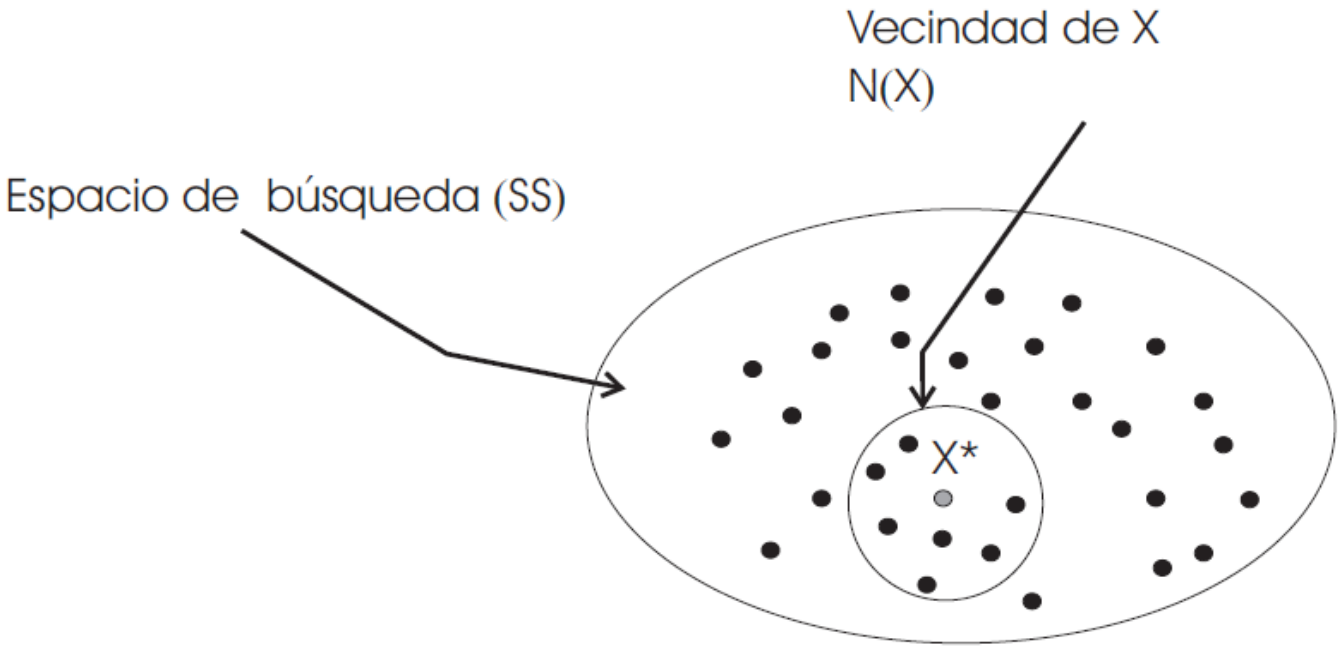
Metaheurísticas

Seminario 2.a. Problemas de optimización con técnicas basadas en trayectorias simples

1. Búsqueda de Entornos. Aspectos Avanzados
2. Ejemplo: Problema del Viajante de Comercio
3. Problemas de Optimización con Trayectorias Simples en el problema de la Asignación de Frecuencias con Mínimas Interferencias [Minimum Interference Frequency Assignment Problems (MI-FAP)]

Búsqueda de Entornos

- Dada un espacio de búsqueda SS para un problema concreto, **que viene definido por el esquema de representación de soluciones considerado**, y dada una solución x ∈ SS, el **entorno/vecindario** N(x) de esa solución es un subconjunto del espacio de soluciones que contiene aquellas soluciones que están "próximas" a la solución considerada:



Búsqueda de Entornos

- Dado un SS para un problema concreto, se puede definir una función de **distancia** dist(x,y) entre cualquier par de elementos x,y ∈ SS:

$$dist: SS \times SS \rightarrow R$$

- Se puede definir el entorno N(x) ⊆ SS de la solución x como:

$$N(x) = \{y \in SS : dist(x,y) \leq \varepsilon\}$$

Ejemplos: Distancia Euclídea en codificación real o distancia de Hamming en representaciones binarias

Búsqueda de Entornos

- En el entorno N(x) de una solución x se encuentran todas aquellas soluciones y ∈ N(x) accesibles desde x a través de una operación llamada **movimiento**/**operador de generación de vecino**

- En otras palabras, dada una solución x, cada solución de su entorno y ∈ N(x) puede alcanzarse (u obtenerse) directamente desde x mediante la aplicación del operador de generación de vecino

