UNIVERSIDAD DE JAÉN
ESCUELA POLITÉCNICA SUPERIOR DE JAÉN

Práctica 1:
Metaheurísticas basadas en trayectorias para el Problema de Asignación de Frecuencias con Mínimas Interferencias (Mi-Fap)

Curso 2017-2018
Grado de Ingeniería Informática

Algoritmos considerados:
Algoritmo Greedy
Búsqueda Local
Búsqueda Tabú
Búsqueda GRASP

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

Los algoritmos de esta práctica tienen en común las siguientes componentes:

• Esquema de representación: Se seguirá la representación entera basada en un vector X de tamaño n (transmisores).

• Función objetivo: Se persigue minimizar las interferencias.

• Generación de la solución inicial: La solución inicial se generará usando el algoritmo simple.

• Esquema de generación de vecinos: Se cambia el valor de frecuencia incluido en un transmisor por otra de las frecuencias disponibles.

• Criterio de aceptación: Se considera una mejora cuando se reduce el valor global de la función objetivo.

• Criterio de parada: Se detendrá la ejecución del algoritmo bien cuando no se encuentre mejora en todo el entorno (BL) o bien cuando se hayan evaluado 10000 soluciones distintas. En los algoritmos BT y GRASP, el criterio de parada será alcanzar un número máximo de iteraciones (es decir, número de soluciones generadas y, por tanto, evaluadas mediante la función objetivo) de 10000 soluciones. A continuación veremos las peculiaridades de cada algoritmo.

Búsqueda local

En este caso el algoritmo de búsqueda local seguirá el método de el 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)

solAux ← sol
penalización ← Comprobar
Bucle i ← 0 hasta 10000 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

Pseudocódigo de las operaciones relevantes (factorización)

Procedimiento ComprobarFact (var: vector, ctr: vector, ctr2: matriz, dom: matriz, sol: vector, FrecInicial, transmisor, valor)

```
devolver ← valor
Bucle i ← 0 hasta tamaño de ctr2 hacer
        trans1 ← ctr2(i(trx1))
        trans2 ← ctr2(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
        Si trx1 es igual a transmisor
                a ← dom(trx1(FrecInicial))
        Fin_Si
        Sino
                b ← dom(trx2(FrecInicial))
        Fin_Sino
        c ← a - b
        c ← abs(c)
        Si c mayor que diferencia de frecuencias de la restricción
                devolver ← devolver + penalización de la restricción
        Fin_Si
```

Búsqueda Tabú

La búsqueda tabú es un algoritmo que se basa en el uso de una lista de movimientos considerados “inválidos”, los cuales permitirán la selección o descarte de determinados vecinos del entorno de soluciones. De entre estos vecinos se elegirá al mejor de 20 mediante la selección del Primer Mejor, sin importar de momento si mejora o no al mejor valor de la función objetivo Z. El tamaño de partida de la lista tabú será NTrx/6, donde NTrx es el número de transmisores de la instancia estudiada.

Se incluirán en la lista tabú los movimientos (TRX, FrecuenciaTRX) donde TRX es el Transmisor en evaluación actualmente y FrecuenciaTRX es la frecuencia obtenida de la solución actual. No se permiten estos cambios en la lista tabú, salvo

que cumplan el Criterio de aspiración. Se seguirá aplicando la Factorización de igual forma que el caso de la BLocal.

Pseudocódigo del método de búsqueda

Procedimiento BusquedaTabu (var:vector, ctr: vector, ctr2: matriz, dom: matriz, sol: vector)

```
solTotal ← sol
solAux ← sol
penalización ← Comprobar
penalizaciónTotal ← penalización
matrix(tamaño var)
Bucle i ← 0 hasta tamaño solAux hacer
        insertar matriz
Fin_Bucle
listaTabu(tamaño var/6)
reinicio ← 0
k ← 0
Bucle i ← 0 hasta 10000 hacer
        Si reinicio igual a 2000
                reinicializar
                reinicio ← 0
        Fin_Si
        Buscamos un transmisor aleatorio
        Guardamos una frecuencia inicial (FreqInicial)
        Elegimos el sentido de frecuencias
        pen ← infinito
        penAux ← infinito
        Bucle j ← 0 hasta 20 hacer
                Comprobamos de forma factorial la solución de los nuevos vecinos
                (como en el búsqueda local)
                Si esTabu
                        Si penAux es menor que penalizacionTotal
                                pen ← penAux
                                frecuencia ← pos
                                reinicio ← 0
                        Fin_Si
                Fin_Si
                Sino Si penAux menor que pen
```

```
                    pen ← penAux
                    frecuencia ← pos
                    reinicio ← 0
            Fin_Sino
            Incrementamos las iteraciones y el reinicio
        Fin_Bucle
        sol(transmisor) ← FrecBest
        penalizaciçon ← pen
        insertarTabu
        insertarMatrix
        Si penalizacionTotal es mayor que pen
            solTotal(transmisor) ← solAux(transmisor)
            penalizacionTotal ← pen
        Fin_Si
        Incrementamos reinicio
        sol ← solTotal
```

