Mínimum Order Frequency Assignment Problem For Antenna Planning

Felipe Cisternas Alvarez

14 de octubre de 2023

Resumen

En este informe se analizara el Minimum Order Frequency Assignment Problem (MO-FAP), problema que nace a partir de resolver el Frequency Assignment Problem (FAP), el cual dado una cantidad de antenas se busca asignar frecuencias para cada una de ellas, sujeto a una serie de restricciones. Existen variados métodos para poder resolver este problema, como lo pueden ser mediante programación lineal entera, enfoques heurísticos, entre otros.

1. Introducción

El Mínimum Order Frequency Assignment Problem, es un problema de asignación y optimización en el cual dado una cantidad de antenas se busca asignar frecuencias para cada una de ellas, el objetivo es minimizar la cantidad de frecuencias totales distintas asignadas a cada antena, cada antena tiene su propio dominio de frecuencias a las cuales puede tener acceso y las frecuencias asignadas deben estar a una determinada distancia mínima para no causar interferencia entre si. Este problema es bastante interesante de estudiar debido al gran impacto real que tiene en las telecomunicaciones que usamos a diario como lo puede ser la telefonía móvil, la televisión, la radio, incluso hasta en ámbitos tan avanzados y sensibles como lo son la comunicación Militar y Satélital. En la sección 2 se definirá el problema, en la sección 3 se verá el estado del arte, en la sección 4 se verá el modelo matemático para finalizar con la sección 5 con unas conclusiones.

2. Definición del Problema

El problema consiste en asignar el menor numero de frecuencias distintas dada una cantidad de antenas, cada antena tiene su rango de frecuencias y estas frecuencias deben estar a una distancia mínima para no causar interferencias entre si f[2].

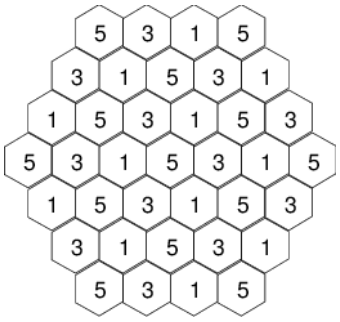

Figura 1: Rejillas hexagonales representando las distintas antenas y dentro su numero de frecuencia

Este problema tiene muchas similitudes con el problema de coloreo de grafos, en el cual se busca minimizar la cantidad de colores distintos utilizados para colorear el grafo, para el Mínimum Order Frequency Assignment Problem en vez de colores, se busca minimizar la cantidad de frecuencias distintas utilizadas por las antenas.

2.1. Parámetros

- V: Conjunto de antenas.
- F: Conjunto de frecuencias disponibles.
- F_v: Conjunto de frecuencias disponibles para la antena v.
- d_{vw}: Distancia mínima entre las frecuencias f y g asignadas a v y w, donde $|f - g| \leq d_{vw}$

2.2. Restricciones

- Una frecuencia se puede asignar **solamente** si esa frecuencia esta en uso
- Cada antena debe tener asignada **una** sola frecuencia
- Se asignara **al menos una** frecuencia entre dos antenas en donde no se cumpla la distancia mínima de interferencia

2.3. Variantes

Mínimum Order Frequency Assignment Problem nace del Frequency Assignment Problem, el cual es un problema mas general, dentro de esta rama de problemas en los que se encuentra MO-FAP podemos encontrar distintas variaciones [4] tales como

- F-FAP: Feasibility Frequency Assignment Problem
- Max-FAP: Maximum Service FAP
- MS-FAP: Minimum span FAP
- MI-FAP: Minimum interference FAP

3. Estado del Arte

El MO-FAP, nace del Frequency Assignment Problem, el cual fue introducido entre los años 1960 y 1970 [1] gracias al uso de las redes y dispositivos inalámbricos, este problema consiste en asignar frecuencias a una cantidad de antenas evitando interferencias, a medida que aumentan las frecuencias y las antenas el problema se vuelve muy difícil de resolver matemáticamente, debido a la gran combinatoria de posibilidades sujetas a las restricciones del problema. el Mínimum Order Frequency Assignment Problem además busca minimizar la cantidad de frecuencias totales utilizadas, se ha abordado el problemas desde distintos enfoques, cada uno con sus ventajas y desventajas, los cuales los veremos a continuación

3.1. Enfoque de Programación Lineal Entera (ILP)

Una de las formulaciones del MO-FAP es con programación lineal entera, para implementar ILP se utilizo el BCAT (Budapest Complexity Analysis Toolkit) framework, este framework facilita la implementación de algoritmos, se utilizo el COST259 benchmark que incluye data de diversas redes GSM 900 y GSM 1800, cada problema fue resuelto 10 veces y se pondero la media de sus iteraciones, en la tabla siguiente se pueden ver el numero de iteración en donde fue encontrada la solución para el MO-FAP para distintos valores de tiempo, cada columna representa una instancia distinta del MO-FAP, cabe destacar que donde hay un guión (-) significa que el problema no fue capaz de encontrar una solución con el limite de tiempo entregado.

