UNIVERSIDAD DE JAÉN
ESCUELA POLITÉCNICA SUPERIOR DE JAÉN

Práctica 3:
Búsquedas híbridas para el Problema de Asignación de Frecuencias con Mínimas Interferencias (Mi-Fap)

Curso 2017-2018

Grado de Ingeniería Informática

Algoritmos considerados:

Algoritmo Greedy

AM-(1,1.0)

AM-(1,0.1)

AM-(1,0.1mej)

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, la cual, consistirá en seleccionar una frecuencia de las disponibles para cada transmisor. A cada uno de los transmisores, se le asignará una frecuencia de transmisión dentro de su dominio, de forma aleatoria y mediante una serie de semillas pre-establecidas en el inicio del programa.

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
```

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

Busqueda local

El algoritmo de búsqueda local se usará dependiendo del algoritmo memético que se esté evaluando en todo momento. Sobre cada uno de los individuos seleccionados de la población se aplicará este algoritmo, el cual seguirá el método del primer mejor. En la generación de cada vecino, se determina un número al azar para decidir el transmisor por el cual empezar la búsqueda. Posteriormente se intercambia la frecuencia asignada al transmisor correspondiente por alguna de las frecuencias disponibles dentro de su propio dominio de frecuencias. Si un vecino mejora el coste de la solución actual, se incluirá en la solución se actualizará dicho coste. Para ello se hace uso de la factorización, esto es restando el coste de los transmisores implicados y sus restricciones, respecto de la frecuencia actual y sumando el coste de la nueva frecuencia asignada, a el resultado de la función objetivo.

Representación de la factorización:

$$\Delta f\,(\pi') = -\sum_{j=0\ j!=a}^{n} MI(TRX_a\ TRXj) + \sum_{j=0\ j!=b}^{n} MI(TRX_b\ TRXj)$$

Donde: $f(\pi)$ es: $f(\pi') = f(\pi) + \Delta f(\pi)$

Pseudocódigo del método de búsqueda

Procedimiento BúsquedaLocal (var: vector, ctr: vector, ctr2: matriz, dom: matriz, sol: vector, iteraciones: int)

solAux ← sol
penalización ← Comprobar
Bucle i ← 0 hasta iteraciones hacer
    elegir transmisor de comprobación (Transmisor)
    elegir sentido de comprobación de vecinos en las frecuencias
    elegimos una nueva frecuencia con la que comprobar una nueva solución (FrecInicial)
    si el vector de dominio de frecuencias se desborda por un lateral, colocamos el iterador a el inicio del vector
    penalizaciónNew ← Infinito
    Mientras que FrecInicial y la penalización sea menor que infinito
        penalizaciónNew ← ComprobarFact
        elegimos la nueva frecuencia y la añadimos a la solución

```
            Incrementamos las iteraciones ya comprobadas
        Fin_Mientras
        Si penalizaciónNew menor que penalización
            sol = solAux;
        Fin_Si
Fin_Bucle
```

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
```

4. Descripción 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 en esta práctica siempre es el mismo, ya que, solo utilizaremos el algoritmo generacional

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 20 individuos, sólo se cruzarán 14 miembros, es decir, 7 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 utilizando el torneo binario. El cruce que utilizaremos es siempre el cruce BLX, una vez obtenidos los dos 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 haber realizado la competición de los nuevos hijos con los peores de la población. Mutamos un individuo en cada generación.

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

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 del método de hibridación

Métodos de hibridación

Existirán tres métodos de hibridación a evaluar, estos métodos consistirán en una mezcla del uso de los algoritmos generacionales genéticos usados anteriormente con el uso del algoritmo de búsqueda local, el cual será aplicado cada 10 generaciones del algoritmo generacional.

Los métodos de hibridación son los siguientes:

Método 1: AM-(10,1.0): Cada 10 generaciones, aplicar la BL sobre todos los cromosomas de la población.

Método 2: AM-(10,0.1): Cada 10 generaciones, aplicar la BL sobre un subconjunto de cromosomas de la población seleccionado aleatoriamente con probabilidad pLS igual a 0.1 para cada cromosoma.

