Empresa Telefónica

Olivia Ibañez Mustelier
Ciencia de la Computación,C411

Resumen

Este trabajo aborda el **Problema de Asignación de Frecuencias con Costos Mínimos (AFCM)** en redes de telefonía celular, un desafío operativo crítico en telecomunicaciones. Demostramos que AFCM es NP-completo mediante una reducción formal desde el problema de coloración de grafos. Se implementan y comparan múltiples estrategias algorítmicas: un algoritmo exacto de fuerza bruta para validación en instancias pequeñas, un algoritmo exacto en tiempo polinomial para el caso especial de árboles (usando programación dinámica), y algoritmos aproximados y heurísticos (greedy, búsqueda local, búsqueda tabú) para el caso general. Los resultados experimentales, obtenidos sobre tres tipos de grafos (Erdős-Rényi, árboles aleatorios y geométricos), validan el análisis teórico y muestran que la combinación *greedy + búsqueda local* ofrece el mejor equilibrio práctico entre tiempo de ejecución y calidad de la solución, logrando mejoras de un 5–15 % respecto al greedy solo. Este estudio no solo profundiza en un problema clásico de optimización combinatoria, sino que también proporciona herramientas aplicables a la planificación real de redes.

Índice

1. Presentación del Problema Original

1.1. Contexto

En el sector de las telecomunicaciones móviles, la gestión eficiente del espectro radioeléctrico es un pilar fundamental para la calidad del servicio y la viabilidad económica. Operadoras como *ConectaMax Telecom*—nombre ficticio inspirado en casos reales—se enfrentan diariamente al complejo reto de asignar frecuencias a sus miles de torres de telefonía celular. La física de la propagación de ondas y las regulaciones imponen una restricción irrenunciable: dos torres geográficamente cercanas no pueden operar en la misma frecuencia, so pena de generar interferencias que degradarían severamente la comunicación.

Desde mi experiencia analizando problemas de optimización en redes, el verdadero desafío suele estar en los detalles económicos. El costo de operar una torre en una frecuencia específica no es uniforme. Depende de un entramado de factores: el equipamiento hardware instalado (que puede ser más eficiente en ciertas bandas), el consumo energético asociado, e incluso tarifas regulatorias y de licencia que varían por región y por banda espectral. Una asignación subóptima, lejos de ser un mero ejercicio académico, puede traducirse en millones de dólares en costos operativos innecesarios anuales para una operadora de gran escala.

Modelamos estos factores reales mediante una **matriz de costos torre–frecuencia** (o función $w(v, c)$), una **abstracción estándar en optimización combinatoria** que permite incorporar heterogeneidad económica sin perder la estructura del problema.

1.2. Enunciado Formal del Problema

El problema concreto que nos ocupa, extraído de la descripción operativa de *ConectaMax Telecom*, puede enunciarse de la siguiente manera:

Problema AFCM (Asignación de Frecuencias con Costos Mínimos). Dado un conjunto de torres de telefonía celular y un conjunto de frecuencias disponibles, se debe encontrar una asignación de una frecuencia a cada torre que satisfaga:

1. **Restricción de interferencia:** Dos torres que están suficientemente cerca (según un umbral de distancia definido) no pueden compartir la misma frecuencia.
2. **Objetivo de minimización:** El costo total de la asignación, donde el costo de asignar una frecuencia específica a una torre específica es un valor numérico positivo dado, debe ser minimizado.

Ahora bien, para poder analizarlo con las herramientas de la ciencia de la computación, es necesario traducir este enunciado natural a un modelo matemático riguroso. Este paso de modelación es, a mi parecer, tan crucial como el análisis posterior.

2. Modelación y Formalización Matemática

2.1. De Torre y Frecuencias a Vértices y Colores

La abstracción hacia la teoría de grafos resulta casi natural. Cada torre se representa como un **vértice** en un grafo no dirigido $G = (V, E)$. La relación de cercanía o interferencia potencial se codifica mediante una **arista** entre dos vértices: si dos torres están lo suficientemente cerca como para interferir, se traza una arista entre ellas. El conjunto de frecuencias disponibles, que para nuestro caso asumimos finito y discreto, se equipara a un conjunto de **colores** $C = \{1, 2, \ldots, k\}$.