Pseudocódigo de las operaciones relevantes (reinicialización)

```
Procedimiento reinicializar (sol: vector, matrix: matriz)

Bucle i ← 0 hasta tamaño matriz hacer
        contMax ← infinito_minimo
        frec ← 0
        Bucle j ← 0 hasta tamaño matriz hacer
            Si matrix(i(cont)) es mayor que contMax
                contMax ← matrix(i(j(cont))
                frec ← matrix(i(j(frec))
            Fin_Si
        Fin_bucle
        sol(i) ← freq
Fin_Bucle
```

Búsqueda GRASP

El algoritmo de búsqueda GRASP consiste en una solución inicial Greedy aleatorizada o LRC, y en una aplicación de un algoritmo de búsqueda local.

La LRC consiste en un método de relleno de las frecuencias elegidas aleatoriamente dentro de nuestro vector solución, este proceso se aplicará k veces (en nuestro caso, 10 veces), el resto de posiciones se completan seleccionando aquellas las que implican un menor coste a la solución. Crearemos un vector de costes, cuyo contenido se obtiene calculando el coste de supondrá incorporar cada frecuencia que se añade a la solución que se está construyendo. La frecuencia que finalmente se incorpora a la solución parcial se selecciona aleatoriamente de la LRC mediante sesgo y probabilidad de distribuciones. Se añade el valor, y se repite hasta completar el vector solución.

Finalmente, se aplicará una búsqueda local a cada solución obtenida, y se devolverá la mejor solución encontrada en el búsqueda local.

Pseudocódigo del método de búsqueda

Procedimiento Grasp(var: vector, ctr: vector, ctr2: matriz, dom: vector, sol: vector)

```
solTotal (tamaño var)
aleatorios ← 10
insertados ← 0
Bucle i ← 0 hasta tamaño solTotal hacer
        tam ← tamaño solTotal
        Si tam - aleatorios igual a insertados
                InsertarBusquedaLocal
        Fin_Si
        solAux (tamaño var)
        solAux ← solTotal
        insertarAleatorios
        insertarBusquedaLocal
        pos, costes, a
        Bucle i ← 0 hasta tamaño solAux hacer
                pos ← i
                a(transmisor) ← i
                Si (solAux(i(elegido))
                        a(coste) ← infinito
                Fin_Si
```

```
            Sino
                a(coste) ← ComprobarGrasp
            Fin_Sino
            a(pos) ← 0
            a(sesgo) ← 0
            insertar costes (a)
        Fin_Bucle
        Ordenar(costes)
        coger aleatorio y transmisor
        totalSum ← 0
        Bucle i ← 0 hasta tamaño costes hacer
            totalSum ← totalSum + costes(i(sesgo))
                Si totalSum mayor o igual que aleatorio
                    transmisor ← costes(i(transmisor))
                Fin_Si
        Fin_Bucle
        Marcamos la frecuencia de solTotal igual que la de solAux y colocado y
        elegido a verdadero
        Bucle i ← 0 hasta tamaño solTotal hacer
            insertar frecuencia de solTotal en sol
        Fin_Bucle
Fin_Bucle
```