Método 3: AM-(10,0.1mej): Cada 10 generaciones, aplicar la BL sobre los 0.1·N mejores cromosomas de la población actual (N es el tamaño de ésta).

Además siempre tendremos en cuenta que la búsqueda local se aplicará con un método de baja intensidad (200 iteraciones).

Descripción general del método de hibridación paso a paso:

Inicialmente, generamos una población de individuos haciendo uso de nuestra función inicializar, que nos inserta en una población tantos individuos como tamaño de la población tenemos.

Tras esto, realizamos un torneo binario para elegir los padres que serán cruzados, y los cruzamos haciendo uso de nuestro cruce BLX, al finalizar y habiendo obtenido ya nuestros dos hijos, los hacemos competir con los peores individuos de nuestra población, y por último mutamos un individuo aleatorio.

Comprobaremos si se debe llamar a reinicializar o no tras haber terminado la competición y la mutación, y por último si se da el caso de que llevamos 10 generaciones ejecutados realizaremos uno de nuestros “Métodos” descritos anteriormente dependiendo del tipo de algoritmo que vayamos a ejecutar pudiendo ser realizar búsqueda local a todos los individuos, solo a un 10 % aleatoriamente o al 10 % de los mejores individuos.

1-Pseudocódigo de la hibridación AM-(10, 1.0)

En este caso el algoritmo de hibridación, se aplicará solo sobre toda la población mandando todos sus individuos al búsqueda local.

```
Procedure AGGBLX1(cont: int, poblacion: vector (poblacion))
    generacion → 1
    cont → 0
    contComprobaciones → 20
    Mientras contComprobaciones sea menor que el numero de comprobaciones
        mejorIndividuo → primer elemento de poblacion
        realizamos el torneo binario
        Para cada i → 0 hasta el tamaño del vector de padres hacer
            hijo1, hijo2
            llamamos al operador de cruce BLX con el hijo1 y el hijo2
            calculamos el coste de cada uno de los hijos llamando a
            comprobar con la nueva solución
            registramos los nuevos hijos en el vector de hijos general
        Fin_Para
        Llamamos a competición al vector de hijos con el vector poblacion
        Elegimos un aleatorio y lo mandamos a mutar
        contComprobaciones ← contComprobaciones +1
        Si reiniciar → true
            Llamamos a inicializacion
            contComprobaciones ← contComprobaciones +19
            generacion → 1
        Fin_Si
        Cambiamos el último elemento de la poblacion por el mejor individuo
        para mantener el elitismo
        Ordenamos la poblacion
        Si generacion es igual a 10 /*Método de hibridación*/
            Para i → 0 hasta tamaño de poblacion hacer
                iteraciones → 200
                Llamamos al búsqueda local
                contComprobaciones ← contComprobaciones +200
                comprobamos el nuevo coste de la poblacion y lo
                actualizamos
            Fin_Para
            generacion → 1
        Fin_Si
        generacion → generacion +1
```

```
        cont → cont +1
    Fin_Mientras
```

2-Pseudocódigo de la hibridación AM-(10, 0.1)

En este caso el algoritmo de hibridación, se aplicará solo sobre un 10% aleatorio de la población, la cual al ser 20 individuos, hará que se mejoren 2 individuos de la población mandandolos al búsqueda local.

```
Si generacion es igual a 10 /*Método de hibridación*/
        Para i → 0 hasta tamaño*0.1 hacer
                iteraciones → 200
                aleatorio ← rand()%tamaño población;
                Llamamos al búsqueda local con el individuo en la
posicion aleatorio.
                contComprobaciones ← contComprobaciones +200
                comprobamos el nuevo coste de la poblacion y lo
                actualizamos
        Fin_Para
        generacion → 1
Fin_Si
```

3-Pseudocódigo de la hibridación AM-(10, 0.1mej)