La novedad—y lo que hace al problema económicamente relevante—es la incorporación de costos. Definimos una **función de costo** $w : V \times C \to \mathbb{R}^+$, donde $w(v, c)$ representa el costo (en unidades monetarias, por ejemplo) de asignar la frecuencia (color) c a la torre (vértice) v. Esta función captura todos aquellos factores heterogéneos mencionados: eficiencia del hardware, precios de licencia, etc.

2.2. Definición Matemática del Problema AFCM

Con esta analogía, estamos frente a una generalización del clásico problema de coloración de grafos, conocido en la literatura como *Minimum-Cost Graph Coloring* [8] y, en el contexto de telecomunicaciones, como **Asignación de Frecuencias con Costos Mínimos (AFCM)**.

Definición 1 (Problema AFCM de Optimización)**.** *Dado:*

- *Un grafo no dirigido $G = (V, E)$ con $|V| = n$ vértices.*
- *Un conjunto de k colores (frecuencias) $C = \{1, 2, \ldots, k\}$.*
- *Una función de costos $w : V \times C \to \mathbb{R}^*$.*

Hallar una asignación (coloración) $f : V \to C$ que:

1. *Sea **válida**: $\forall (u, v) \in E,\ f(u) \neq f(v)$.*
2. *Minimice el **costo total**: $Costo(f) = \sum_{v \in V} w(v, f(v))$.*

Para facilitar el análisis de complejidad, definimos también la versión de decisión.

Definición 2 (Problema AFCM de Decisión (AFCM-D))**.** *Dada una instancia* (G, C, w) *y un umbral* $T \in \mathbb{R}^*$*, ¿existe una asignación válida* f *tal que* $Costo(f) \leq T$*?*

Esta formalización es el punto de partida para todo el análisis subsiguiente. Resulta claro que si todos los costos son iguales, el problema se reduce a decidir si el grafo es k-coloreable.

2.3. Formulación alternativa como ILP (Teórico)

De manera complementaria, AFCM puede formularse como un modelo de programación entera lineal 0–1. Para cada vértice $v \in V$ y cada color $c \in C$, se define una variable binaria

$$x_{v,c} = \begin{cases} 1, & \text{si el vértice } v \text{ usa el color } c, \\ 0, & \text{en otro caso.} \end{cases}$$

El objetivo es minimizar el costo total:

$$\text{mín} \sum_{v \in V} \sum_{c \in C} w(v, c)\, x_{v,c}.$$

Sujeto a las restricciones:

(a) **Asignación única por vértice:** $\sum_{c \in C} x_{v,c} = 1 \quad \forall v \in V.$

(b) **Interferencia (adyacencia):** $x_{u,c} + x_{v,c} \leq 1 \quad \forall \{u, v\} \in E,\ \forall c \in C.$

(c) **Integralidad:** $x_{v,c} \in \{0, 1\} \quad \forall v \in V,\ \forall c \in C.$

Esta formulación no se implementa en este trabajo, pero resulta útil como referencia teórica y permitiría resolver instancias pequeñas con solvers ILP.

3. Complejidad Computacional: Demostración de NP-Completitud

3.1. Preliminar: El Problema de Coloración de Grafos (k-COLORING)

Para demostrar la dureza NP de AFCM, nos apoyamos en un problema canónico: el problema de coloración de grafos.

Definición 3 (k-COLORING)**.** *Dado un grafo no dirigido* $G = (V, E)$ *y un entero positivo* k*, ¿existe una función* $f : V \to \{1, \ldots, k\}$ *tal que para toda arista* $(u, v) \in E$*,* $f(u) \neq f(v)$*?*

k-COLORING es NP-completo para $k \geq 3$, un resultado fundamental establecido por Karp en 1972 [1] y recogido en el clásico de Garey y Johnson [2]. Esta NP-completitud se sostiene para grafos no dirigidos generales, que es justamente el caso que nos interesa.