Runtime	**Best solution found**					
half min.	24	23	13	11	–	30
1 min.	–	21	12	12	37	28
2 min.	–	22	–	11	34	29
5 min.	22	22	11	11	30	28
10 min.	22	22	12	12	-	28
quarter hour	–	22	14	12	27	27
half hour	–	25	12	12	32	29
1 hour	29	22	13	11	32	27
2 hour	24	–	–	11	29	29

Figura 2: Tabla con las mejores soluciones de MO-FAP para distintos valores de tiempo

Como se puede observar, hay casos donde en medio minuto se encuentra una solución y por mucho tiempo que se le de al algoritmo, como lo pueden ser dos horas, la solución a veces no cambia, incluso a medida que aumenta el tiempo hay veces donde el algoritmo no puede encontrar una solución, por eso es que la mejor opción es detener el algoritmo de ILP tan pronto como se haya completado el la mitad de un minuto.

3.2. Enfoques Heurísticos

Se pueden utilizar muchas Heurísticas para resolver el MO-FAP, nos centraremos en analizar las mas famosas y óptimas.

3.2.1. Local Search (LS)

Local Search o búsqueda local es un algoritmo que dada una solución inicial, va cambiando alguno de los parámetros, para evaluar la nueva solución y si es mejor, reemplazar la solución

anterior por la nueva como mejor solución, así iterativamente, si entre todas las soluciones vecinas no existe una mejor, LS termina. Este algoritmo permite obtener soluciones factibles en tiempos relativamente cortos

```
1: Input: current solution S
2: nextSectors ← {1, ..., numberOfSectors}
3: while (nextSectors != ∅) do
4:    currentSectors ← nextSectors
5:    nextSectors ← ∅
6:    while (currentSectors != ∅) do
7:       sec ← extract a random sector from currentSectors
8:       neighbour ← reassign frequencies of S in sector sec
9:       if (neighbour improves S) then
10:         S ← neighbour
11:         nextSectors += sectors interfered by sec
12:         nextSectors += sectors that interfere sec
13:      end if
14:   end while
15: end while
16: return S
```

Figura 3: Pseudocódigo para Local Search

La siguiente tabla muestra el resultado de una Local Search versus un Random Choice en un problema real de aplicar FAP en las ciudades de Denver y Seattle

	Random $\bar{x}_{\pm\sigma_n}$	Local Search $\bar{x}_{\pm\sigma_n}$
Seattle	$55\ 204.11_{\pm 1815.99}$	$3\ 692.66_{\pm 405.19}$
Denver	$115\ 577\ 436.48_{\pm 3\ 902\ 668.53}$	$105\ 155.60_{\pm 2\ 077.20}$

Figura 4: Media y desviación estándar de 30 soluciones de la función objetivo para la ciudad de Denver con 2612 antenas y 18 frecuencias y la ciudad de Seattle con 970 antenas y 15 frecuencias

Esta Heurística como muchas otras, se puede combinar para resolver el problema de manera mas óptima, un ejemplo de esto es **Local Search Heuristic Restart**, que es un Local Search que se le añaden pequeñas perturbaciones para evitar caer en mínimos locales

```
1: initialize(probability matrix F)
2: S* ← S ← generateFrom(F)
3: while not time-limit do
4:    S ← localSearch(S)
5:    S* ← best(S,S*)
6:    F ← update(F,S*)
7:    S ← generateFrom(F)
8: end while
```

Figura 5: Pseudocódigo de un Local Search con Heuristics Restarts (LSHR)

3.2.2. Evolutionary Algorithm (EA)

Evolutionary Algorithm o Algoritmo Evolutivo es un algoritmo que se basa en el principio de la evolución biológica, en estos algoritmos se mantiene un conjunto de entidades que representan

posibles soluciones, las cuales se mezclan, y compiten entre sí, de tal manera que las más aptas son capaces de prevalecer a lo largo del tiempo, evolucionando hacia mejores soluciones cada vez.

```
1: initialize(population)
2: population ← localSearch(population)
3: while not time-limit do
4:    offspring ← mutation(population)
5:    offspring ← localSearch(offspring)
6:    for i = 0 to populationSize do
7:       if population(i) is blocked for softBloq generations
         then
8:          population(i) ← offspring(i)
9:       else
10:         population(i) ← best(population(i), offSpring(i))
11:      end if
12:   end for
13:   if population is blocked for hardBloq generations then
14:      if |population| < maxPopSize then
15:         increase population size attaching a new individual
16:      end if
17:   end if
18: end while
```

