, en caso de ser mejores que estos, en caso contrario, serán desechados. Es aquí cuando se verá el esquema de evolución de los estacionarios con los distintos operadores de cruce.

Pseudocódigo del AGE

```
Procedimiento AGE (iteraciones: int, población: vector(individuo))
cont ← 0
Mientras que cont sea distinto de iteraciones
        MejorPadre ← Guardamos el mejor padre de la población actual
        Realizamos un torneo binario, y lo asignamos a un vector de padres
        Bucle desde ← 0 hasta número de padres
                establecemos dos hijos y aplicamos el operador de cruce
                correspondiente, bien el Cruce 2 puntos o bien el cruce BLX
                establecemos dos aleatorios
                Si los aleatorios son menores que 0,2 mutamos, sino,
                comprobamos el coste de la solución
                Repetimos para ambos hijos
                Incluimos la solución en nuestros vectores de hijos
        Fin_Bucle
        Mandamos a los hijos a una competición con los padres, a fin de
seleccionar a los mejores.
        Comprobamos si se ha de reinicializar por estancamiento
Fin_Mientras
```

4. Descripción del mecanismo de selección de los individuos

Descripción

Para realizar una nueva generación se deben de elegir una serie de padres que serán cruzados para obtener unos hijos que sustituirán a algún miembro de la población si son mejores que estos.

El método de elección depende del algoritmo generacional que estemos usando:

Selección en Algoritmo Estacionario

En el algoritmo estacionario, solamente se cruzan dos padres obteniendo así solamente dos hijos.

Para la elección de dichos padres realizamos un torneo binario, es decir, elegimos 4 padres al azar con los cuales formamos 2 parejas, tras esto de cada pareja elegimos un ganador con un número aleatorio.

Los dos padres ganadores se cruzan utilizando un cruce BLX o dos puntos, como ya explicaremos posteriormente.

Procedimiento torneoBinarioAGE(población: vector(individuo))

Reservamos memoria para almacenar a los padres;

padre1 ← insertar un individuo de población con rand()
padre2 ← insertar un individuo de población con rand()
padre3 ← insertar un individuo de población con rand()
padre4 ← insertar un individuo de población con rand()

aleatorio1 ← aleatorio entre 0 y 1;
aleatorio2 ← aleatorio entre 0 y 1;

Si aleatorio1 es 0
    padres ← insertamos padre1 en padres;
Fin_Si
Si aleatorio1 es 1

```
        padres ← insertamos padre2 en padres;
Fin_Si
Si aleatorio2 es 0
        padres ← insertamos padre3 en padres;
Fin_Si
Si aleatorio2 es 1
        padres ← insertamos padre4 en padres;
Fin_Si

Devolvemos padres;
```

Selección en Algoritmo Generacional

En este algoritmo en cada generación se cruza un 70% de la población, como en nuestro caso la población es de 50 individuos, sólo se cruzarán 35 miembros pero al necesitar parejas, elegiremos 36 individuos, 18 parejas.

Para la elección de cada padre, elegimos con un número aleatorio al ganador que posteriormente será cruzado con otro padre para obtener así dos hijos. Los nuevos hijos competirán con los peores padres.

```
Procedimiento torneoBinarioAGG(población: vector(individuo))

        Reservar memoria para guardar los padres;

        límite ← guardar el tamaño de la población * 0.7;

        Si límite es impar
                limite ← limite + 1;
        Fin_Si

        Bucle de i ← 0 hasta límite
                padres ← padre elegido aleatoriamente de población;
        Fin_Bucle;

        Devolver padres;
```

5. Descripción de los métodos de cruce y mutación

Descripción

Para obtener hijos debemos de cruzar a los padres que hemos elegido anteriormente, estos cruces se realizarán de dos maneras, siguiendo un cruce por dos puntos o un cruce BLX, una vez obtenidos los hijos debemos comprobar su coste y hacerles competir con los peores individuos de nuestra población actual.

En la mutación, mutamos un miembro de la población tras la competición, en el algoritmo generacional y en el algoritmo estacionario, mutamos los dos hijos obtenidos en cada generación un 2% de las veces.

La mutación consiste en cambiar las frecuencias aleatoriamente de un 10% de los transmisores del individuo elegido para la mutación.

Cruce dos puntos

Este cruce consiste en, dados dos padres, intercambiar sus frecuencias entre dos puntos. En este caso, dicho punto de cambio es el primer tercio y el segundo tercio.