Pseudocódigo de las operaciones relevantes (LRC)

```
Procedimiento Ordenar(costes: vector)

        sort (costes)
        coste ← infinito
        pos ← 1
        Bucle i ← 0 hasta tamaño costes hacer
            Si costes(i(coste)) es igual a coste
                coste ← costes(i(coste))
                costes(i(pos)) ← pos
            Fin_Si
            Sino
                coste ← costes(i(coste))
                pos ← i + 1
                costes(i(pos)) ← pos
            Fin_sino
```

```
        a ← costes(i(pos))
        b ← costes(i(pos))
        Si b es distinto de infinito
            totalSum ← totalSum + 1/a
        Fin_Si
Fin_Bucle

Bucle i ← 0 hasta tamaño costes hacer
        pos ← costes(i(pos))
        b ← costes(i(pos))
        Si b es distinto de infinito
            costes(i(sesgo)) ← (1 / pos) / (totalSum)
        Fin_Si
Fin_Bucle
```

4. Descripción del algoritmo de comparación Greedy

Greedy

El algoritmo con el que inicialmente compararemos nuestras soluciones, es el algoritmo Greedy. Usamos un modelo de Greedy estándar, en el que el vector solución de frecuencias, se rellena de forma aleatoria. Este se ejecutará 5 veces usando la función de C++, srand() y una semilla preestablecida o introducida por teclado.

Pseudocódigo del método de búsqueda

```
Sol ← 0
Bucle i ← 0 hasta nº transmisores hacer
        sol ← insertar frecuencia aleatoria
Fin_Bucle
```

5. 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 Visual Studio 2017, usando el compilador visual C++. El sistema operativo usado ha sido Windows 8.1.

Manual de usuario

Disponemos de un ejecutable el cual, al activarse, nos dará opción a seleccionar entre escoger una serie de semillas de nuestra elección, metiéndolas desde teclado, o bien eligiendo 5 semillas ya pre-establecidas mediante permutaciones de nuestros DNI, para lo cual cuando se nos pregunte por la primera semilla en ejecución, introduciremos 0 desde teclado.

A continuación se procederá a ejecutar para cada una de las instancias dadas para la práctica, todos y cada uno de los 4 algoritmos de búsqueda, obteniendo sus resultados en costes, y el tiempo de ejecución obtenido en cada algoritmo.

Tras la visualización de la ejecución de cada semilla, habrá que presionar 'Enter' para proceder a la ejecución de la siguiente semilla, así hasta agotar el número de semillas introducidas, o por defecto, lo cual dará paso a la finalización de nuestro programa.

6. Experimentos y 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 muy rápidas (Greedy y GRASP(LRC)).

2) Heurísticas de búsqueda-compleja: Son mucho más costosas en tiempo y recursos (Búsqueda Local, Búsqueda Tabú, GRASP(Búsqueda Local)).

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 obtenidos Greedy

Greedy	CELAR 06		CELAR 07		CELAR 08		CELAR 09		CELAR 10	
	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo
Ejecución 1	1908,00	0,00	4034,00	0,00	8179,00	0,00	5615,00	0,00	5738,00	0,00
Ejecución 2	2020,00	0,00	4244,00	0,00	8181,00	0,00	5753,00	0,00	5731,00	0,00
Ejecución 3	1891,00	0,00	4093,00	0,00	8088,00	0,00	5791,00	0,00	5858,00	0,00
Ejecución 4	1900,00	0,00	4052,00	0,00	7953,00	0,00	5804,00	0,00	5839,00	0,00
Ejecución 5	1890,00	0,00	4099,00	0,00	8347,00	0,00	5875,00	0,00	5542,00	0,00
Mejor	1890,00	0,00	4034,00	0,00	7953,00	0,00	5615,00	0,00	5542,00	0,00
Peor	2020,00	0,00	4244,00	0,00	8347,00	0,00	5875,00	0,00	5858,00	0,00
Media	1921,80	0,00	4104,40	0,00	8149,60	0,00	5767,60	0,00	5741,60	0,00
Desv. típica	55,38	0,00	82,69	0,00	144,28	0,00	96,06	0,00	125,50	0,00

GRAPH 05		GRAPH 06		GRAPH 07		GRAPH 11		GRAPH 12	
Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo
1596,00	0,00	3062,00	0,00	3184,00	0,00	5395,00	0,00	5432,00	0,00
1687,00	0,00	3234,00	0,00	3151,00	0,00	5332,00	0,00	5596,00	0,00
1555,00	0,00	3130,00	0,00	3057,00	0,00	5428,00	0,00	5568,00	0,00
1627,00	0,00	3083,00	0,00	3001,00	0,00	5386,00	0,00	5750,00	0,00
1620,00	0,00	3076,00	0,00	3201,00	0,00	5380,00	0,00	5632,00	0,00
1555,00	0,00	3062,00	0,00	3001,00	0,00	5332,00	0,00	5432,00	0,00
1687,00	0,00	3234,00	0,00	3201,00	0,00	5428,00	0,00	5750,00	0,00
1617,00	0,00	3117,00	0,00	3118,80	0,00	5384,20	0,00	5595,60	0,00
48,20	0,00	70,21	0,00	86,23	0,00	34,57	0,00	114,77	0,00