- Los algoritmos de búsqueda por entorno/trayectorias simples se basan en manejar una única solución en el SS y "moverse" a otras "soluciones vecinas"

Búsqueda de Entornos: Aspectos Avanzados – Exploración del Entorno

- En algunas ocasiones **puede no ser recomendable explorar el entorno al completo**

- Esto ocurre por ejemplo cuando:
  - su tamaño es muy grande (p.e., instancias muy grandes o entorno exponencial con respecto al tamaño del problema)
  - muchas soluciones vecinas presentan el mismo coste que la solución actual (espacio objetivo "plano")

- En dichos casos se recomienda una **exploración parcial del entorno**

Búsqueda de Entornos: Aspectos Avanzados – Exploración del Entorno

- Cuando se considere una búsqueda local del mejor en ese tipo de problemas (así como siempre que se use una búsqueda local del primer mejor) es interesante considerar **información específica del problema** (heurística) para:
  - Restringir el entorno de una solución para incluir únicamente movimientos prometedores llamados **listas de candidatos** (reducir el entorno)
  - Definir un **orden sistemático de exploración** que considere primero la aplicación de los movimientos más prometedores (explorar el entorno con preferencias)

Búsqueda de Entornos: Aspectos Avanzados - Factorización

- Las soluciones pertenecientes a un entorno son muy parecidas entre sí, los operadores de vecino provocan cambios “suaves”

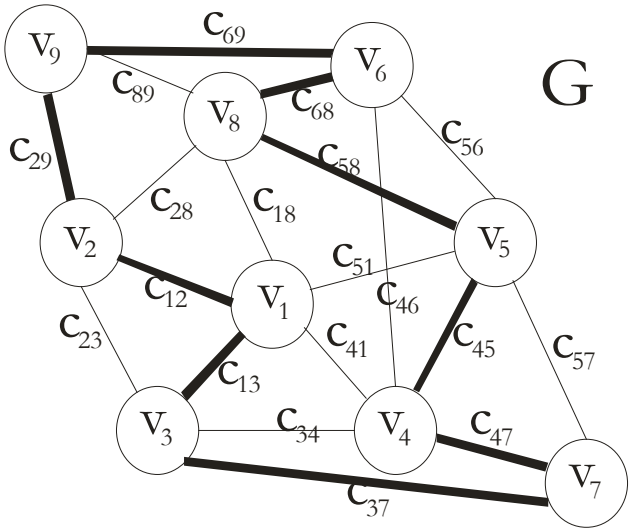
- Por esta razón, cuando se aplica un movimiento sobre una solución x para obtener una solución vecina x’ que es necesario evaluar se debe evitar el cálculo del valor de la función objetivo (coste) de la nueva solución haciendo uso de la fórmula original, es altamente ineficiente

- En su lugar, **siempre que sea posible** se aplica un mecanismo denominado **factorización** para calcular el coste de x’ a partir del de x considerando sólo los cambios realizados por el operador de vecino

Ejemplo: Problema del Viajante de Comercio

- **Problema del Viajante de comercio, TSP**:

  Un viajante de comercio ha de visitar n ciudades, comenzando y finalizando en su propia ciudad. Conociendo el coste de ir de cada ciudad a otra, determinar el recorrido de coste mínimo



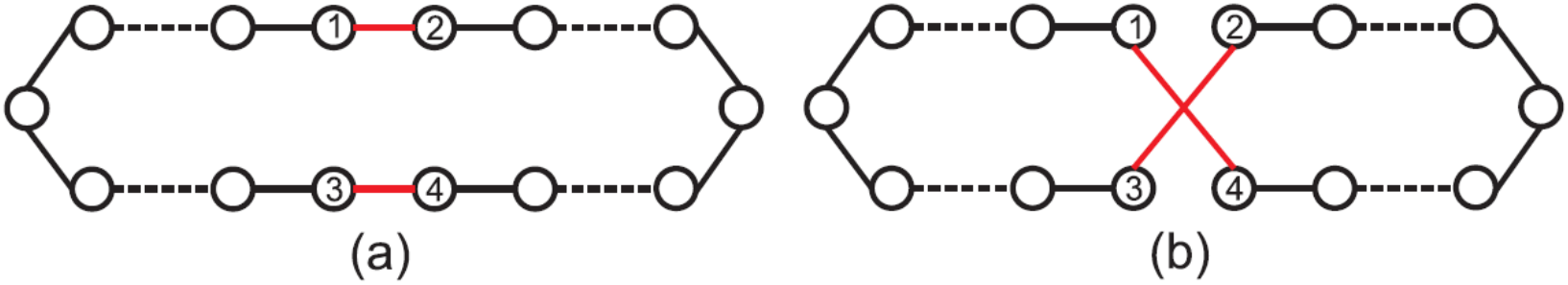
$$TSP = \left\{ \begin{array}{ll} min: & c_{\pi(1)\pi(n)} + \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} \end{array} \right.$$