En este caso el algoritmo de hibridación, se aplicará solo sobre un 10% de la población seleccionando solo los mejores individuos, la cual al ser 20 indivividuos, hará que se mejoren 2 individuos de la población mandándolos al búsqueda local. Estos individuos al encontrarse en una poblacion ordenada por coste, se localizaran al principio del vector de poblacion.

```
Si generacion es igual a 10 /*Método de hibridación*/
        Para i → 0 hasta tamaño * 0’1 hacer
                iteraciones → 200
                Llamamos al búsqueda local dandole el individuo i
                contComprobaciones ← contComprobaciones +200
                comprobamos el nuevo coste de la poblacion y lo
                actualizamos
        Fin_Para
        generacion → 1
Fin_Si
```

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

En esta práctica, los datos que se muestran por pantalla son irrelevantes, porque los resultados calculados del programa se guardan en un archivo txt que podrá ser accedido tras la finalización de este.

El programa se ejecuta tantas veces como archivos tenemos por cada semilla que necesitamos, es decir, el programa ejecuta los tres algoritmos por cada archivo (3 * 10 archivos = 30) y esto lo realiza tantas veces como semillas (30 * 5 semillas = 150), por tanto, ejecuta 150 algoritmos.

Como hacemos uso de hilos, cada ejecución de un algoritmo por archivo se realiza concurrentemente y el programa de durar varias horas pasa a durar algo más de una hora.

Un algoritmo de media tarda unos 300 segundos por archivo,lo que serían, 5 minutos. 150 * 5 minutos son 750 minutos, unas 12 horas y media. Sin embargo, al hacerlo con hilos, cada algoritmo se ejecuta de manera concurrente, y el programa nos tarda algo menos de 3 horas. La ejecución usando hilos sería 4 veces más rápida que la de sin hilos.

8. Análisis y comparativa 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 híbridas:

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 BLX 1

AGG BLX 1	CELAR 06		CELAR 07		CELAR 08		CELAR 09		CELAR 10	
	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo
Ejecución 1	159,00	0,80	573,00	149,24	1534,00	246,88	1005,00	181,26	858,00	178,13
Ejecución 2	182,00	0,80	643,00	131,26	1768,00	251,58	1135,00	179,50	1085,00	190,64
Ejecución 3	201,00	0,73	664,00	131,26	1560,00	251,58	1128,00	403,15	1102,00	182,82
Ejecución 4	164,00	0,77	620,00	135,94	1552,00	257,83	906,00	289,76	987,00	179,70
Ejecución 5	212,00	550,03	668,00	134,38	1387,00	248,50	1093,00	179,70	1042,00	184,39
Mejor	159,00	0,73	573,00	131,26	1387,00	246,88	906,00	179,50	858,00	178,13
Peor	212,00	550,03	668,00	149,24	1768,00	257,83	1135,00	403,15	1102,00	190,64
Media	183,60	110,62	633,60	136,42	1560,20	251,27	1053,40	246,67	1014,80	183,13
Desv. típica	22,92	245,63	38,90	7,45	135,95	4,19	97,29	99,52	98,27	4,87

GRAPH 05		GRAPH 06		GRAPH 07		GRAPH 11		GRAPH 12	
Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo
218,00	165,63	535,00	10,94	620,00	107,82	1369,00	176,81	1329,00	176,57
253,00	0,66	536,00	106,26	702,00	107,32	1196,00	170,32	1398,00	182,82
243,00	0,61	603,00	106,26	589,00	646,91	1248,00	171,88	1365,00	185,47
243,00	0,64	579,00	110,94	686,00	104,69	1308,00	168,76	1422,00	181,26
284,00	0,64	593,00	110,94	736,00	101,57	1305,00	174,80	1488,00	182,88
218,00	0,61	535,00	10,94	589,00	101,57	1196,00	168,76	1329,00	176,57
284,00	165,63	603,00	110,94	736,00	646,91	1369,00	176,81	1488,00	185,47
248,20	33,64	569,20	89,07	666,60	213,66	1285,20	172,51	1400,40	181,80
23,83	73,79	31,92	43,74	60,50	242,21	65,72	3,28	60,20	3,29