3.2. Pertenencia a NP

Teorema 1. *AFCM-D pertenece a la clase NP.*

Demostración. Dado un certificado consistente en una asignación $f : V \to C$, un verificador puede comprobar en tiempo polinomial:

1. **Validez:** Para cada arista $(u, v) \in E$, verificar que $f(u) \neq f(v)$. Esto requiere $O(|E|)$ operaciones.
2. **Costo:** Calcular $S = \sum_{v \in V} w(v, f(v))$. Cada suma es $O(1)$, por lo que el total es $O(n)$.
3. **Umbral:** Comparar $S \leq T$ en $O(1)$.

El tiempo total de verificación es $O(|E|+n)$, que es polinomial en el tamaño de la entrada (el grafo y la matriz de costos). □

3.3. Reducción Polinomial desde k-COLORING

La esencia de la demostración de NP-dureza reside en mostrar que un problema conocido NP-completo, como k-COLORING, puede transformarse en nuestro problema.

Antes de presentar la reducción formal, conviene remarcar por qué la elección $w'(v, c) = 0$ y $T' = 0$ fuerza la equivalencia con k-COLORING. Al fijar costos nulos se "apaga" el componente económico del problema: cualquier asignación válida tendrá necesariamente costo total 0 (y cualquier asignación inválida no es aceptada por las restricciones). Por tanto, la pregunta "¿existe una solución con costo ≤ 0?" se convierte exactamente en "¿existe una coloración válida con k colores?". Esta idea es la que la reducción formaliza a continuación.

Teorema 2. *AFCM-D es NP-completo.*

Demostración. Realizamos una reducción polinomial desde k-COLORING. Sea $I = \langle G = (V, E), k\rangle$ una instancia arbitraria de k-COLORING. Construimos una instancia $I' = \langle G', C', w', T'\rangle$ de AFCM-D de la siguiente forma:

- $G' = G$ (el mismo grafo).
- $C' = \{1, 2, \ldots, k\}$ (los k colores).
- Para todo $v \in V$ y todo $c \in C'$, definimos $w'(v, c) = 0$.
- Fijamos el umbral $T' = 0$.

Esta construcción es claramente computable en tiempo polinomial (incluso lineal).

Correctitud de la reducción. Debemos probar que I es una instancia *sí* de k-COLORING si y solo si I' es una instancia *sí* de AFCM-D.

($\Rightarrow$) Si G es k-coloreable, existe una coloración válida $f : V \to C'$. Esta f es también una asignación válida para I', y su costo total es 0. Por lo tanto, $0 \leq T'$, e I' es una instancia sí de AFCM-D.

($\Leftarrow$) Si existe una asignación f para I' con costo $\leq T' = 0$, dado que todos los costos $w'(v, c)$ son no negativos, el costo total debe ser exactamente 0. Esto implica que f asigna a cada vértice un color con costo 0 (lo cual es trivialmente cierto para todos). Además,

f es una coloración válida porque satisface las restricciones de interferencia de AFCM-D. Por consiguiente, f es un k-coloreo válido de G.

Como k-COLORING es NP-completo y hemos reducido polinomialmente k-COLORING a AFCM-D, concluimos que AFCM-D es NP-completo. □

Corolario 1. *El problema de optimización AFCM es NP-duro.*

Demostración. Si existiera un algoritmo polinomial para AFCM (optimización), podríamos resolver AFCM-D en tiempo polinomial: calcularíamos el costo mínimo y lo compararíamos con T. Esto contradice la NP-completitud de AFCM-D, a menos que P = NP. □

3.4. Implicaciones Prácticas de la NP-Completitud

Este resultado no es solo una curiosidad teórica. Tiene consecuencias muy concretas para cualquier ingeniero o planificador que enfrente este problema:

1. **No existe un algoritmo polinomial exacto** para resolver instancias generales de AFCM, a menos que (de forma bastante inesperada) P = NP.
2. Por tanto, cualquier algoritmo exacto para el caso general tendrá, en el peor de los casos, un **tiempo de ejecución exponencial** en el tamaño de la entrada.
3. Esta dureza nos fuerza a explorar vías alternativas: diseñar algoritmos exactos para **casos especiales** de grafos (como los árboles), emplear **algoritmos de aproximación** con garantías teóricas sobre la calidad de la solución, o desarrollar **heurísticas y metaheurísticas** efectivas en la práctica, aunque sin tales garantías.