El hijo1 estará formado por el primer y tercer tercio de frecuencias del padre1 y el segundo tercio de frecuencias del padre2. Como consecuencia el hijo2 estará formado por el contrario del hijo1, el segundo tercio del padre1 y el primer y tercer tercio del padre2.

Procedimiento cruceDosPuntos(padre1: individuo, padre2: individuo, hijo1: individuo, hijo2: individuo)

```
    hijo1 ← limpiamos el contenido de hijo1 e insertar padre1;
    hijo2 ← limpiamos el contenido de hijo2 e insertar padre2;

    Reservar memoria para una variable aux;

    Bucle de i ← tamaño/3 hasta tamaño*⅔
        aux ← frecuencia hijo1 en i;
        frecuencia hijo1 en i ←  frecuencia hijo2 en i;
        frecuencia hijo2 en i ← aux;
    FinBucle;
```

//Los hijos son devueltos por parámetro.

Cruce BLX

El cruce BLX se basa en buscar una frecuencia dentro del rango de las frecuencias de los padres (explotación) y un alfa fuera de este rango (exploración), este alfa se calcula multiplicando por el tamaño de la distancia entre las dos frecuencias padre.

El valor alfa es elegido en esta práctica es de 0.5.

Procedimiento cruceBLX(var: vector, dom: vector(vector(int)), padre1: individuo, padre2: individuo, hijo1: individuo, hijo2: individuo)

hijo1 ← limpiar el contenido de hijo1;
hijo2 ← limpiar el contenido de hijo2;

Declaramos alfa = 0.5, mayor: float, menor: float y diff: float
Declaramos freqPadre1: int y freqPadre2: int;

Bucle de i ← 0 hasta numero de transmisores
freqPadre1 ← guardamos la frecuencia de padre1 en i;
freqPadre2 ← guardamos la frecuencia de padre2 en i;

mayor ← guarda la frecuencia mayor de las dos;
menor ← guarda la frecuencia menor de las dos;

diff ← (mayor-menor)*alfa; //Guarda el rango de exploracion.

//Almacenamos los nuevos límites del rango
mayor ← mayor+diff;
menor ← menor-diff;

//Guarda apuntadores a las mínimas frecuencias encontradas dentro de los límites.
vector<int>::iterator up ← lower_bound(mayor);
vector<int>::iterator low ← lower_bound(menor);

//Para obtener las posiciones restamos con var.begin()
posFreqMayor ← (up-dom->at(var->at(i)).begin());
posFreqMenor ← (low-dom->at(var->at(i)).begin());

//Insertamos los aleatorios obtenidos en los hijos.

freq de hijo1 en i ← frecuencia aleatoria entre posFreqMenor y posFreqMayor;
freq de hijo2 en i ← frecuencia aleatoria entre posFreqMenor y posFreqMayor;
FinBucle;

//Hijo1 e hijo2 se devuelven como parámetro ya modificados.

Mutación

Como ya hemos explicado el proceso de mutación consiste en cambiar un 10% de las frecuencias de los transmisores de este, por frecuencias aleatorias. El porcentaje de selección de un individuo es un 2% de la población.

```
Procedimiento mutación (var: vector, dom: matriz, individuo: individuo)
    Bucle desde i ← 0 hasta el tamaño de la solución hacer
        obtenemos un aleatorio entre 0 y 1 con 4 decimales.
        Si el aleatorio es menor que 0,1 entonces
            insertamos en la frecuencia un nuevo valor aleatorio,
            extraido del dominio de frecuencias del transmisor actual
        Fin_Si
    Fin_Bucle
    Actualizamos el coste del individuo llamando a comprobar
```

6. Descripción de la evolución generacional del algoritmo y del esquema de reemplazamiento

Esquema de reemplazamiento

En la función representada a continuación se establece el mecanismo general por el cual se puede proceder realmente a la sustitución de una solución por otra catalogada como mejor. Además se puede producir el reemplazamiento también a la hora de reiniciar.

```
Procedimiento competición (hijos: vector(individuo), población: vector(individuo))

    ordenar el vector de hijos
    pPoblación ← tamaño de la población
    Bucle desde i ← tamaño de la población hasta i mayor o igual que 0 hacer
        Si coste del hijo de i es menor que el coste de la pPoblación entonces
            copiamos la solución del hijo de i, en población de pPoblación
            copiamos el coste del hijo de i, en en población de pPoblación
            pPoblación ← pPoblación - 1
        Fin_Si
    Fin_Bucle
    Ordenamos la población
```

Reinicialización

En el caso particular de la renicialización, se llevará a cabo una sustitución completa o casi completa de los cromosomas que conforman nuestra población. A saber:

En caso del AGG, la reinicialización se produce cuando: Se alcanza un 80% de cromosomas con igual coste y genes, o cuando no se mejora cada 20 generaciones.