Análisis de resultados

El Greedy como ya hemos indicado anteriormente, es muy estático, sin posibilidad de mejora mientras este se ejecuta, esto se deja notar en la selección de frecuencias y en lo resultados de la función objetivo, los cuales son similares. Aún así es un algoritmo muy rápido.

Resultados obtenidos Búsqueda local

BL	CELAR 06		CELAR 07		CELAR 08		CELAR 09		CELAR 10	
	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo
Ejecución 1	1082,00	1,08	2964,00	0,76	6776,00	0,78	4253,00	0,89	4269,00	0,73
Ejecución 2	1145,00	0,85	3049,00	0,97	6867,00	0,78	4400,00	0,84	4420,00	0,79
Ejecución 3	1129,00	0,85	2863,00	0,95	6896,00	0,68	4730,00	0,77	4681,00	0,78
Ejecución 4	1143,00	0,87	2771,00	0,73	6517,00	0,78	4562,00	0,77	4556,00	0,73
Ejecución 5	916,00	0,79	2962,00	0,88	6911,00	0,66	4439,00	0,72	4342,00	0,68
Mejor	916,00	0,79	2771,00	0,73	6517,00	0,66	4253,00	0,72	4269,00	0,68
Peor	1145,00	1,08	3049,00	0,97	6911,00	0,78	4730,00	0,89	4681,00	0,79
Media	1083,00	0,89	2921,80	0,86	6793,40	0,74	4476,80	0,80	4453,60	0,74
Desv. típica	96,76	0,11	106,98	0,11	163,15	0,06	179,43	0,07	165,72	0,04

GRAPH 05		GRAPH 06		GRAPH 07		GRAPH 11		GRAPH 12	
Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo
785,00	0,77	2074,00	0,57	2041,00	689,00	3961,00	0,63	4161,00	0,69
779,00	0,79	2315,00	0,57	2031,00	0,64	4005,00	0,70	4328,00	0,79
837,00	0,63	2193,00	0,55	2131,00	0,57	4033,00	0,69	4373,00	0,80
847,00	0,61	2186,00	0,63	2231,00	0,65	4132,00	0,86	4544,00	0,92
821,00	0,56	2318,00	0,71	2246,00	0,64	4126,00	0,67	4402,00	0,83
779,00	0,56	2074,00	0,55	2031,00	0,57	3961,00	0,63	4161,00	0,69
847,00	0,79	2318,00	0,71	2246,00	689,00	4132,00	0,86	4544,00	0,92
813,80	0,67	2217,20	0,61	2136,00	138,30	4051,40	0,71	4361,60	0,81
30,55	0,10	102,21	0,07	101,49	307,85	75,37	0,09	138,20	0,08

Análisis de resultados

Como se puede observar en la tabla, el algoritmo de búsqueda local es muy rápido al uso de la búsqueda del Primer Mejor la cual resulta fácil de realizar. Este algoritmo si permite hacer una selección adaptativa de resultados por la selección de los distintos vecinos. Esta búsqueda de vecinos es buena, porque no se estanca aunque no mejore en el espacio de búsqueda actual. Esto se debe a que es un algoritmo muy simple, ya que las soluciones iniciales aleatorias son parecidas entre sí, y la forma de selección de posibles vecinos tienen cierta componente aleatoria.

El número de iteraciones se acaba agotando en este caso porque si se estanca en un mínimo local se prosigue con otro transmisor y seguirá tratando de evolucionar hasta llegar al final de la ejecución.