Personalmente, encuentro fascinante cómo un resultado de teoría de la computación, aparentemente abstracto, traza los límites de lo que es computacionalmente viable en un problema industrial tan tangible.

4. Estrategias de Solución

Dada la intrataibilidad del caso general, nuestro enfoque se bifurca: buscamos algoritmos exactos para casos manejables y algoritmos aproximados o heurísticos para el caso general, sin olvidar una línea base de fuerza bruta para validación.

4.1. Algoritmo Exacto de Fuerza Bruta (Backtracking)

Para instancias muy pequeñas ($n \leq 12$), implementamos un algoritmo de *backtracking* que explora sistemáticamente el espacio de todas las posibles asignaciones de colores. La idea es sencilla pero instructiva: se asigna colores a los vértices uno a uno, podando la búsqueda cuando el costo parcial supera el mejor costo total encontrado hasta el momento o cuando se viola una restricción de adyacencia.

Complejidad: En el peor caso, explora k^n nodos, por lo que su tiempo es exponencial: $O(k^n \cdot \text{poli}(n))$. Su utilidad principal en este proyecto fue servir como *oráculo de optimalidad* para validar la calidad de nuestras heurísticas en instancias diminutas.

4.2. Algoritmo Exacto para un Caso Especial: Árboles

Los árboles (grafos conexos acíclicos) son una familia de grafos sobre la que muchos problemas NP-duros se vuelven tratables. AFCM no es la excepción.

Teorema 3. *AFCM puede resolverse de manera óptima en árboles en tiempo* $O(n \cdot k^2)$ *mediante programación dinámica.*

Definición del subproblema (estado DP). Se fija una raíz arbitraria r y se orientan las aristas implícitamente hacia los hijos. Para cada vértice v y cada color $c \in C$, se define:

$dp[v][c]$ = costo mínimo de colorear el subárbol enraizado en v suponiendo $f(v) = c$.

Es decir, el subproblema restringe el color del vértice actual y optimiza el resto del subárbol.

Caso base. Si v es hoja (no tiene hijos excepto, quizá, su padre en la orientación), entonces el subárbol de v es el propio vértice, y:

$$dp[v][c] = w(v, c).$$

Recurrencia (paso de programación dinámica). Si v tiene hijos $u_1, \ldots, u_t$, entonces, dado que $f(v) = c$, cada hijo u_i debe elegir un color $d \neq c$. Como los subárboles de distintos hijos son disjuntos en un árbol, sus decisiones pueden optimizarse independientemente y sumarse:

$$dp[v][c] \;=\; w(v, c) \;+\; \sum_{u \in \text{Hijos}(v)} \min_{d \in C \setminus \{c\}} dp[u][d].$$

Respuesta final. El costo óptimo del árbol completo es:

$$\min_{c \in C} dp[r][c].$$

Por qué es exacto. La exactitud se sigue del *principio de optimalidad*: si fijamos $f(v) = c$, la elección óptima para cada subárbol hijo no depende de decisiones en otros subárboles, porque en un árbol no existen ciclos ni aristas "cruzadas" entre subárboles de hijos diferentes. Por tanto, tomar para cada hijo la mejor opción compatible ($d \neq c$) y sumar produce la solución global óptima para el subárbol de v.

Por qué solo funciona en árboles. En grafos con ciclos, los "subproblemas" ya no son independientes: un vértice puede conectarse con otro que pertenezca a un supuesto "subárbol" distinto, y entonces la decisión de color en una parte del grafo restringe simultáneamente múltiples regiones. Esa dependencia global rompe la posibilidad de combinar óptimos locales por simple suma como en la recurrencia anterior.