Figura 6: Pseudocódigo de un Evolutionary algorithm con una Mutación diseñada para FAP

3.2.3. Genetic algorithm (GA)

Genetic Algorithm o Algoritmo Genético, es un algoritmo que se enmarca dentro de los algoritmos Evolutivos (EA), se basa en que dos o mas soluciones iniciales son los padres y de estas nacerán 2 o mas soluciones hijas, que compartirán ciertos genes de los padres, a esto se le suma la mutación de genes para crear variedad de nuevas soluciones.

```
procedure Genetic(G, α)
1    Initialize P;
2    c_i = cost(P_i), i = 1,... α;
3    C = min{c_i};
4    f ← arg min{c_i};
5    DO until no improvement in C
6        Q = SelectParents(P);
7        R = CreateChildren(Q);
8        P ← MergeChildren(R, P);
9        c_i = cost(P_i), i = 1,... |P|;
10       if min{c_i} < C
11           f ← arg min{c_i}, C ← min{c_i};
12       fi
13   OD;
14   return f;
end Genetic;
```

Figura 7: Pseudocódigo de un Genetic algorithm

En la tabla siguiente podemos observar un Standard Steady State Genetic Algorithm (SSGA) diseñado específicamente para FAP [3]

```
1: population ← ∅
2: initialize(population)
3: while not time-limit do
4:     parents ← binaryTournament(population)
5:     offspring ← UX(parents,p_c)
6:     offspring ← randomMutation(offspring,p_m)
7:     offspring ← localSearch(offspring,localSearchSteps)
8:     population ← insert(population,offspring)
9: end while
```

Figura 8: Pseudocódigo de un Standard Steady State Genetic Algorithm

3.2.4. Scatter Search

Scatter Search o Búsqueda de Dispersión es un algoritmo que funciona con un conjunto pequeño de soluciones llamado RefSet, este conjunto está compuesto por las soluciones más representativas de la población. La población inicial se genera aleatoriamente. RefSet se divide en dos grupos, las soluciones de calidad y las soluciones mas diferentes. El número de individuos para cada subconjunto se ha configurado especialmente para resolver el problema de FAP.

```
1: initialize(population)
2: population ← localSearch(population)
3: RefSet ← generateFrom(population)
4: while not time-limit do
5:     SubSet ← subSetGenerator(RefSet)
6:     SubSet ← combinationMethod(SubSet)
7:     RefSet ← localSearch(SubSet)
8:     RefSet ← generateFrom(RefSet, population)
9: end while
```

Figura 9: Pseudocódigo de un Scatter Search Optimizado para FAP

3.3. Resultados Experimentales

Resultados Empíricos de los distintos métodos heurísticos para cuatro lapsos de tiempo, para Las ciudades de Seattle y Denver respectivamente [3].

- **Denver**: 2612 antenas y 18 frecuencias.
- **Seattle:** 970 antenas y 15 frecuencias.

Parámetros:

- **SSGA:** Population size = 10, uniform crossover with pc = 1.0, random mutation with pm = 0.2, selection with binary tournament, replacement = worst individual
- **SS:** Population size = 40, RefSet size = 9, Solution combination method = uniform crossover
- **EA:** softBloq = 50, hardBloq = 300, maxPopSize = 5, pmut = 0.9, mutSelected = 7
- **LSHR:** Learning rate fr = 0.001

Los resultados fueron realizados bajo las mismas condiciones y configuración de hardware y software, fueron realizadas 30 veces seguidas por cada algoritmo, cada ejecución independiente de la anterior para asegurar consistencia

- **Procesador:** Intel Xenon a 3Ghz.
- **RAM:** 2GB.
- **SO:** Red Hat Linux 2.4.21-4.
- **Software:** GCC version 3.2.3.

En las Tablas siguientes se pueden analizar los resultados, la **media** y **desviación estándar** de la función de coste, donde el **menor** valor es el **mejor** resultado.