Resultados AGG BLX 2

AGG BLX 2	CELAR 06		CELAR 07		CELAR 08		CELAR 09		CELAR 10	
	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo
Ejecución 1	159,00	351,89	536,00	71,60	1217,00	345,95	855,00	370,97	827,00	559,83
Ejecución 2	180,00	237,27	493,00	539,27	1302,00	62,34	1109,00	110,99	917,00	949,79
Ejecución 3	161,00	368,14	555,00	268,78	1294,00	83,71	925,00	11,62	945,00	117,18
Ejecución 4	169,00	397,77	514,00	45,59	1330,00	112,07	820,00	120,25	890,00	944,25
Ejecución 5	182,00	136,58	491,00	491,56	1199,00	127,75	821,00	483,53	952,00	635,45
Mejor	159,00	517,80	491,00	45,59	1199,00	62,34	820,00	11,62	827,00	117,18
Peor	182,00	397,77	555,00	539,27	1330,00	345,95	1109,00	483,53	952,00	949,79
Media	170,20	298,33	517,80	283,36	1268,40	146,36	906,00	219,47	906,20	641,30
Desv. típica	10,57	109,00	27,67	229,36	57,09	114,39	121,24	198,43	50,63	342,17

GRAPH 05		GRAPH 06		GRAPH 07		GRAPH 11		GRAPH 12	
Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo
194,00	294,26	479,00	281,25	553,00	415,00	253,00	462,05	1166,00	760,65
182,00	133,46	540,00	624,14	540,00	315,65	1114,00	99,78	1148,00	109,46
165,00	286,00	581,00	80,44	525,00	411,25	1169,00	467,40	1200,00	56,78
173,00	12,76	511,00	179,58	562,00	748,00	1116,00	610,29	1219,00	299,58
159,00	56,74	441,00	189,74	509,00	154,76	1230,00	59,00	1234,00	656,76
159,00	12,76	441,00	80,44	509,00	154,76	253,00	59,00	1148,00	56,78
194,00	294,26	581,00	624,14	562,00	748,00	1230,00	610,29	1234,00	760,65
174,60	156,64	510,40	271,03	537,80	408,93	976,40	339,70	1193,40	376,65
13,87	129,32	54,00	209,81	21,30	216,98	407,17	245,38	35,91	318,42

Resultados AGG BLX 3

AGG BLX 3	CELAR 06		CELAR 07		CELAR 08		CELAR 09		CELAR 10	
	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo
Ejecución 1	121,00	130,62	386,00	290,18	1316,00	104,35	842,00	44,21	864,00	122,09
Ejecución 2	131,00	291,65	436,00	280,03	1318,00	620,38	1055,00	446,74	979,00	598,71
Ejecución 3	180,00	16,33	581,00	526,45	1288,00	843,97	942,00	176,59	945,00	131,11
Ejecución 4	163,00	344,55	462,00	310,82	1315,00	788,81	866,00	933,38	767,00	716,16
Ejecución 5	141,00	259,54	472,00	389,08	1256,00	487,83	831,00	521,30	976,00	281,11
Mejor	121,00	16,33	386,00	280,03	1256,00	104,35	831,00	44,21	767,00	122,09
Peor	180,00	344,55	581,00	526,45	1318,00	843,97	1055,00	933,38	979,00	716,16
Media	147,20	208,54	467,40	359,31	1298,60	569,07	907,20	424,44	906,20	369,83
Desv. típica	24,05	133,25	71,71	102,76	26,81	295,40	93,29	344,57	90,60	273,21

GRAPH 05		GRAPH 06		GRAPH 07		GRAPH 11		GRAPH 12	
Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo
151,00	216,02	464,00	365,00	424,00	33,81	1067,00	419,55	1203,00	526,26
197,00	220,51	419,00	400,82	420,00	294,40	1093,00	99,85	1274,00	644,09
158,00	132,36	507,00	260,48	470,00	364,41	1096,00	127,51	1175,00	692,05
134,00	164,35	475,00	253,14	511,00	523,47	1102,00	450,49	1209,00	355,30
202,00	108,13	570,00	152,51	500,00	258,79	1103,00	405,94	1267,00	841,70
134,00	108,13	419,00	152,51	420,00	33,81	1067,00	99,85	1175,00	355,30
202,00	220,51	570,00	400,82	511,00	523,47	1103,00	450,49	1274,00	841,70
168,40	168,27	487,00	286,39	465,00	294,98	1092,20	300,67	1225,60	611,88
29,75	49,83	56,09	98,71	42,05	177,87	14,69	171,74	43,02	182,65

Descripción de los resultados

Como hemos podido observar la obtención de una buena población con sus individuos generados por el nuevo Greedy nos ayuda bastante a obtener unos tiempos bastante buenos en los tres algoritmos.