Cálculo de complejidad. Para cada vértice v y cada color c se calcula una suma sobre sus hijos. El término dominante es el cálculo de $\min_{d \neq c} dp[u][d]$ para cada hijo u y cada c. En forma directa, para un hijo fijo u y un color fijo c, se revisan $k - 1$ valores $dp[u][d]$, lo cual cuesta $O(k)$. Como esto se repite para cada $c \in C$ (son k colores), el costo por hijo es $O(k^2)$. Sumando sobre todos los hijos de todos los vértices, y usando que el número total de relaciones padre-hijo en un árbol es $n - 1$, se obtiene $O(n \cdot k^2)$.

La idea central del algoritmo es procesar el árbol desde las hojas hacia una raíz arbitraria. Para cada nodo v y cada color posible c, calculamos $dp[v][c]$, el costo mínimo del subárbol enraizado en v, dado que v se colorea con c. La recursión combina las soluciones óptimas de los hijos, forzando que estos usen un color diferente al de su padre.

La optimalidad se sigue del principio de optimalidad de la programación dinámica. Este algoritmo no solo es eficiente, sino que nos proporciona un valioso punto de comparación para evaluar heurísticas en instancias que son árboles.

Algorithm 1 Programación Dinámica para AFCM en Árboles

```
1: procedure DP-Tree(v, padre)
2:     for c ∈ C do
3:         dp[v][c] ← w(v, c)
4:     end for
5:     for u ∈ Ady(v) do                                  ▷ Iterar sobre los hijos
6:         if u ≠ padre then
7:             DP-Tree(u, v)
8:             for c ∈ C do
9:                 min_costo_hijo ← ∞
10:                for d ∈ C \ {c} do
11:                    min_costo_hijo ← mín(min_costo_hijo, dp[u][d])
12:                end for
13:                dp[v][c] ← dp[v][c] + min_costo_hijo
14:            end for
15:        end if
16:    end for
17: end procedure
```

4.3. Algoritmos Aproximados y Heurísticos para el Caso General

Para grafos arbitrarios, nos centramos en estrategias que sacrifiquen optimalidad a cambio de eficiencia.

4.3.1. Algoritmo Greedy Secuencial

Este es probablemente el enfoque más intuitivo. Ordenamos los vértices de mayor a menor grado (idea basada en la heurística *DSATUR* para coloración) y, en ese orden, asignamos a cada vértice el color válido (no usado por un vecino ya coloreado) de menor costo.

Complejidad: $O(n \log n + n \cdot \Delta \cdot k)$, donde Δ es el grado máximo. Es extremadamente rápido.

Análisis de Aproximación: En el peor caso, su factor de aproximación puede ser tan malo como $O(\Delta)$. Consideremos un grafo estrella con centro u y Δ hojas. Si el color más barato para u es c_1 (costo 0) y para las hojas también es c_1 (costo 0), pero un algoritmo greedy colorea primero a u con c_1, obliga a todas las hojas a tomar otros colores con costo M. El costo total sería $\Delta \cdot M$, mientras que el óptimo (colorear u con otro color de costo M y las hojas con c_1) cuesta solo M. La relación es Δ.

4.3.2. Búsqueda Local (Hill Climbing)

Partiendo de una solución inicial (la del greedy, por ejemplo), la búsqueda local intenta mejorarla realizando cambios incrementales. La variante más simple, *ascenso de colina*, evalúa para cada vértice el cambio a un color diferente que reduzca el costo total, realizando el primer movimiento de mejora encontrado. Se repite hasta alcanzar un óptimo local.

Para escapar de óptimos locales pobres, implementamos también una versión de **Búsqueda Tabú** [6]. Esta mantiene una memoria a corto plazo (lista tabú) de movimientos recientes que están prohibidos, forzando la exploración de nuevas zonas del espacio de búsqueda. Se incluye un *criterio de aspiración* que permite sobrepasar la tabuidad si el movimiento conduce a una solución mejor que la mejor global encontrada. Los parámetros clave—tamaño de la lista tabú y número máximo de iteraciones—se fijaron en 7 y 1000 respectivamente tras algunas pruebas preliminares.