3.3.1. Seattle

Seattle	120 s	600 s	1800 s	3600 s
SSGA	$1894.67_{\pm 79.83}$	$1757.60_{\pm 87.49}$	$1676.12_{\pm 63.02}$	$1628.05_{\pm 51.17}$
SS	$2115.56_{\pm 191.43}$	$1606.85_{\pm 183.21}$	$1348.74_{\pm 145.95}$	$1238.68_{\pm 141.55}$
(1+1) EA	$1417.28_{\pm 141.94}$	$1142.65_{\pm 108.47}$	$989.30_{\pm 73.07}$	$917.43_{\pm 51.92}$
LSHR	$1677.73_{\pm 208.45}$	$1341.40_{\pm 123.48}$	$1194.13_{\pm 105.85}$	$1102.13_{\pm 115.49}$

Figura 10: Media y Desviación Estándar de resultados de Seattle para distintos tiempos de los distintos algoritmos heurísticos

3.3.2. Denver

Denver	120 s	600 s	1800 s	3600 s
SSGA	$89540.41_{\pm 1008.14}$	$87850.79_{\pm 583.45}$	$86908.94_{\pm 386.37}$	$86870.40_{\pm 320.02}$
SS	$89401.36_{\pm 1091.63}$	$87233.42_{\pm 874.73}$	$86122.66_{\pm 666.58}$	$85525.17_{\pm 494.54}$
(1+1) EA	$89798.47_{\pm 1305.70}$	$87859.68_{\pm 1038.68}$	$86835.99_{\pm 1016.04}$	$86363.98_{\pm 790.68}$
LSHR	$92946.57_{\pm 1519.37}$	$89680.33_{\pm 902.81}$	$88646.53_{\pm 526.92}$	$88367.90_{\pm 406.18}$

Figura 11: Media y Desviación Estándar de resultados de Resultados de Denver para distintos tiempos de los distintos algoritmos heurísticos

4. Modelo Matemático

Modelo matemático obtenido en [1]

4.1. Variables

- V,F = Conjunto de Antenas y Frecuencias respectivamente
- F_v = Conjunto de frecuencias disponibles para la antena v
- d_{vw} = Distancia mínima entre las frecuencias f y g asignadas a v y w, donde $|f - g| \leq d_{vw}$
- $X_{vf} = \begin{cases} 1 & \text{si la frecuencia } f \text{ se asigna a la antena } v \\ 0 & \text{si no} \end{cases}$
- $Y_f = \begin{cases} 1 & \text{si la frecuencia } f \text{ esta en uso} \\ 0 & \text{si no} \end{cases}$

4.2. Función Objetivo y Restricciones

Función Objetivo:

$$min \sum_{f} Y_f$$

Sujeto a:

$$X_{vf} \leq Y_f, \quad \forall v \in V, f \in F_v \tag{1}$$

$$\sum_{f_v} X_{vf} = 1, \quad \forall v \in V \tag{2}$$

$$X_{vf} + X_{wg} \leq 1, \quad |f - g| \leq d_{vw} \quad \forall f \in F_v, g \in G_w \tag{3}$$

Explicación:

- **Función Objetivo:** En simples palabras se busca minimizar la cantidad de frecuencias en uso.
- **Restricción 1:** Una Frecuencia se puede asignar solamente si esa frecuencia esta en uso.
- **Restricción 2:** Cada antena debe tener asignada una sola frecuencia.
- **Restricción 3:** entre 2 antenas donde exista interferencia, se asignara a lo mas una de las dos frecuencias.

5. Representación

5.1. Representación de la Solución

la representación utilizada para la solución del problema es:

$$[f_1, f_2, f_3, ..., f_n]$$

es una lista/vector que contiene las frecuencias escogidas que cumplen con las restricciones, mientras menor sea el largo de la lista mas óptima es la solución porque lo que buscamos es minimizar la cantidad de frecuencias, pero siempre teniendo en cuenta de cumplir con las restricciones, de otra forma la solución seria trivial siendo una lista vacía sin cumplir ninguna restricción.