En caso del AGE, esta solo se produce al llegar al 80% de igualdad genética.

Pseudocódigo de la reinicialización

```
Procedimiento reiniciar (población: vector(individuo))

    Creamos un contador de tipo vector
    Bucle desde i ← 0 hasta tamaño de la población hacer
```

```
        individuo ← población en i
        metido igual a falso
        Bucle desde j ← 0 hasta tamaño del contador hacer
                Si solucion del indiviuo es igual a la del contador en j entonces
                        Incrementamos contador
                        metido igual a verdadero
                Fin_Si
        Fin_Bucle
        Si no metido entonces
                introducimos el individuo en el contador
        Fin_Si
Fin_Bucle
Bucle desde i← 0 hasta i menor que tamaño del contador
        Si el contador es mayor del 80% entonces
                Devolver verdadero
        Fin_Si
Fin_Bucle
Devolver falso
```

7. Desarrollo de la práctica

Entorno de realización

La práctica ha sido realizada en C++, y los resultados obtenidos han sido en el entorno de programación CodeBlocks, usando el compilador MinGW. El sistema operativo usado ha sido Windows 8.1.

Manual de usuario

Cuando se ejecuta el programa, aparece la consola mostrando los archivos a los que tenemos acceso para trabajar sobre ellos, nos movemos con las flechas y para seleccionar el archivo pulsamos ENTER.

Una vez elegido el archivo el programa nos muestra una serie de información antes de comenzar a trabajar, el número de transmisores, el número de condiciones y el nombre del archivo sobre el que trabajamos.

El programa comienza a ejecutar los distintos algoritmos de manera concurrente, ya que usamos hilos, para ahorrar tiempo en la obtención de los resultados.

Al finalizar, el programa nos muestra los datos necesarios para la práctica, los costes del mejor individuo obtenido en la población de cada algoritmo, además del primer coste del Greedy.

8. Análisis de resultados

Descripción de los casos y parámetros utilizados

El problema de la Asignación de Frecuencias con Mínimas Interferencias (MI-FAP) está entre los problemas clásicos de optimización en el mundo de las comunicaciones. Existen algunas variantes que hemos podido constatar:

-Minimum Order FAP (MO-FAP): minimizar las frecuencias que se usan en la red.
-Minimum Span FAP (MS-FAP): minimizar la diferencia entre la frecuencia más alta y más baja.
-Minimum Blocking FAP (MB-FAP): minimizar el bloqueo en la red.
-Minimum Interference FAP (MI-FAP).

Para elegir el método heurístico a seguir escogeremos entre:

1) Heurísticas aleatorias-simples: Son rápidas (Greedy).

2) Heurísticas basadas en poblaciones: Es el caso de los algoritmos evolutivos genéticos como el AGG y el AGE, estos algoritmos son más lentos y requieren de un consumo mayor de recursos del sistema.

Las instancias utilizadas han sido:
1. CELAR 06 con 200 transmisores
2. CELAR 07 con 400 transmisores
3. CELAR 08 con 916 transmisores
4. CELAR 09 con 680 transmisores
5. CELAR 10 con 680 transmisores
6. GRAPH 05 con 200 transmisores
7. GRAPH 06 con 400 transmisores
8. GRAPH 07 con 400 transmisores
9. GRAPH 11 con 680 transmisores
10. GRAPH 12 con 680 transmisores