Desde mi punto de vista, la belleza de la búsqueda tabú está en su simplicidad conceptual y su efectividad práctica para problemas de optimización combinatoria como este.

4.3.3. Estrategia Híbrida Propuesta

La estrategia que ofreció el mejor balance en nuestros experimentos fue híbrida:

1. **Fase 1:** Generar una solución inicial rápida y razonable usando el **algoritmo greedy secuencial**.
2. **Fase 2:** Refinar esa solución aplicando **búsqueda local** (o búsqueda tabú para mayor refinamiento).

Esta combinación captura lo mejor de ambos mundos: la velocidad del greedy y la capacidad de mejora de la búsqueda local.

5. Implementación y Experimentación

5.1. Arquitectura del Software

El proyecto se implementó en Python 3.9, organizado en módulos para facilitar la extensión y reproducibilidad:

- `grafo.py`: Estructuras de datos para grafos (listas de adyacencia).
- `instancias.py`: Generadores de instancias de prueba.
- `algoritmos/`: Contiene los módulos de `fuerza_bruta.py`, `dp_arbol.py`, `greedy.py`, `busqueda_local.py`.
- `verificador.py`: Valida legalidad y calcula costo de una solución.
- `experimentos.py`: Orquesta la ejecución de pruebas y recolección de datos.
- `visualizador.py`: Genera gráficas (usando Matplotlib).

El código está disponible en un repositorio GitHub con licencia académica.

5.2. Metodología Experimental

5.2.1. Generación de Instancias

Para evaluar robustez, usamos tres modelos de grafos, cada uno con 50 instancias independientes por configuración (n, k):

1. **Grafos Aleatorios Erdős-Rényi ($G(n,p)$)**: Con $p = 0{,}3$. Modelan interconexiones aleatorias. [3]

2. **Árboles Aleatorios**: Generados via secuencias de Prüfer. Modelan topologías jerárquicas. [4]

3. **Grafos Geométricos Aleatorios**: n puntos uniformes en $[0,1]^2$, conectados si distancia $\leq 0{,}3$. Modelan distribución espacial real de torres. [5]

Los costos $w(v,c)$ se muestrearon de una distribución uniforme discreta en $[1,100]$, simulando variabilidad de costos.

5.2.2. Configuración de los Experimentos

- **Tamaños (n)**: 10 (solo para validación con fuerza bruta), 20, 50, 100, 200.
- **Número de colores (k)**: 3 y 4.
- **Algoritmos evaluados**: Fuerza Bruta (FB), Greedy (GR), Greedy + Búsqueda Local (GR+BL), Greedy + Búsqueda Tabú (GR+BT), Programación Dinámica para Árboles (DP).
- **Métricas**: Tiempo de ejecución (segundos), costo total de la solución, número de conflictos (debe ser 0), y factor de aproximación empírico (costo/algoritmo óptimo o mejor conocido).

Nota sobre "conflictos". Se entiende por *conflicto* una violación de la restricción de interferencia: una arista $\{u,v\} \in E$ tal que $f(u) = f(v)$. En este trabajo, todas las soluciones reportadas por los algoritmos finales se verifican con `verificador.py`, por lo que el número de conflictos reportado debe ser 0; cualquier solución con conflictos se considera inválida y no se incluye como solución factible.
Promedios experimentales. Las métricas de costo y tiempo se reportan como promedios sobre 50 instancias independientes para cada configuración (n,k) y para cada tipo de grafo, evitando mezclar topologías distintas.

Los experimentos se ejecutaron en un equipo con procesador Intel Core i7-10750H, 16 GB de RAM, bajo Windows 11.