5.2. Otras Representaciones

para poder obtener la lista solución se necesitan utilizar otras representaciones que ayudaran a encontrar la mejor solución, las estructuras de datos utilizadas son:

- **antenas:** $[1, 1, 2, 1, .., 6..., 3]$ lista/vector utilizada para almacenar el dominio que utiliza cada antena, esta lista tiene tamaño variable, según el numero total de antenas del problema en el ejemplo la antena 1 utiliza el dominio 1, la antena 2 también utiliza el dominio 1, la antena 3 utiliza el dominio 2 y así sucesivamente, las antenas van desde 1 hasta n, y los dominios van desde 1 a 7, por eso en la lista solo hay números del 1 al 7 y el índice representa la antena.
- **dominios:** $[[9, 13, 17, 25..], [13, 34, 50], [2, 17..], ..., [9, 50, .., 80]]$ lista de listas o vector de vectores, donde el índice representa el dominio y la lista las frecuencias a las cuales se puede acceder, la lista tiene tamaño 8, pero las listas dentro de esta lista tienen largo variable dependiendo el problema, en el índice 0 va el dominio general, con todas las frecuencias del problema, y después del índice 1 al 7 van los respectivos dominios, en el índice 1 se encuentra la lista con frecuencias del dominio 1 y así sucesivamente hasta el 7.

- **índices de antenas:** $[13, 15, 25, 53, 57..]$ lista/vector que contiene el numero de cada antena y su índice se utiliza para encontrar el dominio de la antena en la lista de antenas, esta lista tiene tamaño variable, según el numero total de antenas del problema.
  según el problema, las antenas no siempre parten desde 1 y se pueden saltar números, por eso se necesita esta lista para indexarla con la lista de antenas y así obtener su dominio, para este ejemplo la antena 13 se encuentra en el índice 0, con este índice 0 en la lista de antenas operamos **antenas[0]** y obtenemos que su dominio es 1, con este dominio 1 en la lista de dominios operamos **dominios[1]** y obtenemos que sus frecuencias disponibles son **[13,34,50]**

- **vecinos:** $[[13, 25, 90], [13, 25, 120], [13, 25, 300], ..., [13, 25, 800]]$ lista de listas o vector de vectores, de largo variable, cada lista dentro de esta lista de vecinos es un vecino que comparte las primeras n frecuencias y solo cambia la ultima, ya que el algoritmo parte de una lista vacía, genera todos los vecinos y selecciona el mejor, luego con esa nueva solución se generan todos los vecinos y se vuelve a seleccionar la mejor frecuencia, y así hasta que se completen las restricciones, en el ejemplo esta lista de vecinos se genero a partir de la solución **[13,25]**, se itero sobre todas las frecuencias que no están en la solución y se generaron todas las combinaciones para crear la lista de vecinos, de aquí se seleccionara al mejor vecino y se procederá a crear una nueva solución ahora con 3 frecuencias y se generara una nueva lista de vecinos.

- **Valores:** $[5, 2, 17, 100, 30, 20...]$ lista/vector que contiene los distintos valores de evaluación para los distintos vecinos según su índice, es decir que el vecino n, tiene un valor representado en la lista por el valor n, tiene largo variable y es igual a la cantidad de vecinos, en este ejemplo el vecino 0 que proviene de **vecinos[0]** tiene un valor de 5 que proviene de **valores[0]**, esta lista nos permite comparar los valores de los distintos vecinos y obtener el mejor de todos.

6. Descripción del algoritmo

Para resolver el MO-FAP se utilizo un enfoque heurístico de Hill Climbing con Mejor Mejora, esto significa que se iterara hasta llegar al óptimo global seleccionando a el mejor de todos los vecinos generados, que sea mejor que la solución actual, hasta que se cumplan todas las restricciones.

6.1. Solución Inicial

Como solución inicial se utilizo una lista vacía, la cual no cumple con ninguna restricción por lo que tiene valor 0, de esta solución inicial se generan los vecinos representados por una lista de listas, de la cual se va a seleccionar la mejor de entre todas y como es mejor que una solución de valor 0, la vamos a reemplazar, así continuaremos iterativamente el algoritmo, como la lista parte vacía y se van agregando frecuencias en cada iteración, una vez que se cumplan todas las restricciones, esa sera la lista con menor tamaño que resuelve el problema, minimizando y resolviendo MO-FAP.

6.2. Pseudocódigo del algoritmo

Listing 1: MOFAP Algoritmo Hill Climbing + Mejor Mejora

```
antennas,index_antennas = get_antennas(text_file)                    #1
domains = get_domains(text_file)                                     #2
sol = []                                                             #3
max_constraints = get_max_constraints(text_file)                     #4
total_frequencies = domain[0]                                        #5
constraints_lines = read_file(txt_file)                              #6
temp_sol = sol                                                       #7

for i in range (total_frecuencies):                                  #8
    neighbours = gen_neighbours(temp_sol)                            #9
    values = []                                                      #10
    total_neighbours = len(neighbours)                               #11
    for j in range(total_neighbours):                                #12
        value = 0                                                    #13
        neighbour = neighbours[j]                                    #14
        for line in constraints_lines:                               #15
            value = value + check_constraint(line,neighbour)        #16
        values.append(value)                                         #17
    best_value_index = get_index(max(values))                        #18
    best_value = neighbours[best_value_index]                        #19
    if best_value == max_constraints:                                #20
        sol = best_value
        return sol
    if best_value > temp_sol:                                        #21
        temp_sol = best_value
        sol = best_value
    if best_value == temp_sol:                                       #22
        temp_sol = best_value
```

