Análisis Comparativo de Algoritmos para el Problema de Coloreado de Grafos de Costo Mínimo

Darío López Falcón, Diego M Viera Martínez, Pablo Gómez Vidal
Universidad de La Habana

Enero 2026

Resumen

El problema de asignación de frecuencias en redes de telecomunicaciones puede modelarse como un problema de coloreado de grafos donde asignar cada color tiene un costo. Este artículo presenta un estudio comparativo de cinco algoritmos para resolver el problema de Coloreado de Grafos de Costo Mínimo (MCGC): enfoques constructivos voraces (Greedy, Welsh-Powell, DSATUR) y metaheurísticas (Simulated Annealing y Tabu Repair). Se evalúa su rendimiento en términos de costo de la solución, tiempo de ejecución y escalabilidad sobre grafos aleatorios. Los resultados experimentales demuestran que, mientras los algoritmos voraces ofrecen tiempos casi lineales, las metaheurísticas proporcionan un equilibrio óptimo para instancias complejas, y el algoritmo exacto queda limitado a grafos pequeños ($N < 15$).

1. Introducción

El problema de coloreado de grafos (GCP) es uno de los problemas clásicos de la optimización combinatoria y la teoría de grafos. En su versión más básica, el objetivo es asignar un color a cada vértice del grafo de tal manera que dos vértices adyacentes no compartan el mismo color, utilizando la menor cantidad de colores posible (número cromático $\chi(G)$). Este problema es conocido por ser NP-Completo [5], lo que implica que, a menos que P=NP, no existe un algoritmo de tiempo polinomial capaz de resolverlo de manera exacta para cualquier instancia.

Una variante de gran interés práctico, y en la que se centra este artículo, es el Problema de Coloreado de Grafos de Costo Mínimo (MCGC). En el MCGC, cada color disponible tiene un costo asociado cuando se asigna a un vértice específico. El objetivo ya no es minimizar el número total de colores, sino minimizar la suma total de los costos de los colores asignados a todos los vértices del grafo, respetando las restricciones de adyacencia.

Esta variante modela situaciones del mundo real con mayor fidelidad que el GCP clásico. Un ejemplo paradigmático es la asignación de frecuencias en redes de telecomunicaciones móviles. En este escenario, los vértices representan torres de transmisión y los colores representan bandas de frecuencia. Las restricciones de adyacencia evitan interferencias entre torres cercanas, pero asignar una frecuencia específica a una torre puede tener costos variables debido a licencias, consumo energético del equipamiento o características de propagación de la señal en esa ubicación.

En este trabajo, exploramos y comparamos diferentes enfoques para resolver el MCGC, desde algoritmos exactos como Backtracking (limitados a instancias pequeñas) hasta heurísticas constructivas voraces y metaheurísticas avanzadas como Simulated Annealing y Tabu Repair, evaluando su eficacia en términos de calidad de la solución y tiempo de cómputo sobre grafos aleatorios.

2. Definición informal del Problema

A continuación se presenta la descripción del problema original sobre el cual está basado este trabajo.

En ConectaMax Telecom, nuestra misión es proporcionar una conectividad móvil ininterrumpida y de alta calidad a millones de usuarios. Para lograrlo, operamos una extensa y compleja red de torres de telefonía celular. La eficiencia y la calidad de nuestra red dependen críticamente de cómo gestionamos uno de nuestros recursos más valiosos: el espectro de radiofrecuencias.

Nos enfrentamos a un desafío operativo y financiero significativo en la asignación de frecuencias a nuestras torres. Disponemos de un conjunto limitado de frecuencias que podemos utilizar. La regla fundamental, dictada por la física y la regulación, es que dos torres que están geográficamente muy cerca una de la otra (y que, por lo tanto, podrían interferir entre sí) no pueden operar en la misma frecuencia. Esto es vital para evitar la degradación de la señal y

asegurar un servicio fiable.La complejidad adicional, y donde reside nuestro mayor reto, es que el costo de operar una torre con una frecuencia específica no es uniforme. Asignar una frecuencia particular a una torre determinada conlleva costos variables. Estos costos pueden deberse a múltiples factores:

1. **Equipamiento de la Torre:** Algunas torres tienen hardware más antiguo o especializado que es más eficiente con ciertas frecuencias, mientras que otras frecuencias podrían requerir adaptaciones o un mayor consumo energético en ese mismo equipo.
2. **Consumo Energético:** La eficiencia energética de la transmisión varía según la frecuencia y el tipo de equipo de la torre, impactando directamente nuestras facturas de electricidad.
3. **Regulaciones Locales y Licencias:** En ciertas áreas o para bandas de frecuencia específicas, pueden existir tarifas de licencia más elevadas o regulaciones que exigen configuraciones especiales, aumentando los costos operativos.

Nuestro objetivo principal es asignar una frecuencia a cada una de nuestras torres de tal manera que:

1. **Se eviten todas las interferencias:** Ninguna torre cercana a otra utilice la misma frecuencia.
2. **Se minimice el costo operativo total:** La suma de los costos individuales de asignar cada frecuencia a cada torre sea la más baja posible.

Una gestión subóptima de esta asignación no solo puede generar interferencias que afectan la calidad del servicio y la satisfacción del cliente, sino que también puede resultar en millones de dólares en costos operativos innecesarios.

2.1. Reducción formal del Problema

Usando la descripción informal previamente presentada, se definió el problema como un problema de Optimización Lineal Entera:

Dado un grafo no dirigido **G = (V, E)** donde **V** es el conjunto de vértices y **E** el conjunto de aristas:

Sea $\mathbf{C} = \{c_1, c_2, \ldots, c_k\}$ un conjunto de colores disponibles, y una función de costo **w: C x V** $\rightarrow$ **R**, donde **w(c_i, v)** representa el costo de asociarle el color c_i al vértice **v**.