TSP: Representación y Operador de Vecino

- **Representación**: Problema de ordenación: una permutación $\pi=[\pi(1), \ldots, \pi(n)]$. Permite verificar las restricciones

- **Operador de vecino 2-opt y su entorno**: El entorno de una solución π está formado por las soluciones accesibles desde ella a través de un movimiento 2-opt

  2-opt: Dada una solución (circuito cerrado) se consideran dos arcos no contiguos (a), cada uno conectando dos ciudades diferentes, se eliminan y se intercambian las conexiones (b)



TSP: Operador de Vecino 2-opt

- En la práctica, usando representación de permutación $\pi=[\pi(1), \ldots, \pi(n)]$, el operador de vecino 2-opt se aplica escogiendo dos posiciones no contiguas *i*,*j* e intercambiando sus contenidos:

$$\pi=[\pi(1), \ldots, \pi(i), \ldots, \pi(j), \ldots, \pi(n)]$$

$$\pi'=[\pi(1), \ldots, \pi(j), \ldots, \pi(i), \ldots, \pi(n)]$$

Move(π,*i*,*j*) verifica las restricciones, si la solución original π es factible siempre genera una solución vecina π' factible

- En general, este operador de vecino se denomina **intercambio** y es uno de los más empleados en representaciones de orden. Su aplicación provoca que el tamaño del entorno sea:

$$|N(\pi)|=\frac{n\cdot(n-1)}{2}$$

TSP: Listas de Candidatos y Exploración del Entorno

- El uso de listas de candidatos es obligado cuando se manejan instancias grandes del TSP
- Se ha demostrado que en muchos casos pueden obtenerse soluciones óptimas o casi óptimas limitando la exploración del entorno a un número pequeño de arcos de menor coste para cada nodo
- Un procedimiento muy simple consiste en **crear una lista de candidatos definida por el grafo de vecinos más cercano**. Los movimientos permitidos para cada entorno están restringidos a un número limitado de los nodos más cercanos a cada nodo
- Sin embargo, hay instancias del TSP donde las mejores soluciones no verifican esta condición. Por ejemplo, aquellos casos en que los nodos están agrupados en distintos clusters en el plano
- Por tanto, puede ser necesario considerar mecanismos más eficientes basados en que no todos los nodos de la lista de candidatos estén definidos por la lista de arcos más cercanos

TSP: Factorización del Movimiento 2-Opt

- Sea f(π) el coste de la solución original
- Para generar π', el operador de vecino elimina los arcos:

$$\pi(i)\pi(i+1) \text{ y } \pi(j)\,\pi(j+1),$$

- y restablece el circuito con los arcos:

$$\pi(i)\pi(j+1) \text{ y } \pi(j)\,\pi(i+1)$$

- Por lo tanto, el coste del movimiento se puede factorizar como:

$$\Delta f(\pi) = c_{\pi(i)\pi(j+1)} + c_{\pi(i+1)\pi(j)} - (c_{\pi(i)\pi(i+1)} + c_{\pi(j)\pi(j+1)})$$

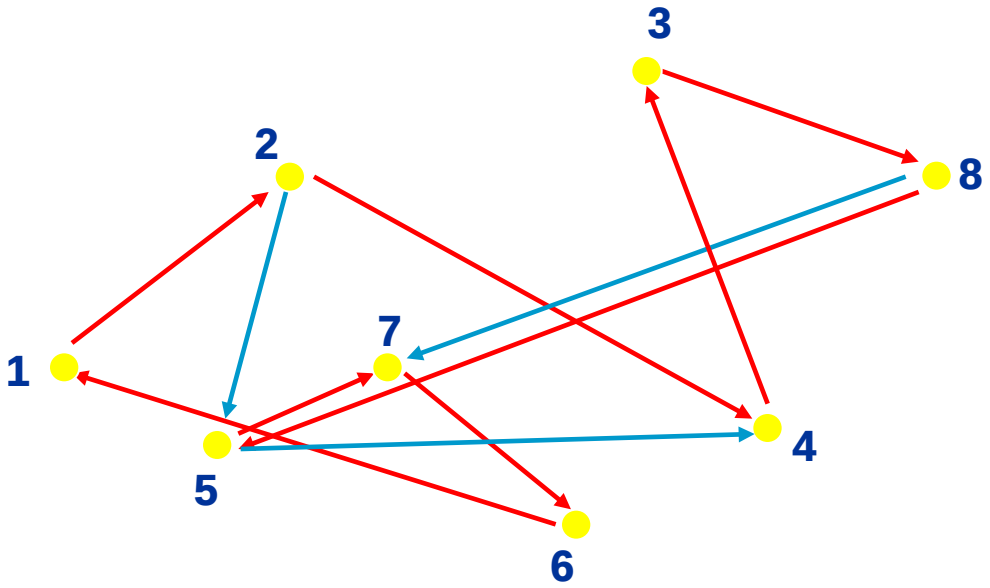
- y la variación en el coste de f(π) es:

$$f(\pi') = f(\pi) + \Delta f(\pi)$$

TSP: Otros operadores de vecino para representación de permutación

- Inserción: se escoge un elemento situado en una posición, se extrae de ella y se inserta en otra posición distinta:

(1 2 _ 4 3 8 5 7 6)
(1 2 5 4 3 8 7 6)



Tamaño del entorno:

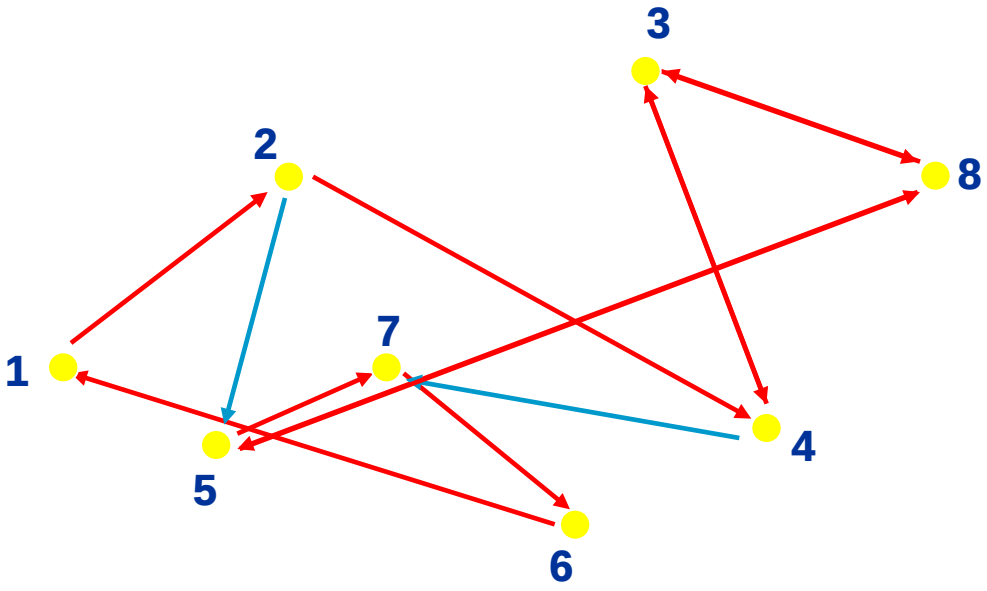
$$|N(\pi)| = n \cdot (n-1)$$

TSP: Otros operadores de vecino para representación de permutación

- Inversión: se escoge una sublista (determinando una posición inicial y otra final) y se invierten los elementos de la misma

(1 2 <u>4 3 8 5</u> 7 6)

(1 2 <u>5 8 3 4</u> 7 6)



Tamaño del entorno:

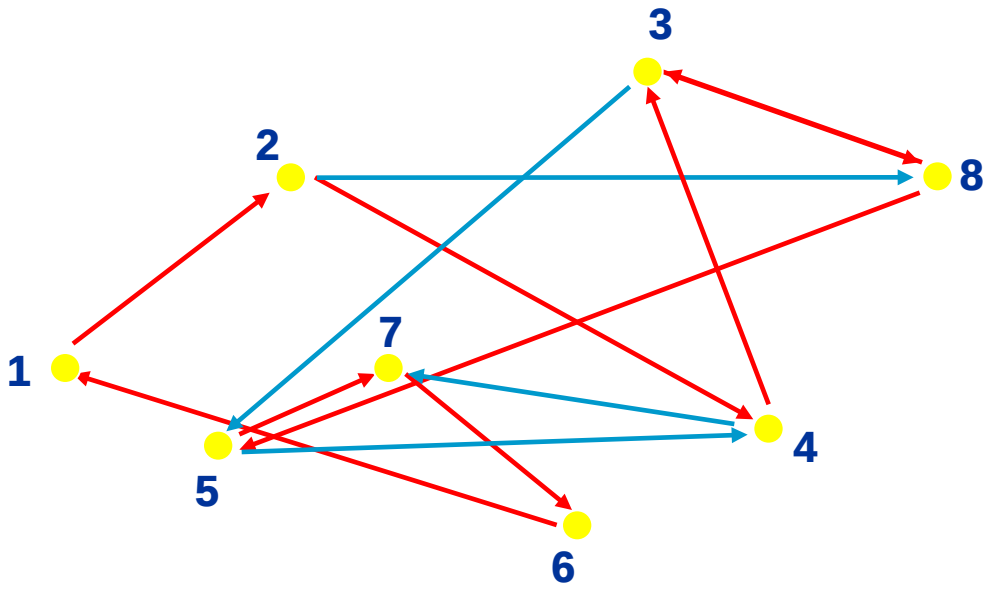
$$|N(\pi)| = \frac{n \cdot (n-1)}{2}$$

TSP: Otros operadores de vecino para representación de permutación

- Sublista aleatoria: se escoge una sublista y se “barajan” los elementos de la misma:

(1 2 4 3 8 5 7 6)

(1 2 8 3 5 4 7 6)



Ejemplos y Tipologías de Problemas: Representación de Permutación

- Algunas preguntas:

- ¿cuánto de distinta puede llegar a ser una solución vecina dependiendo del operador empleado?

- ¿cuál es el tamaño del entorno del operador de sublista aleatoria?

- ¿Provocan el mismo efecto todos los operadores en cualquier problema de optimización que considere una representación de orden?

  Por ejemplo, ordenación vs. asignación. ¿Es lo mismo generar un vecino mediante el operador de inserción en el TSP que en otros problemas?

El Problema del MI-FAP

- El **problema de la Asignación de Frecuencias con Mínimas Interferencias (MI-FAP)** consiste en asignar frecuencias a una serie de trasmisores (los cuales poseen un rango de frecuencias disponibles en las que transmitir). El objetivo es minimizar las interferencias entre las frecuencias.
- Problema de la Asignación de Frecuencias (FAP) es un problema clásico. Existen diferentes variantes del problema dependiendo del objetivo:
  - 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).

El Problema del MI-FAP

Definición del problema

- Sea T = $\{t_1, t_2, \ldots t_n\}$ un conjunto de n trasmisores (TRX) y sea $F_i = \{f_{i1}, f_{i2}, \ldots f_{ik}\}\} \subset N$ un conjunto de frecuencias válidas que pueden ser asignadas a los transmisores $t_i \in T$, i=1.. , n. Nótese que el cardinal de k de F_i no es necesariamente el mismo para todos los transmisores. Se define una matriz de interferencias entre las frecuencias MI.

- La función objetivo se puede definir de la siguiente forma:

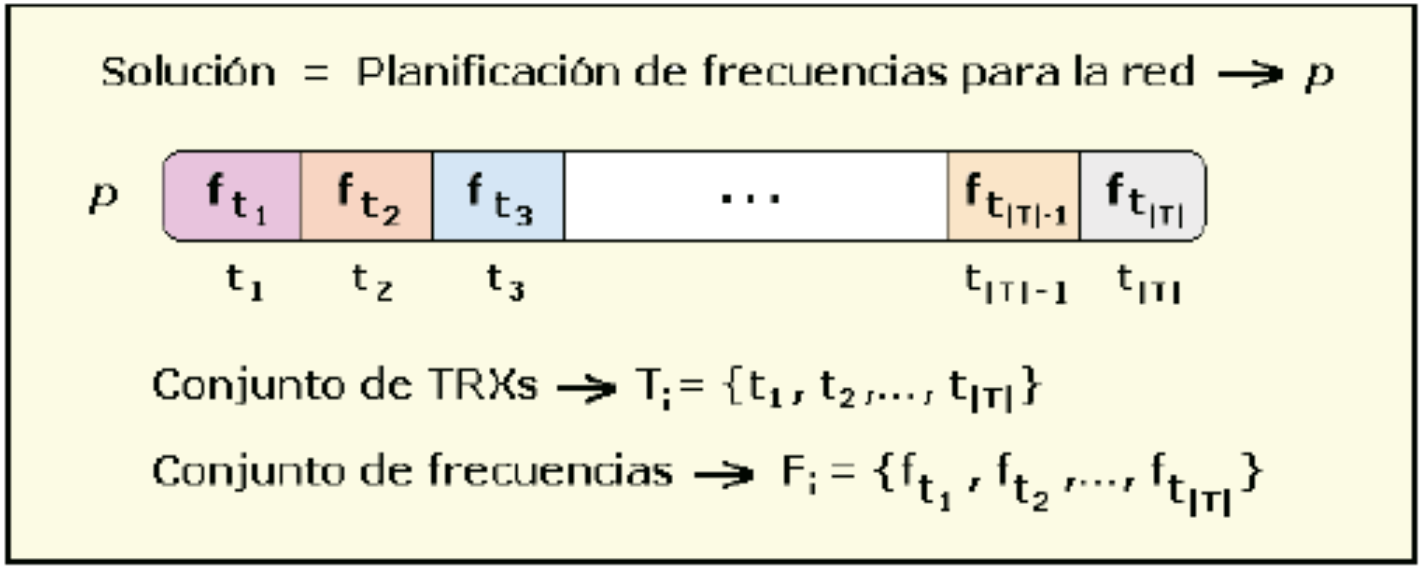
$$fitness = \sum_{i=0}^{n} \sum_{j=i+1}^{n} MI(TRXi, TRXj)$$