Resultados Greedy

Greedy	CELAR 06		CELAR 07		CELAR 08		CELAR 09		CELAR 10	
	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo
Ejecución 1	216,00	0,00	672,00	0,00	1487,00	0,00	1271,00	0,00	1271,00	0,00
Ejecución 2	216,00	0,00	672,00	0,00	1487,00	0,00	1271,00	0,00	1271,00	0,00
Ejecución 3	216,00	0,00	708,00	0,00	1542,00	0,00	1254,00	0,00	1248,00	0,00
Ejecución 4	216,00	0,00	672,00	0,00	1496,00	0,00	1271,00	0,00	1305,00	0,00
Ejecución 5	216,00	0,00	708,00	0,00	1487,00	0,00	1254,00	0,00	1271,00	0,00
Mejor	216,00	0,00	672,00	0,00	1487,00	0,00	1254,00	0,00	1248,00	0,00
Peor	216,00	0,00	708,00	0,00	1542,00	0,00	1271,00	0,00	1305,00	0,00
Media	216,00	0,00	686,40	0,00	1499,80	0,00	1264,20	0,00	1273,20	0,00
Desv. típica	0,00	0,00	19,72	0,00	23,91	0,00	9,31	0,00	20,38	0,00

GRAPH 05		GRAPH 06		GRAPH 07		GRAPH 11		GRAPH 12	
Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo
323,00	0,00	589,00	0,00	708,00	0,00	1417,00	0,00	1486,00	0,00
323,00	0,00	560,00	0,00	746,00	0,00	1417,00	0,00	1486,00	0,00
350,00	0,00	589,00	0,00	708,00	0,00	1426,00	0,00	1486,00	0,00
323,00	0,00	589,00	0,00	738,00	0,00	1417,00	0,00	1486,00	0,00
340,00	0,00	547,00	0,00	708,00	0,00	1405,00	0,00	1486,00	0,00
323,00	0,00	547,00	0,00	708,00	0,00	1405,00	0,00	1486,00	0,00
350,00	0,00	589,00	0,00	746,00	0,00	1426,00	0,00	1486,00	0,00
331,80	0,00	574,80	0,00	721,60	0,00	1416,40	0,00	1486,00	0,00
12,56	0,00	19,98	0,00	18,84	0,00	7,47	0,00	0,00	0,00

Descripción de los resultados:

El Greedy usado es una mejora del greedy tradicional, el cual obtiene resultados iniciales bastante buenos. Gracias a estos resultados, obtenemos una partida inicial aceptable de cara a los siguientes algoritmos.

Resultados AGG 2 puntos

AGG dos puntos	CELAR 06		CELAR 07		CELAR 08		CELAR 09		CELAR 10	
	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo
Ejecución 1	216,00	33,84	672,00	65,41	1487,00	157,69	1271,00	112,87	1271,00	94,87
Ejecución 2	234,00	32,99	648,00	64,78	1454,00	158,10	1282,00	113,04	1270,00	94,66
Ejecución 3	208,00	33,45	657,00	65,12	1492,00	157,50	1271,00	112,45	1301,00	94,22
Ejecución 4	212,00	33,23	680,00	65,32	1481,00	157,89	1274,00	112,65	1269,00	94,91
Ejecución 5	224,00	33,87	672,00	65,31	1456,00	157,85	1236,00	112,86	1285,00	94,71
Mejor	208,00	32,99	648,00	64,78	1454,00	157,50	1236,00	112,45	1269,00	94,22
Peor	234,00	33,87	680,00	65,41	1492,00	158,10	1282,00	113,04	1301,00	94,91
Media	218,80	33,48	665,80	65,19	1474,00	157,81	1266,80	112,77	1279,20	94,67
Desv. típica	10,35	0,38	12,97	0,25	17,79	0,23	17,80	0,23	13,83	0,27

GRAPH 05		GRAPH 06		GRAPH 07		GRAPH 11		GRAPH 12	
Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo
215,00	32,61	547,00	55,17	640,00	53,63	1270,00	85,83	1321,00	100,61
261,00	33,54	528,00	54,35	652,00	53,54	1248,00	83,21	1352,00	108,32
241,00	32,54	524,00	55,89	640,00	53,24	1250,00	85,23	1298,00	105,94
224,00	36,94	547,00	51,94	638,00	51,95	1199,00	89,21	1324,00	121,32
198,00	35,24	512,00	55,12	678,00	58,45	1238,00	91,01	1318,00	114,21
198,00	32,54	512,00	51,94	638,00	51,95	1199,00	83,21	1298,00	100,61
261,00	36,94	547,00	55,89	678,00	58,45	1270,00	91,01	1352,00	121,32
227,80	34,17	531,60	54,49	649,60	54,16	1241,00	86,90	1322,60	110,08
24,20	1,89	15,24	1,53	16,82	2,49	26,19	3,15	19,33	7,96

Descripción de los resultados:

El AGG 2 puntos realiza una ejecución algo más lenta que el Greedy usado anteriormente, pero más efectiva, ya que obtiene resultados considerablemente mejores. El AGG usa una selección de 50 cromosomas de la población inicial, la cual será costosa en su evaluación. Este AGG hace reinicializaciones de forma frecuente en busca de un valor local que mejore hasta el guardado por el momento, es precisamente este sistema el que habilita a el AGG con sistema de cruce por dos puntos a obtener dichas soluciones tan buenas.