6.3. Explicacion

1. obtenemos las antenas y sus índices
2. obtenemos los dominios
3. solución inicial que después sera la final, lista vacía
4. obtenemos el máximo de restricciones
5. obtenemos todas las frecuencias disponibles
6. obtenemos las lineas del archivo de restricciones
7. solución temporal para el algoritmo
8. iteramos hasta que se cumplan las restricciones o hasta que recorramos todas las frecuencias disponibles
9. generamos los vecinos según la solución temporal
10. se inicializa la lista de valores que contendrá el valor de los distintos vecinos
11. obtenemos el total de vecinos
12. iteramos por cada vecino
13. se inicializa el valor del vecino en 0
14. se escoge el vecino
15. se itera en cada una de las lineas de las restricciones
16. si el vecino cumple con la restricción aumenta su valor
17. se agrega el valor final del vecino a la lista de valores
18. se obtiene el índice del mayor valor de la lista de valores
19. se obtiene al mejor vecino
20. si el mejor vecino cumple con todas las restricciones, retornamos la solución final
21. si el mejor vecino es mejor que nuestra solución actual, la reemplazamos tanto para la solución temporal, como para la solución final
22. si el vecino es igual a la solución actual, o no hay ningún vecino mejor, se reemplaza la solución temporal para agregar una frecuencia mas y generar mas vecinos

El algoritmo en si lo que hace es partir de una solución inicial que es una lista vacía, genera los vecinos agregando cada una de las frecuencias disponibles y escoge la mejor de todas esas frecuencias, así opera iterativamente en cada iteración seleccionando una frecuencia, que es la mejor de las que no están ya agregadas a la solución, así hasta que se cumplan todas las restricciones, gracias a esta forma de operar siempre se asegura que se retorne la menor lista posible de frecuencias que cumplan con las restricciones.

7. Experimentos

7.1. Metodología

Para poder evaluar y experimentar con el algoritmo previamente detallado, se utilizaron 11 instancias distintas unas de otras, cada una con su conjunto de antenas y su conjunto de frecuencias, cada uno estos escenarios contienen 3 archivos VAR.TXT, DOM.TXT y CTR.TXT, de los cuales se leen los parámetros para poder ejecutar el algoritmo.

- **VAR.TXT:** Por cada linea de este archivo tenemos el numero de la antena y su correspondiente dominio, esto nos permite asociar cada antena con su conjunto de frecuencias por medio del archivo DOM.TXT

Listing 2: VAR.TXT

```
1 1
2 1
3 2
. .
. .
916 7
```

- **DOM.TXT:** Cada linea de este archivo representa los distintos dominios, en primer lugar tenemos el numero del dominio, después la cantidad de frecuencias en ese dominio y por ultimo las frecuencias ordenadas de menor a mayor

Listing 3: DOM.TXT

```
0  48  16  30  44  58  ... 750 764 778 792
1  44  16  30  44  58  ... 764 778 792
.  .   .   .   .   .   .   .   .   .
.  .   .   .   .   .   .   .   .   .   .
7  22  16  30  44  58  72 ... 352 366 380 394
```

- **CTR.TXT:** Cada linea de este archivo representa las restricciones entre antenas, los dos primeros números representan el numero de cada antena, después existe una letra que se puede obviar para los experimentos, luego viene la diferencia absoluta de sus frecuencias y por ultimo el numero objetivo a cumplir, en el caso de la primera linea del ejemplo de mas abajo la frecuencia de la antena 1 menos la frecuencia de la antena 2 tiene que ser igual a 238, $|F_1 - F_2| = 238$, en el caso de la segunda linea seria $|F_1 - F_{171}| > 8$

Listing 4: CTR.TXT

```
1    2 D = 238
1  171 L >   8
1  172 C >  56
.  .   .  .  .
.  .   .  .  .
911 912 D = 238
913 914 D = 238
915 916 D = 238
```

7.2. Entorno de Experimentación

Los resultados fueron realizados bajo las mismas condiciones y configuración de hardware y software,los resultados fueron ejecutados 10 veces seguidas por cada instancia de problema con su respectiva cantidad de antenas y frecuencias, cada ejecución independiente de la anterior para asegurar consistencia

- **Procesador:** Intel core i5 7300HQ a 2.50Ghz.
- **RAM:** 16GB.
- **SO:** Linux Pop!_OS 21.04.
- **Software:** G++ versión 10.3.0.