Resultados obtenidos Búsqueda Tabú

BT	CELAR 06		CELAR 07		CELAR 08		CELAR 09		CELAR 10	
	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo
Ejecución 1	686,00	1,17	2233,00	1,31	6097,00	1,86	4001,00	1,46	3786,00	1,48
Ejecución 2	881,00	1,15	2256,00	1,15	6593,00	1,80	3833,00	1,40	4001,00	1,41
Ejecución 3	913,00	1,08	2253,00	1,33	6217,00	1,85	3979,00	1,43	4134,00	1,45
Ejecución 4	858,00	1,14	2554,00	1,18	5987,00	1,68	3786,00	1,52	3975,00	1,44
Ejecución 5	895,00	1,04	2434,00	1,35	6369,00	1,37	3944,00	1,47	3680,00	1,21
Mejor	686,00	1,04	2233,00	1,15	5987,00	1,37	3786,00	1,40	3680,00	1,21
Peor	913,00	1,17	2554,00	1,35	6593,00	1,86	4001,00	1,52	4134,00	1,48
Media	846,60	1,12	2346,00	1,26	6252,60	1,71	3908,60	1,46	3915,20	1,40
Desv. típica	92,00	0,05	141,89	0,09	237,40	0,20	94,20	0,04	180,85	0,11

GRAPH 05		GRAPH 06		GRAPH 07		GRAPH 11		GRAPH 12	
Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo
683,00	1,01	1783,00	0,94	1815,00	1,12	3608,00	1,14	3795,00	1,31
765,00	0,98	1691,00	0,90	1650,00	1,06	3722,00	1,42	3813,00	1,48
720,00	0,95	1998,00	0,93	1941,00	0,92	3649,00	1,39	4002,00	1,45
670,00	0,96	2017,00	1,05	1815,00	1,13	3520,00	1,97	3941,00	1,45
679,00	0,81	1694,00	0,96	1887,00	1,05	3832,00	1,36	3953,00	1,53
670,00	0,81	1691,00	0,90	1650,00	0,92	3520,00	1,14	3795,00	1,31
765,00	1,01	2017,00	1,05	1941,00	1,13	3832,00	1,97	4002,00	1,53
703,40	0,94	1836,60	0,95	1821,60	1,05	3666,20	1,46	3900,80	1,44
39,36	0,08	160,47	0,06	109,62	0,08	117,96	0,31	91,49	0,08

Análisis de resultados

Este algoritmo es mejor que el búsqueda local con una ejecución ligeramente más lenta y con robustez alta, también agota las iteraciones, aunque sin embargo no llega a usar la matriz de reinicialización como se ha indicado anteriormente. Esto es porque al usar un búsqueda local, este suele mejorar rápidamente y de forma continua los resultados.Seguramente bajando las iteraciones para reinicializar, seguramente si tuviera efectividad. Además la intensificación sigue la misma generación de vecinos que la búsqueda local y sus resultados son igual de óptimos. La lista tabú es la que realmente hace la mejora entre el búsqueda local y el tabú, debido a la mayor facilidad para alcanzar óptimos locales.

Nuestra proposición de mejora pasaría por una búsqueda más completa por los vecindarios, y bajar las iteraciones del reinicio, así como aumentar el tamaño de la lista Tabú como ya se comentó anteriormente.

Resultados obtenidos Búsqueda GRASP

	CELAR 06		CELAR 07		CELAR 08		CELAR 09		CELAR 10	
	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo
Ejecución 1	1528,00	29,73	3185,00	167,00	5806,00	999,77	4005,00	520,56	3537,00	522,75
Ejecución 2	1577,00	40,01	2785,00	181,49	5613,00	846,91	3920,00	425,90	3782,00	456,10
Ejecución 3	1577,00	22,83	3268,00	178,01	5631,00	828,60	4189,00	434,06	3895,00	473,85
Ejecución 4	1572,00	16,60	3189,00	71,33	5424,00	388,47	4011,00	208,12	4005,00	206,85
Ejecución 5	1459,00	19,66	2947,00	82,19	5417,00	386,20	4024,00	204,37	4190,00	198,18
Mejor	1459,00	---	2785,00	---	5417,00	---	3920,00	---	3537,00	---
Peor	1577,00	---	3268,00	---	5806,00	---	4189,00	---	4190,00	---
Media	1542,60	25,76	3074,80	136,00	5578,20	689,99	4029,80	358,60	3881,80	371,54
Desv. típica	51,07	9,34	201,76	54,48	162,52	284,17	97,98	143,95	244,53	156,25