Experimentalmente hemos observado que por regla general el BLX obtiene mejores soluciones que el 2 puntos, aunque como media, ambos obtienen resultados similares. El BLX podemos concluir que sirve para diversificar ampliamente los resultados obtenidos.

Resultados AGG BLX

AGG BLX	CELAR 06		CELAR 07		CELAR 08		CELAR 09		CELAR 10	
	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo
Ejecución 1	216,00	24,80	605,00	57,90	1259,00	157,69	1127,00	83,36	1127,00	71,20
Ejecución 2	212,00	23,93	601,00	57,84	1278,00	158,90	1136,00	83,45	1189,00	71,23
Ejecución 3	221,00	24,35	617,00	58,34	1234,00	157,98	1143,00	83,12	1103,00	71,01
Ejecución 4	231,00	24,32	640,00	57,68	1296,00	157,65	1102,00	83,94	1189,00	71,35
Ejecución 5	211,00	23,99	596,00	58,49	1237,00	157,45	1144,00	83,34	1134,00	71,46
Mejor	211,00	23,93	596,00	57,68	1234,00	157,45	1102,00	83,12	1103,00	71,01
Peor	231,00	24,80	640,00	58,49	1296,00	158,90	1144,00	83,94	1189,00	71,46
Media	218,20	24,28	611,80	58,05	1260,80	157,93	1130,40	83,44	1148,40	71,25
Desv. típica	8,17	0,35	17,57	0,35	26,57	0,57	17,27	0,30	38,80	0,17

GRAPH 05		GRAPH 06		GRAPH 07		GRAPH 11		GRAPH 12	
Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo
263,00	21,60	558,00	32,74	668,00	31,84	1333,00	68,55	1453,00	59,60
254,00	21,54	541,00	35,64	665,00	30,99	1339,00	65,21	1584,00	62,14
267,00	22,31	511,00	31,87	678,00	32,70	1387,00	68,91	1487,00	57,24
199,00	26,48	527,00	32,24	612,00	35,24	1347,00	61,25	1369,00	61,24
214,00	25,32	555,00	33,79	668,00	31,87	1451,00	70,12	1422,00	63,49
199,00	21,54	511,00	31,87	612,00	30,99	1333,00	61,25	1369,00	57,24
267,00	26,48	558,00	35,64	678,00	35,24	1451,00	70,12	1584,00	63,49
239,40	23,45	538,40	33,26	658,20	32,53	1371,40	66,81	1463,00	60,74
30,86	2,29	19,67	1,52	26,29	1,63	49,24	3,60	80,36	2,41

Descripción de resultados:

El AGG BLX realiza al igual que en el anterior operador de cruce, una ejecución algo más lenta que el Greedy usado anteriormente, pero más efectiva, ya que obtiene resultados considerablemente mejores. Este AGG hace reinicializaciones de forma frecuente en busca de un valor local que mejore hasta el guardado por el momento, es precisamente este sistema el que habilita a el AGG con sistema de cruce por dos puntos a obtener dichas soluciones tan buenas.

Tanto en el caso de el 2 puntos como en el de BLX para este algoritmo AGG, la reinicialización se da principalmente por 20 generaciones sin mejorar que no por 80% de igualdad genética en sus cromosomas.