Como el algoritmo propuesto es Hill Climbing con Mejor Mejora, que es un algoritmo que va reparando la solución siempre tomando soluciones mejores que la original, no hay parámetros que variar ni cambiar (no hay temperatura, ni lista tabú, etc) por lo que los resultados son deterministas, lo que varia en cada instancia es el numero de antenas, restricciones y las frecuencias disponibles.

8. Resultados

Se realizaron un total de 10 iteraciones del algoritmo por cada una de las 11 instancias, se calculo el tiempo promedio de ejecución de cada instancia y se comparo versus la cantidad de antenas y la cantidad de restricciones, podemos observar la interesante correlación entre el numero de restricciones y el tiempo de ejecución del algoritmo, así como también influye el numero de antenas de cada problema en el tiempo de ejecución,

8.1. Complejidad y Gráficos

Instancia	**Numero de Antenas**	**Numero de Restricciones**	**Tiempo de Ejecución Promedio [ms]**
01	916	5548	6393.8
02	200	1235	564.1
03	400	2760	2869
04	680	3967	6602.4
05	400	2598	3450.2
06	200	1332	912.4
07	400	2865	2323.6
08	916	5744	11582.6
09	680	4103	6875.1
10	680	4103	6879.9
11	680	4013	5340.8

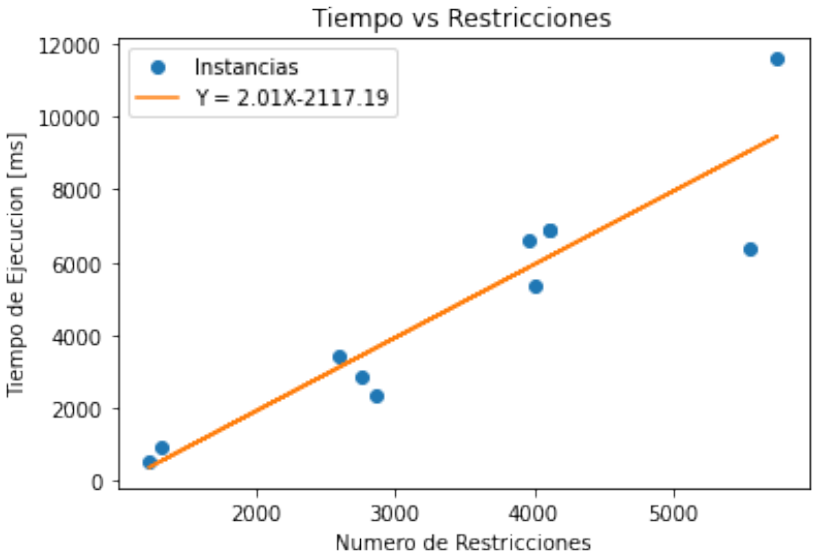

Figura 12: Gráfico Representando el tiempo de Ejecución Promedio en milisegundos versus el numero de Restricciones de la instancia

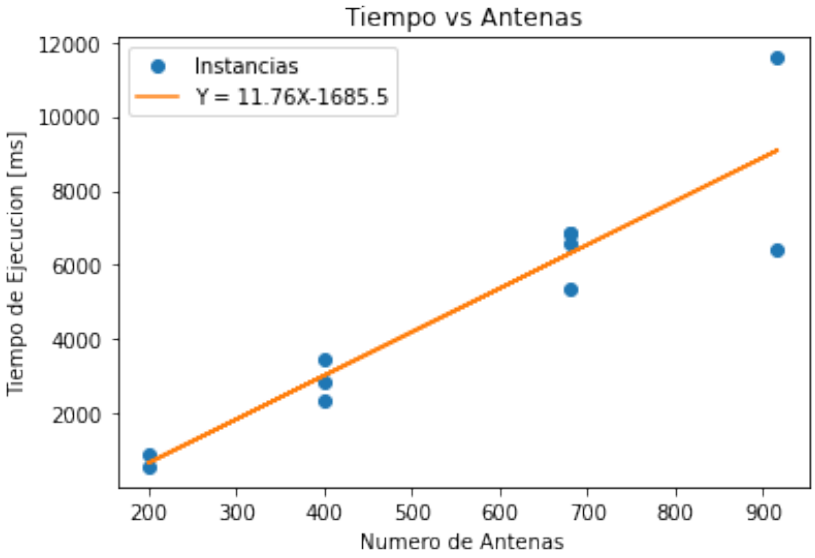

Figura 13: Gráfico Representando el tiempo de Ejecución Promedio en milisegundos versus el numero de Antenas de la instancia