	CELAR 06		CELAR 07		CELAR 08		CELAR 09		CELAR 10	
	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo
GRASP (med)	1542,60	25,76	3074,80	136,00	5578,20	689,99	4029,80	358,60	3881,80	371,54
GRASP (desv)	51,07	9,34	201,76	54,48	162,52	284,17	97,98	143,95	244,53	156,25

GRAPH 05		GRAPH 06		GRAPH 07		GRAPH 11		GRAPH 12	
Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo
1168,00	36,05	2423,00	144,91	2266,00	138,63	3865,00	387,21	3638,00	410,54
1255,00	29,71	2372,00	118,16	2346,00	113,89	3808,00	289,70	3794,00	242,45
1274,00	28,88	2220,00	120,37	2291,00	117,70	3861,00	297,11	3848,00	244,85
1250,00	13,94	2211,00	57,26	2385,00	53,50	3782,00	150,70	3830,00	179,78
1244,00	14,26	2417,00	53,41	2389,00	54,96	3897,00	153,47	3642,00	162,65
1168,00	---	2211,00	---	2266,00	---	3782,00	---	3638,00	---
1274,00	---	2423,00	---	2389,00	---	3897,00	---	3848,00	---
1238,20	24,57	2328,60	98,82	2335,40	95,74	3842,60	255,64	3750,40	248,05
40,82	9,95	105,16	41,09	55,30	39,05	46,55	102,03	102,65	97,98

GRAPH 05		GRAPH 06		GRAPH 07		GRAPH 11		GRAPH 12	
Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo
1238,20	24,57	2328,60	98,82	2335,40	95,74	3842,60	255,64	3750,40	248,05
40,82	9,95	105,16	41,09	55,30	39,05	46,55	102,03	102,65	97,98

Análisis de resultados

Este algoritmo es bastante lento debido a la forma de la obtención de la solución inicial y el cálculo de frecuencias óptimas mediante sesgos y distribuciones. Además el sistema Grasp tiene que volver a arrancar 25 veces en cada ejecución lo cual nos da solo 400 resultados para 10000 iteraciones, esto provoca que el resultado sea malo a pesar del coste tan elevado de recursos, estos resultados no mejoran en ningún caso a el tabú y muchas veces empeoran los del búsqueda local, por lo que para obtener mejores resultados deberíamos dejar mayor número de ejecuciones en cuyo caso a costa de mucho tiempo y recursos si sacaría soluciones mejores.

Por lo que si siguiéramos un método LRC diferente, más rápido, podríamos tener resultados mejores que local, y quizá que el tabú con un tiempo y coste en recursos aceptable.

7. Comparativa entre algoritmos

Por observación, podemos concretar, que en cuanto a resultados de tiempo y de calidad en la penalización, el búsqueda tabú resulta ser el mejor de los algoritmos testados para la realización de este problema, este podría mejorarse más si se le permitiera hacer una mayor intensificación en el entorno de búsqueda, sin embargo, y aunque la lista tabú es una limitación del propio movimiento exploratorio del algoritmo, podemos constatar respecto del búsqueda local, que gracias a esta lista, se permite no caer en territorios ya explorados, evitando caer en un bucle de el que no pudiésemos salir, no obstante, hemos comprobado que con algunas iteraciones más y quizá una variación al alza en el tamaño de la lista tabú, pudiéramos sacar un mayor rendimiento de este algoritmo.

La Solución Greedy, (junto a en determinados momentos la Grasp), da las peores soluciones para nuestra función objetivo, aunque eso sí, muy buenas en tiempo, ya que se obtienen casi de forma instantánea.

Sin embargo en costes, Greedy es tan malo, que la Búsqueda local mejora siempre sus resultados, sin caer demasiado en la desventaja del tiempo ya que obtiene sus soluciones por el método de el Primer mejor, el cual según hemos comprobado, suele sacar mejores resultados si se hace un recorrido en el sentido de las frecuencias bajas de los transmisores, debido a que estas presentaran luego menor riesgo de colisiones al evaluar las restricciones.

Finalmente GRASP presenta un coste elevadísimo en tiempo de ejecución, la culpable de esto, es la LRC, la cual realiza un volumen de operaciones altísimo simplemente para el cálculo de una única frecuencia por transmisor. Si bien todo esto es cierto, hemos podido comprobar, que el GRASP no realiza el mismo número de comprobaciones de la solución que el resto de algoritmos, por lo que si este pudiera comprobar tantas veces como comprueban los demás, seguramente diera los mejores resultados, aunque por tiempo de ejecución, estos fueran realmente inviables.