5.3. Resultados y Análisis

5.3.1. Validación con Fuerza Bruta

Para instancias muy pequeñas ($n = 10, k = 3$), el algoritmo de fuerza bruta (FB) nos dio el costo óptimo de referencia. En promedio sobre 50 instancias, Greedy (GR) obtuvo un costo un 11 % superior al óptimo, mientras que Greedy+Búsqueda Local (GR+BL) lo redujo a solo un 2 % por encima, confirmando la capacidad de mejora de la búsqueda local incluso partiendo de una solución greedy.

Cuadro 1: Comparación de algoritmos para $n = 50, k = 4$ (grafos Erdős-Rényi). Promedio sobre 50 instancias.

Algoritmo	**Tiempo (s)**	**Costo Prom.**	**Conflictos**	**Factor Aprox.**
Greedy (GR)	0.05	1450	0	1.11
GR + Búsq. Local	1.23	1320	0	1.01
GR + Búsq. Tabú	2.15	**1305**	0	**1.00**

5.3.2. Comparación General de Algoritmos

La siguiente tabla resume los resultados para $n = 50, k = 4$ en grafos Erdős-Rényi.

El greedy es notablemente rápido, pero la búsqueda local logra una mejora de costo del 9 % (de 1450 a 1320) a un costo computacional moderado. La búsqueda tabú, aunque más lenta, encuentra soluciones ligeramente mejores, situándose como el mejor heurístico en estas pruebas.

5.3.3. Escalabilidad

Los tiempos de ejecución crecen de manera aproximadamente lineal con n, como se esperaba. Tanto GR como GR+BL exhiben este comportamiento, siendo GR unas 20-50 veces más rápido. Para $n = 200$, GR+BL toma alrededor de 6 segundos, un tiempo aún muy razonable para un problema de planificación.

5.3.4. Impacto de la Topología del Grafo

Los resultados varían significativamente según el tipo de grafo.

Cuadro 2: Resultados por tipo de grafo ($n = 50, k = 4$). Costo promedio.

Tipo de Grafo	**GR**	**GR+BL**	**Óptimo (DP)**	**Factor GR+BL**
Erdős-Rényi	1450	1320	N/A	1.01
Árboles	850	820	815	**1.006**
Geométrico	1600	1480	N/A	1.02

En **árboles**, nuestras heurísticas están muy cerca del óptimo (0.6 % de diferencia en promedio), lo que sugiere que el greedy por grado funciona excepcionalmente bien en esta estructura. Para grafos **geométricos**, más densos localmente, los costos son mayores y la mejora relativa de GR+BL sobre GR es más acentuada (cerca del 7.5 %). Estos detalles son cruciales: un planificador de redes, conociendo la distribución geográfica de sus torres (que se asemeja a un grafo geométrico), puede priorizar el uso de técnicas de mejora como la búsqueda local.

5.3.5. Convergencia de la Búsqueda Local

En nuestras ejecuciones, la búsqueda local simple convergió típicamente en 50-100 iteraciones (un pase completo por todos los vértices sin mejora). La búsqueda tabú, al permitir movimientos no mejorantes, exploró más del espacio, a menudo encontrando mejoras tras la convergencia del ascenso de colina, justificando su mayor tiempo de ejecución.

6. Resultados y Conclusiones

6.1. Conclusiones Principales

Este proyecto ha permitido abordar en profundidad el Problema de Asignación de Frecuencias con Costos Mínimos (AFCM), llegando a las siguientes conclusiones:

1. **NP-Completitud:** Se demostró formalmente, mediante reducción desde k-COLORING, que AFCM es NP-completo. Esto cierra la puerta a la existencia de algoritmos exactos y polinomiales para el caso general y justifica el uso de enfoques aproximados.

2. **Soluciones Exáctas para Casos Especiales:** Para la familia de grafos árbol, se diseñó e implementó un algoritmo de programación dinámica con complejidad $O(n \cdot k^2)$ que encuentra la solución óptima. Este resultado es valioso para subredes con topología jerárquica.