8.2. Rendimiento del Algoritmo

Como se puede observar en los dos gráficos, el tiempo de ejecución va incrementando a medida que aumentamos las antenas o aumentamos las restricciones del problema, al comparar las ecuaciones de ambas rectas de regresión, $Y_1 = 2{,}01X - 2117{,}19$ para el gráfico de Tiempo vs Restricciones y $Y_2 = 11{,}76X - 1685{,}5$ para el gráfico de Tiempo vs Antenas, se puede notar que la pendiente de la segunda recta es mayor que la primera recta, esto significa que a medida que

aumentamos las antenas el tiempo va a crecer mas rápido que si aumentamos las restricciones.Se puede observar también que el algoritmo es bastante rápido, de las 11 instancias solo una demoro en promedio mas de 10 segundos, y en general si ponderamos todos los tiempos promedios obtenemos un tiempo promedio general entre las 11 instancias de 5 segundos aproximadamente, $T_{Promedio} = 4890{,}45[ms]$

9. Conclusiones

A partir de los resultados observados podemos notar como las técnicas heurísticas son mejores cuando la complejidad del problema es muy grande, como es el caso real de las ciudades de Seattle y Denver, por otro lado cuando la complejidad no es tan grande es preferible utilizar un método mas simple como ILP En cuanto a los cuatro algoritmos Heurísticos propuestos, se puede observar en la tabla de resultados como el mejor para la ciudad de Seattle fue el Algoritmo Evolutivo Modificado para FAP ((1+1)EA), obteniendo en cada lapso de tiempo, el menor valor de entre todos los algoritmos.

En cuanto para la ciudad de Denver, podemos observar que al aumentar las antenas se disparan los valores de la función objetivo a minimizar, y se puede observar que ahora la diferencia no es tan notoria como en la ciudad de Seattle, todos los algoritmos obtienen aproximadamente las mismas soluciones excepto por LSHR que es un poco peor que los otros 3 algoritmos (SSGA, SS y (1+1)EA)

Se puede observar que SSGA y (1+1)EA, son algoritmos muy parecidos, ambos se basan en la evolución de sus soluciones, en cambio LSHR y SS optan por otro enfoque, de mejorar el espacio de búsqueda, cada algoritmo tiene sus ventajas y sus limitaciones, pero se puede observar según los resultados como los algoritmos Evolutivos/Genéticos son los mas óptimos para resolver MO-FAP.

Existen variados métodos para poder resolver el MO-FAP, se han listado los mas relevantes pero con el avance de la tecnología se han desarrollado nuevas tecnologías muy prometedoras para resolver este tipo de problemas, como lo pueden ser las **Graph Neural Networks**, Redes Neuronales basadas en grafos, y como se pudo observar el MO-FAP es un problema que se puede representar fácilmente a través de grafos ya que es un problema muy parecido al de coloreo de grafos.

Por ultimo cabe resaltar que todos estos enfoques se pueden combinar para crear mejores algoritmos, una idea para investigaciones futuras podría ser las **Genetic Graph Neural Networks** (GGNN), aplicar el método genético (que fue el método heurístico que mejor resultados obtuvo) para un grupo de soluciones compuestas por Graphs Neural Networks (GNN), por el momento combinar algoritmos genéticos con redes neuronales es bastante costoso en términos computacionales, por eso existe un gran trabajo a futuro para poder lograr estos algoritmos.

10. Bibliografía

Referencias

[1] Karen Aardal, Stann van Hoesel, Arie Koster, Carlo Mannino, and Antonio Sassano. Models and solution techniques for frequency assignment problems. 2007.

[2] Peter Jeavons. Towards high order constraint representations for the frequency assignment problem. 1998.

[3] Francisco Luna, Cesar Estebanez, Coromoto Leon, Jose Chaves-Gonzalez, Antonio Nebro, Ricardo Aler, Carlos Segura, Miguel A. Vega-Rodrıguez, Enrique Alba, Jose M. Valls, Gara Miranda, and Juan A. Gomez-Pulido. Optimization algorithms for large-scale real-world instances of the frequency assignment problem.

[4] Zoltán Ádám Mann and Anikó Szajkó. Complexity of different ilp models of the frequency assignment problem. 2012.