- Restricciones: Cada trasmisor solo puede usar una frecuencias de las disponibles para ese trasmisor.

El Problema del MI-FAP

Definición del problema

- Representación de una solución (vector X): Entera.
- No permutación pq las frecuencias se pueden repetir.



Ejemplo*: *con* 5 *TRX

(10,23,45,88,23)

Algoritmo Greedy para MI-FAP

- Para crear una solución inicial se va a utilizar un algoritmo Greedy simple. Consiste en seleccionar aleatoriamente una frecuencia de las disponibles para cada trasmisor.

- Los ficheros que se usaran en la práctica serán:
  - var.txt: la lista de trasmisores.
  - dom.txt: la lista de dominios de frecuencias.
  - ctr.txt: la lista de todas las restricciones. Las restricciones serán de la siguiente forma:
    - trx1 trx2 C > 100 4

      Indica la restricción del trasmisor 1 respecto al trasmisor 2. Se calcula como la diferencia de la frecuencia asignada al trasmisor 1 respecto a la frecuencia asignada al trasmisor 2. En caso de que esa diferencia sea mayor de 100 debo sumar 4 al calculo del coste.
    - No se usan las restricciones que que pongan símbolo D=

Algoritmo Greedy para MI-FAP

- Ejemplo con 4 trasmisores y 2 rangos de frecuencias.

Fichero dom.txt

rangoFrec	Listado de frecuencias
1	16 30 44 58 72 86 100 114 128 142 156 254 268 282 296 310 324 338 352 366 380 394 414 428 442 456 470 484 498 512 526 540 554 652 666 680 694 708 722 736 750 764 778 792
2	36 30 44 58 72 86 100 114 128 142 268 282 296 310 324 338 352 366 380 428 442 456 470 484 498 512 526 540 666 680 694 708 722 736 750 764 778

Fichero var.txt

TRX	rangoFrec
1	1
2	1
3	2
4	2

C es la diferencia de frecuencias No usar las que pongan símbolo D=

Fichero ctr.txt

TRX	Restricciones		
trx	trx	Frec comp	Interf
1	2	D = 238	0
1	3	C > 300	3
1	4	C > 157	2
2	3	C > 100	1
3	4	C > 185	4

Solución(frec.trx1,frec.trx2,frec.trx3,frec.trx4)

Codificación Solución -> $(16, 30, 268, 338)$

Restricción trx1 y trx3 = |16-268| < 300 -> 0
Restricción trx1 y trx4 = |16-338| > 157 -> 2
Restricción trx2 y trx3 = |30-268| > 100 -> 1
Restricción trx3 y trx4 = |268-338| < 185 -> 0

|16-268| y |16-338| y |30 -268| y |268-338|
Coste o Fitness= 0 + 2+ 1+ 0 = 3

MI-FAP: Búsqueda Local del Primer Mejor

Procedimiento Búsqueda Local del Primer Mejor

Inicio
GENERA(S_{act}); *// Generar solución usando Greedy*

Repetir

Repetir
$S' \leftarrow$ GENERA_VECINO(S_{act}); *// Usando operador vecino*
Hasta *(Objetivo(S') **mejor que** Objetivo(Mejor Vecino)) **O***
(se ha generado E(S_{act}) al completo)
$S_{anterior} \leftarrow S_{act}$;
*Si Objetivo(S') **mejor que** Objetivo(S_{act}) entonces*
$S_{act} \leftarrow S'$;
Hasta *(Objetivo(S') **peor o igual que** Objetivo($S_{anterior}$));*

DEVOLVER(S_{act});
Fin

MI-FAP: Búsqueda Local del Primer Mejor

- El procedimiento de búsqueda local se aplica después de la construcción de la solución inicial con greedy.
- Una solución vecina se obtiene a través del siguiente procedimiento. En primer lugar, se determina un **número al azar** para decidir *por cual de los* **trasmisores** (por ejemplo: TRX1) **comenzar la búsqueda**. A continuación se cambia la frecuencia asignada al trasmisor TRX1 por otras de las frecuencias disponibles para ese trasmisor. El vecino se obtiene cambiando la frecuencia por la situada a la derecha o de izquierda de lista de frecuencia (eligiendo el sentido al azar).