Resultados AGE 2 puntos

AGE dos puntos	CELAR 06		CELAR 07		CELAR 08		CELAR 09		CELAR 10	
	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo
Ejecución 1	216,00	33,84	672,00	65,41	1487,00	157,69	1271,00	112,87	1271,00	94,87
Ejecución 2	234,00	32,99	648,00	64,78	1454,00	158,10	1282,00	113,04	1270,00	94,66
Ejecución 3	208,00	33,45	657,00	65,12	1492,00	157,50	1271,00	112,45	1301,00	94,22
Ejecución 4	212,00	33,23	680,00	65,32	1481,00	157,89	1274,00	112,65	1269,00	94,91
Ejecución 5	224,00	33,87	672,00	65,31	1456,00	157,85	1236,00	112,86	1285,00	94,71
Mejor	208,00	32,99	648,00	64,78	1454,00	157,50	1236,00	112,45	1269,00	94,22
Peor	234,00	33,87	680,00	65,41	1492,00	158,10	1282,00	113,04	1301,00	94,91
Media	218,80	33,48	665,80	65,19	1474,00	157,81	1266,80	112,77	1279,20	94,67
Desv. típica	10,35	0,38	12,97	0,25	17,79	0,23	17,80	0,23	13,83	0,27

GRAPH 05		GRAPH 06		GRAPH 07		GRAPH 11		GRAPH 12	
Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo
323,00	36,12	589,00	53,84	708,00	52,85	1417,00	88,46	1486,00	87,84
354,00	38,94	580,00	53,00	751,00	50,84	1479,00	85,23	1451,00	84,32
318,00	35,12	495,00	55,24	708,00	52,85	1428,00	84,69	1463,00	85,96
318,00	35,27	599,00	52,47	723,00	49,57	1399,00	89,21	1412,00	81,33
318,00	35,97	524,00	51,12	694,00	55,31	1458,00	81,25	1498,00	88,47
318,00	35,12	495,00	51,12	694,00	49,57	1399,00	81,25	1412,00	81,33
354,00	38,94	599,00	55,24	751,00	55,31	1479,00	89,21	1498,00	88,47
326,20	36,28	557,40	53,13	716,80	52,28	1436,20	85,77	1462,00	85,58
15,69	1,55	45,41	1,54	21,70	2,19	32,12	3,20	33,52	2,88

Descripción de resultados:

En este caso, a diferencia del AGG, el AGE solo usa 2 pares de cromosomas para establecer el proceso de cruce y mutación, donde uno de los ganadores mutará y entrará en competición con la población de padres anterior. La reinicialización se produce menos veces, ya que se cambia la información genética de menos cromosomas. El mayor problema es el tiempo consumido, en relación a los resultados obtenidos.

Resultados AGE BLX

AGE BLX	CELAR 06		CELAR 07		CELAR 08		CELAR 09		CELAR 10	
	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo
Ejecución 1	216,00	36,21	672,00	70,03	1487,00	177,54	1271,00	121,21	1271,00	103,36
Ejecución 2	232,00	36,48	643,00	70,32	1483,00	177,91	1273,00	122,07	1273,00	102,94
Ejecución 3	230,00	36,01	678,00	69,67	1543,00	177,52	1243,00	121,41	1240,00	103,12
Ejecución 4	216,00	36,23	672,00	70,05	1468,00	177,54	1263,00	121,92	1271,00	103,04
Ejecución 5	210,00	36,33	659,00	69,98	1483,00	177,43	1282,00	121,02	1298,00	102,98
Mejor	210,00	36,01	643,00	69,67	1468,00	177,43	1243,00	121,02	1240,00	102,94
Peor	232,00	36,48	678,00	70,32	1543,00	177,91	1282,00	122,07	1298,00	103,36
Media	220,80	36,25	664,80	70,01	1492,80	177,59	1266,40	121,53	1270,60	103,09
Desv. típica	9,65	0,17	14,02	0,23	28,99	0,19	14,72	0,45	20,57	0,17

GRAPH 05		GRAPH 06		GRAPH 07		GRAPH 11		GRAPH 12	
Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo
323,00	38,64	589,00	59,09	708,00	57,88	1417,00	96,01	1486,00	97,65
356,00	35,41	547,00	54,21	754,00	55,68	1478,00	95,24	1475,00	91,26
314,00	36,87	521,00	52,87	714,00	51,89	1426,00	98,31	1463,00	97,65
354,00	31,20	569,00	58,63	723,00	52,63	1496,00	93,56	1425,00	93,25
326,00	33,26	594,00	55,21	766,00	55,68	1423,00	97,95	1498,00	95,68
314,00	31,20	521,00	52,87	708,00	51,89	1417,00	93,56	1425,00	91,26
356,00	38,64	594,00	59,09	766,00	57,88	1496,00	98,31	1498,00	97,65
334,60	35,08	564,00	56,00	733,00	54,75	1448,00	96,21	1469,40	95,10
19,15	2,93	30,36	2,74	25,57	2,46	36,31	1,96	28,01	2,81

Descripción de resultados:

Es algo más rápido que el BLX en AGG, pero al ser una mutación y un cruce de un número de elementos algo más reducido, se tarda más en llegar a obtener los resultados para reinicializar. Como en el BLX se usa el alfa para diversificar/intensificar, para mejorar los resultados, no sería mala opción modificar dicho valor alfa para intensificar más en la exploración, y así obtener mejores resultados.