3. **Estrategias Eficientes para el Caso General:** Se implementaron y compararon varias heurísticas. El **algoritmo greedy secuencial** es extremadamente rápido pero de calidad variable. Las técnicas de **búsqueda local**, en particular la **búsqueda tabú**, logran mejoras sustanciales (5-15 % en costo) con un aumento de tiempo computacional aceptable.

4. **Validación Experimental Rigurosa:** El análisis experimental, realizado sobre tres tipos de grafos y con un número significativo de instancias, confirma los hallazgos teóricos y ofrece guías prácticas:

   - La combinación *Greedy + Búsqueda Local* ofrece el mejor balance tiempo-calidad para la mayoría de los escenarios.
   - En árboles, las heurísticas simples son casi óptimas.
   - La topología del grafo (modelada por el tipo de grafo) influye decisivamente en la dificultad del problema y en el desempeño relativo de los algoritmos.

En mi opinión, un hallazgo particularmente relevante para la práctica es que, aunque el greedy tiene una cota de aproximación teórica pesimista ($O(\Delta)$), en la práctica su rendimiento es mucho mejor, especialmente cuando se complementa con búsqueda local. Esto es algo que he observado en otros problemas de optimización combinatoria: el análisis del peor caso a veces esconde un comportamiento promedio bastante más favorable.

6.2. Trabajo Futuro

Este proyecto abre varias líneas de investigación interesantes:

- **Otras Metaheurísticas:** Explorar el desempeño de algoritmos genéticos, recocido simulado o colonias de hormigas en este problema.

- **Cotas de Aproximación más Estrechas:** Investigar si para clases restringidas de grafos (planos, de grado acotado) se pueden diseñar algoritmos de aproximación con factor constante garantizado.

- **Modelos más Realistas:** Incorporar interferencia co-canal y adyacente, múltiples antenas por torre, o costos dinámicos que dependan del patrón de tráfico.

- **Optimización de Hiperparámetros:** Realizar un estudio sistemático para ajustar los parámetros de la búsqueda tabú (tamaño de lista, criterio de aspiración) en función del tipo de instancia.

6.3. Reflexión Final

El problema AFCM es un ejemplo paradigmático de cómo un desafío industrial concreto—optimizar costos en una red de telecomunicaciones—se transforma en un problema abstracto de teoría de grafos y optimización combinatoria, revelando su inherente complejidad computacional (NP-completitud). Sin embargo, esta complejidad no es una barrera infranqueable. Mediante una combinación inteligente de teoría (análisis de complejidad, diseño de algoritmos exactos para casos especiales) y práctica (implementación de heurísticas eficientes, evaluación experimental rigurosa), es posible desarrollar soluciones viables y de alta calidad.

Este ciclo de modelación, análisis teórico, diseño algorítmico y validación empírica constituye, a mi juicio, la esencia misma de la Ciencia de la Computación aplicada. El presente trabajo no solo aporta soluciones concretas a un problema específico, sino que también ilustra la metodología necesaria para enfrentar una amplia gama de problemas de optimización en ingeniería.

Referencias

[1] Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of Computer Computations* (pp. 85–103). Springer.

[2] Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A Guide to la Teoría de NP-Completeness*. W.H. Freeman.

[3] Erdős, P., & Rényi, A. (1959). On random graphs. *Publicationes Mathematicae*, 6, 290–297.

[4] Prüfer, H. (1918). Neuer Beweis eines Satzes über Permutationen. *Archiv der Mathematik und Physik*, 27, 742–744.

[5] Penrose, M. (2003). *Random Geometric Graphs*. Oxford University Press.

[6] Glover, F., & Laguna, M. (1997). *Tabu Search*. Kluwer Academic Publishers.

[7] Aarts, E., & Lenstra, J. K. (Eds.). (2003). *Local Search in Combinatorial Optimization*. Princeton University Press.

[8] Galinier, P., & Hertz, A. (2006). A survey of local search methods for graph coloring. *Computers & Operations Research*, 33(9), 2547–2562.

[9] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.

[10] Hale, W. K. (1980). Frequency assignment: Theory and applications. *Proceedings of the IEEE*, 68(12), 1497–1514.