MI-FAP: Búsqueda Local del Primer Mejor

- En caso de ser mejor se reemplaza la solución por la actual y sino se siguen evaluando vecinos de esa solución (TRX1), es decir, cambiar por otra de las frecuencias disponibles para el trasmisor.
- Se evalúan todos los vecinos de ese trasmisor TRX1 si ninguna de las soluciones mejora, se continua el procedimiento con el resto de trasmisores en orden (el que este colocado en la siguiente posición de TRX1).
- Una vez que el procedimiento de mejora encuentra una solución mejor, la siguiente búsqueda de vecino será ejecutado en torno a esta solución mejor.
- La estrategia de búsqueda local es la del primer mejor.

MI-FAP: Factorización

- Sea f(π) el coste de la solución original.

$$f(\pi) = \sum_{i=0}^{n} \sum_{j=i+1}^{n} MI(TRXi, TRXj)$$

- Para generar π', el operador de vecino cambia la frecuencia asignada a un trasmisor. Para ello debo eliminar las interferencias de esa frecuencia **a** y añadir las interferencias de la frecuencia nueva asignada **b**.
- Por lo tanto, el coste del movimiento se puede factorizar como:

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

y la variación en el coste de f(π) es:

$$f(\pi') = f(\pi) + \Delta f(\pi)$$

MI-FAP: Búsqueda Tabú

Procedimiento de la búsqueda Tabú

Algorithm 9.4: Abstract Tabu Search algorithm - application specific aspects

```
1  Create initial solution s;
2  Initialize tabu list T;
3  while termination criterion not satisfied do
4      Determine neighborhood N of current solution s;
5      Choose best non-tabu solution s' from N;
6      Switch over to solution s' (current solution s is replaced by s');
7      Update tabu list T;
8      Update best found solution (if necessary);
9  end
10 return Best found solution;
```

MI-FAP: Búsqueda Tabú

- Solución inicial: Greedy
- Operador del vecino: Operador utilizado en la BL.
- Estrategia de selección de vecino: Examinar los vecinos y escoger el mejor de acuerdo a los criterios tabú. En cada iteración se generarán como *máximo 20 soluciones vecinos* (o menos si no hay tantos vecinos en la solución actual). Se seleccionará el mejor vecino de acuerdo a los criterios tabú.

- Lista tabú: Se almacenan la información del TRX cambiado y la nueva frecuencia asignada a ese trasmisor.
  - Por ejemplo, en la lista tabú se almacena (TRX,FrecNueva).

MI-FAP: Búsqueda Tabú

- Criterio de aspiración: Tener menor coste que la mejor solución obtenida hasta el momento. Un vecino que sea mejor que la mejor solución encontrada se considerado admisible, incluso aunque dicho vecino sea una solución tabú

- Tamaño de la lista Tabú: La longitud de la lista tabú depende del número de trasmisores de la solución.
  - El tamaño de la lista tabú será TRX/6. (probar experimentalmente)
  - Ejemplo: (10,23,45,65,66,88,23,45,65,66) con 10 TRX vale 2.

- Número máximo de iteraciones:10000

MI-FAP: Búsqueda Tabú

- Durante la aplicación de la BT se almacena la matriz de frecuencias de los diferentes cambios que se han producido durante la búsqueda. (Lista Tabú)
- Por ejemplo (TRX,FrecNueva, númeroapariciones).
- Reinicialización: En el caso de que la búsqueda se estanque (es decir, 2000 iteraciones sin mejora) se aplica una reinicialización partiendo de una solución en la que se intenta que las frecuencias asignadas a los TRX sean los más frecuentes (en la matriz de frecuencias).

- Recuerda que debes de almacenar durante todo el proceso la mejor solución obtenida para no perderla durante la búsqueda y reinicialización.