9. Comparativa de algoritmos

Descripción de resultados totales:

Como podemos observar el uso de un Greedy que calcula buenos valores, nos ayuda a que los algoritmos generacionales partan de unas buenas soluciones para comenzar a trabajar.

Tenemos dos diferencias notables en nuestros algoritmos, una el número de hijos que son obtenidos en cada generación, (70% de la población en el generacional, 2 hijos en el estacionario) y dos la manera de cruzar nuestros padres, usando cruce de dos puntos o BLX.

En el caso de cruce de dos puntos no se generan nuevos valores solamente se intercambian valores de un padre con otro por tanto las mejoras se basan en las mutaciones que puedan ocurrir. Cuando se cruzan usando el BLX se obtiene un valor aleatorio dentro del rango de las frecuencias de los padres (explotación) y un alfa % fuera de dicho rango (exploración), el algoritmo que usa este cruce obtiene mejores resultados, ya que no depende solo de la mutación.

	CELAR 06		CELAR 07		CELAR 08		CELAR 09		CELAR 10	
	Z	Tiempo	*Z*	Tiempo	*Z*	Tiempo	*Z*	Tiempo	*Z*	Tiempo
AGG DP (med)	218,80	33,48	665,80	65,19	1474,00	157,81	1266,80	112,77	1279,20	94,67
AGG DP (desv)	10,35	0,38	12,97	0,25	17,79	0,23	17,80	0,23	13,83	0,27
AGG BLX (med)	218,20	24,28	611,80	58,05	1260,80	157,93	1130,40	83,44	1148,40	71,25
AGG BLX (desv)	8,17	0,35	17,57	0,35	26,57	0,57	17,27	0,30	38,80	0,17
AGE DP (med)	218,80	33,48	665,80	65,19	1474,00	157,81	1266,80	112,77	1279,20	94,67
AGE DP (desv)	10,35	0,38	12,97	0,25	17,79	0,23	17,80	0,23	13,83	0,27
AGE BLX (med)	220,80	36,25	664,80	70,01	1492,80	177,59	1266,40	121,53	1270,60	103,09
AGE BLX (desv)	9,65	0,17	14,02	0,23	28,99	0,19	14,72	0,45	20,57	0,17
Greedy (med)	218,20	24,28	611,80	58,05	1260,80	157,93	1130,40	83,44	1148,40	71,25
Greedy (desv)	8,17	0,35	17,57	0,35	26,57	0,57	17,27	0,30	38,80	0,17

GRAPH 05		GRAPH 06		GRAPH 07		GRAPH 11		GRAPH 12	
Z	Tiempo	*Z*	Tiempo	*Z*	Tiempo	*Z*	Tiempo	*Z*	Tiempo
227,80	34,17	531,60	54,49	649,60	54,16	1241,00	86,90	1322,60	110,08
24,20	1,89	15,24	1,53	16,82	2,49	26,19	3,15	19,33	7,96
239,40	23,45	538,40	33,26	658,20	32,53	1371,40	66,81	1463,00	60,74
30,86	2,29	19,67	1,52	26,29	1,63	49,24	3,60	80,36	2,41
326,20	36,28	557,40	53,13	716,80	52,28	1436,20	85,77	1462,00	85,58
15,69	1,55	45,41	1,54	21,70	2,19	32,12	3,20	33,52	2,88
334,60	35,08	564,00	56,00	733,00	54,75	1448,00	96,21	1469,40	95,10
19,15	2,93	30,36	2,74	25,57	2,46	36,31	1,96	28,01	2,81
239,40	23,45	538,40	33,26	658,20	32,53	1371,40	66,81	1463,00	60,74
30,86	2,29	19,67	1,52	26,29	1,63	49,24	3,60	80,36	2,41

10. Referencias bibliográficas

E.-G. Talbi. Metaheuristics. From design to implementation. Wiley, 2009

Sem03-Problemas-Poblaciones-MHs-17-18.pdf