En términos de tiempos, como ya hemos indicado anteriormente el algoritmo más rápido es el Greedy con tiempos que rondan entre los 3 y los 9 milisegundos. Todo ello, debido a que utiliza un algoritmo muy sencillo y con poco coste computacional.

En cuanto al Búsqueda local, tiene un tiempo medio de ejecución promedio bueno, entre 0,1 y 1, debido a que se centra en óptimos locales rápidamente.

La búsqueda Tabú, ofrece tanto buen tiempo, como buenos resultados, debido al uso de la lista tabú y eso teniendo en cuenta que la reinicialización no es efectiva para las 2000 iteraciones especificadas en la práctica, ya que, experimentalmente hemos comprobado que para que el sistema de reinicio fuera realmente eficaz, debería de producirse cada aproximadamente 100-200 iteraciones sin mejorar.

Finalmente GRASP, es el más lento de todos y presenta un resultados mediocres similares a veces al Búsqueda local pero sin mejora alguna, y con un alto costo en recursos.

	CELAR 06		CELAR 07		CELAR 08		CELAR 09		CELAR 10	
	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo
BL (med)	1083,00	0,89	2921,80	0,86	6793,40	0,74	4476,80	0,80	4453,60	0,74
BL (desv)	96,76	0,11	106,98	0,11	163,15	0,06	179,43	0,07	165,72	0,04
BT (med)	846,60	1,12	2346,00	1,26	6252,60	1,71	3908,60	1,46	3915,20	1,40
BT (desv)	92,00	0,05	141,89	0,09	237,40	0,20	94,20	0,04	180,85	0,11
GRASP (med)	1495,60	19,08	3265,40	79,61	5711,60	375,19	3773,40	194,86	4010,20	192,22
GRASP (desv)	81,02	0,91	213,33	2,30	183,64	15,37	145,18	6,89	253,97	11,30
Greedy (med)	Desv. típica	0,00	4104,40	0,00	8149,60	0,00	5767,60	0,00	5741,60	0,00
Greedy (desv)	55,38	0,00	82,69	0,00	144,28	0,00	96,06	0,00	125,50	0,00

GRAPH 05		GRAPH 06		GRAPH 07		GRAPH 11		GRAPH 12	
Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo	Z	Tiempo
813,80	0,67	2217,20	0,61	2136,00	138,30	4051,40	0,71	4361,60	0,81
30,55	0,10	102,21	0,07	101,49	307,85	75,37	0,09	138,20	0,08
703,40	0,94	1836,60	0,95	1821,60	1,05	3666,20	1,46	3900,80	1,44
39,36	0,08	160,47	0,06	109,62	0,08	117,96	0,31	91,49	0,08
1311,80	13,79	2362,00	54,65	2467,40	52,85	3713,40	143,18	3862,80	154,75
33,89	0,85	78,95	2,61	64,07	3,33	125,96	2,82	113,28	3,55
1617,00	0,00	3117,00	0,00	3118,80	0,00	5384,20	0,00	5595,60	0,00
48,20	0,00	70,21	0,00	86,23	0,00	34,57	0,00	114,77	0,00

Teniendo en cuenta la relación Tiempo/Coste calculado, elegir el mejor es algo complicado, aunque claramente sería o bien el búsqueda Tabú o bien el búsqueda local. A nuestro parecer el Tabú es mejor que el local, sin embargo este se queda a veces muy cercano en resultados al local, aunque siempre algo mejores, además consume algo más de memoria, y por método experimental podemos decir que si el local dispone de más iteraciones, es capaz sin casi aumentar el tiempo, de bajar los costes a niveles muy aceptables. Sin embargo, entendemos que con alguna pequeña mejora parcial en el código del Tabú, este debería ser considerado como el mejor algoritmo, respecto a tiempos y costes. Respecto a la robustez el Grasp presenta desviaciones algo superiores a las del resto de algoritmos, que por lo general, se comportan con desviaciones normales para las instancias estudiadas.

8. Referencias bibliográficas

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

Sem02-Problemas-Tray-Simples-Multiples-MHs-17-18.pdf