MI-FAP: Grasp

```
procedure grasp()
1     InputInstance();
2     for GRASP stopping criterion not satisfied →
3           ConstructGreedyRandomizedSolution(Solution);
4           LocalSearch(Solution);
5           UpdateSolution(Solution,BestSolutionFound);
6     rof;
7     return(BestSolutionFound)
end grasp;
```

```
procedure ConstructGreedyRandomizedSolution(Solution)
1     Solution = {};
2     for Solution construction not done →
3           MakeRCL(RCL);
4           s = SelectElementAtRandom(RCL);
5           Solution = Solution ∪ {s};
6           AdaptGreedyFunction(s);
7     rof;
end ConstructGreedyRandomizedSolution;
```

MI-FAP: GRASP

- Construcción Solución Greedy Aleatorizada: La solución se construirá utilizando una Lista Restringida de candidatos.

- Lista Restringida de candidatos(LRC):
  - La LRC se rellena con *k* frecuencias elegidas aleatoriamente, el resto de elementos hasta completar la solución se rellena con frecuencias elegidas seleccionando aquellas que añaden menos coste a la solución en construcción.
    - K igual 10 (probar experimentalmente)
  - Se crea un vector (**vectorcostes**), que se obtiene calculando el coste de supondrá incorporar cada frecuencia que se añade a la solución que se está construyendo.

MI-FAP: GRASP

- La frecuencia que finalmente se incorpora a la solución parcial se selecciona aleatoriamente de la LRC (considerando un sesgo).
  - El sesgo permite dar una probabilidad mayor de selección de los mejores candidatos.
  - Los elementos de la LRC se ordenan de acuerdo a los valores del **vectorcostes**. La probabilidad de seleccionar el elemento es:

$$\pi(r(e)) = \frac{sesgo(r(e))}{\sum_{e' \in RCL} sesgo(r(e'))},$$

  - donde r(e) es la posición del elemento e en la LRC. Existen varias alternativas para asignar sesgos a los elementos. En la práctica se utiliza el sesgo lineal: sesgo(r) = 1/r;

MI-FAP: GRASP

- Una vez que tenemos las probabilidades π, se obtiene un valor aleatorio. Este valor permite escoge al azar cualquiera de la frecuencias cuya probabilidad este por encima de ese valor aleatorio.

- Una vez que se incorpora la frecuencia seleccionada a la planificación, se actualiza la lista de candidatos y se recalculan los costes incrementales de las frecuencias (**vectorcostes**). Esto es lo que le da la metaheurística un carácter adaptativo.

- Este proceso se repite hasta que todos los trasmisores de la solución tienen asignada una frecuencia.

MI-FAP: GRASP

Ejemplo Grasp (Solución Greedy aleatorizada)

- Solución actual con 5 trasmisores (, , ,……,)

1) Escojo al azar 3 trasmisor (aunque en la práctica sería k=10) sin frecuencia asignada. Selecciono frecuencias al azar para esos trasmisores Ej: 1,2,3,..
2) Completo la solución usando el algoritmo greedy Ej: 1,2,3,3,5
3) Se obtiene **vectorcostes**, que es el coste de incorporar cada frecuencia a la solución (restricciones de ctr.txt)

   En el ejemplo de la página 21. TRX1 tiene un coste de 2, TRX2 coste de 1, TRX3 coste 1 y TRX4 coste 2.
4) Calculo la probabilidad con el sesgo lineal. El sesgo permite dar una probabilidad mayor de selección de los mejores candidatos.

MI-FAP: GRASP

Ejemplo Grasp (Solución Greedy aleatorizada)

TRX	coste	posición	π (*probabilidad*)
1	2	2	=0,5/2,28 -> 0,219
2	5	4	=0,25/2,28-> 0,1096
3	10	5	=0,2/2,28
4	1	1	=1/2,28 -> 0,438
5	4	3	=0,33/2,28

5) Se genera un valor aleatorio, que permite escoge al azar cualquiera de la frecuencias cuya probabilidad este por encima del valor aleatorio. Ejemplo, si sale 0,2 escojo al azar entre TRX1 o TRX4
6) Incluyo la frecuencia seleccionada en la solución actual Ej:(1,,,,)
7) Volver al punto 1, hasta que todos los trasmisores tengan frecuencias asignadas.
8) Devolver la solución

MI-FAP: GRASP

- Búsqueda local del primer mejor: Se detendrá la búsqueda en el momento que se hayan superado 400 iteraciones (en fase de la BL) o antes si después de examinar todo el vecindario de una solución no se han encontrado soluciones mejores. Implicara (seguramente) obtener varias soluciones mejores que la actual.
  - Se incrementa el número de iteraciones cada vez que se genera una solución candidata.

- Número máximo de iteraciones: será 10000 (en cada ejecución del algoritmo se generarán al menos 25 soluciones iniciales con greedy aleatorizado, es decir, 400*25 =10000).