Aunque como cabe esperar a unas pequeñas diferencias entre ellos tanto en tiempos como en penalizaciones, obviamente el AGG BLX 3 obtiene unas penalizaciones mejores, mucho menores que AGG BLX 2 y AGG BLX 1 en algunos casos.

Podemos decir que esta diferencia en penalizaciones se debe a que realizamos la búsqueda local con 200 cada 10 generaciones sobre solo el 10% de los mejores individuos de la población (2 individuos), con esto conseguimos unos individuos muy buenos que luego usaremos para cruzar y mutar y obtener mejoras entre ellos. Sin embargo, los dos otros algoritmos también realizan búsqueda local sobre un 10 % de individuos aleatorios y sobre toda la población.

El algoritmo AGG BLX 2 realiza la búsqueda local sobre 2 individuos que son aleatorios por tanto puede escoger individuos que son los peores, esto nos ayuda a que podamos mejorar la solución global de la población pero a la hora de escoger el mejor no ayuda mucho.

El algoritmo AGG BLX 1, realiza cada 10 generaciones una búsqueda local a todos los individuos de la población, esto es un buen enfoque pero, al realizar muchas comprobaciones, termina antes de ejecutar y no le da tiempo a encontrar una solución que sea mejor que la de los dos otros algoritmos.

En cuanto a los tiempos, como ya hemos comentado, los tiempos dependen del número de comprobaciones que realicen cada algoritmo, el algoritmo AGG BLX 1 es el que más comprobaciones realiza y por tanto el que antes finaliza. Los otros dos algoritmos tienen un número de comprobaciones iguales, dependería de sus mutaciones y si caen en una reinicialización en la que deben de generar de nuevo la población desde cero.

	CELAR 06		CELAR 07		CELAR 08		CELAR 09		CELAR 10	
	Z	Tiempo	*Z*	Tiempo	*Z*	Tiempo	*Z*	Tiempo	*Z*	Tiempo
AGG BLX 1 (med)	183,60	110,62	633,60	136,42	1560,20	251,27	1053,40	246,67	1014,80	183,13
AGG BLX 1 (desv)	22,92	245,63	38,90	7,45	135,95	4,19	97,29	99,52	98,27	4,87
AGG BLX 2 (med)	170,20	298,33	517,80	283,36	1268,40	146,36	906,00	219,47	906,20	641,30
AGG BLX 2 (desv)	10,57	109,00	27,67	229,36	57,09	114,39	121,24	198,43	50,63	342,17
AGE BLX 3 (med)	147,20	208,54	467,40	359,31	1298,60	569,07	907,20	424,44	906,20	369,83
AGE BLX 3 (desv)	24,05	133,25	71,71	102,76	26,81	295,40	93,29	344,57	90,60	273,21
Greedy (med)	216,00	0,00	686,40	0,00	1499,80	0,00	1264,20	0,00	1273,20	0,00
Greedy (desv)	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
248,20	33,64	569,20	89,07	666,60	213,66	1285,20	172,51	1400,40	181,80
23,83	73,79	31,92	43,74	60,50	242,21	65,72	3,28	60,20	3,29
174,60	156,64	510,40	271,03	537,80	408,93	976,40	339,70	1193,40	376,65
13,87	129,32	54,00	209,81	21,30	216,98	407,17	245,38	35,91	318,42
168,40	168,27	487,00	286,39	465,00	294,98	1092,20	300,67	1225,60	611,88
29,75	49,83	56,09	98,71	42,05	177,87	14,69	171,74	43,02	182,65
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

9. Referencias bibliográficas

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

Sem04-Problemas-Hibridaciones-MHs-17-18.pdf