Se definen las variables binarias de decisión:

$$x_{v,k} = \begin{cases} 1 & \text{si al vértice } v \text{ se le asigna el color } k \\ 0 & \text{en otro caso} \end{cases}$$

El Objetivo es:

$$\text{Minimizar } Z = \sum_{v \in V} \sum_{c \in C} x_{v,c} \cdot w(c, v)$$

Sujeto a:

$$x_{u,c} + x_{v,c} \leq 1, \quad \forall (u, v) \in E, \forall c \in C \tag{1}$$

(Restricción de adyacencia)

En dicha reducción cada nodo del grafo representa una torre del problema original, cada arista $\langle u, v \rangle$ del grafo representa que las torres u y v son cercanas, cada color representa una frecuencia y la lógica tras el cálculo del costo operativo de una torre con una frecuencia asignada se encapsuló dentro de la función w, además la restricción de adyacencia evita que dos torres cercanas tengan asociadas el mismo color.

3. Análisis de Complejidad Computacional

Presentamos una demostración formal de que el problema de Coloreado de Grafos de Costo Mínimo (MCGC) es NP-completo.

Dado que este es un problema de optimización, analizamos su **versión de decisión** para estudiar su complejidad computacional.
Se le llamó (MCGC-D) a la versión de decisión, y se definió de la siguiente manera:

Dado un grafo no dirigido **G = (V, E)** donde **V** es el conjunto de vértices y **E** el conjunto de aristas:

Sea $\mathbf{C} = \{c_1, c_2, \ldots, c_k\}$ un conjunto de colores disponibles, y una función de costo **w: C x V** $\rightarrow$ **R**, donde $w(c_i, v)$ representa el costo de asociarle el color c_i al vértice **v**.

Se define como coloración válida a una función $f : V \rightarrow C$ tal que $f(u) \neq f(v)$ para toda arista $(u, v) \in E$
Pregunta:
¿Existe una coloración válida f, tal que $\sum_{v \in V} w(v, f(v)) \leq B$?

3.0.1. Pertenecia a NP

Para demostrar que MCGC-D pertenece a la clase NP, basta con mostrar que una solución candidata puede ser verificada en tiempo polinomial.
Certificado
Una función de asignación $f : V \rightarrow C$.
Verificación

1. Para cada arista $(u, v) \in E$, verificar que $f(u) \neq f(v)$. Esto puede hacerse en tiempo $O(|E|)$.
2. Calcular el costo total $\sum_{v \in V} w(v, f(v))$ en tiempo $O(|V|)$.
3. Verificar que el costo total sea menor o igual a B es $O(1)$.

Dado que todos estos pasos se ejecutan en tiempo polinomial, concluimos que:

$$\text{MCGC-D} \in \text{NP}$$

3.0.2. Pertenencia a NP-Hard

Para demostrar que MCGC-D es NP-Hard, reducimos desde el problema **k-Coloring**, conocido por ser NP-completo para todo $k \geq 3$

Definición del problema K-Coloring :

Dado un grafo $G = (V, E)$ y un conjunto de k colores$\mathbf{C} = \{c_1, c_2, \ldots, c_k\}$, determinar si existe una coloración válida de G utilizando solamente colores de C.

Reducción

Sea una instancia arbitraria de k-Coloring dada por un grafo $G = (V, E)$ y un conjunto C de k colores.

Construimos una instancia de MCGC-D de la siguiente manera:

1. Utilizamos el mismo grafo $G = (V, E)$ y el mismo conjunto de colores $C = \{c_1, c_2, \ldots, c_k\}$. 3. Definimos la función de costo como:

$$w(c, v) = 0 \quad \forall v \in V, \forall c \in C$$

4. Definimos el presupuesto como:

$$B = 0$$

Esta construcción puede realizarse en tiempo polinomial.

Correctitud de la Reducción

- (**Si**) Si el grafo G es k-colorable, entonces existe una coloración válida que cumple las restricciones de adyacencia. Dado que todos los costos son cero, el costo total es $0 \leq B$, por lo que la instancia de MCGC-D responde **si**.
- (**Solo si**) Si existe una solución a la instancia de MCGC-D con costo total menor o igual a $B = 0$, entonces necesariamente se trata de una coloración válida del grafo G con k colores, lo que implica que G es k-colorable.

Por lo tanto, la instancia de k-Coloring tiene solución si y solo si la instancia construida de MCGC-D tiene solución.

Conclusión

Dado que:

- MCGC-D pertenece a NP, y
- MCGC-D es NP-Hard.

concluimos que:

$$\text{MCGC-D es NP-completo}$$

En consecuencia, el problema original de optimización de coloración con costos es **NP-Hard** y **NP-Completo**.

4. Estado del Arte

Debido a la naturaleza NP-Hard del problema MCGC, la literatura se ha centrado en dos vertientes principales: enfoques exactos para grafos pequeños y aproximaciones heurísticas/metaheurísticas para instancias de gran escala.

4.1. Enfoques Clásicos y Heurísticos

Los algoritmos voraces como **Greedy Secuencial** y **DSATUR** (Degree of Saturation) siguen siendo referencias fundamentales por su eficiencia $O(n^2)$. DSATUR, propuesto originalmente por Brelaz [1], destaca por minimizar dinámicamente el espacio de búsqueda. Investigaciones recientes continúan hibridando estos métodos constructivos con búsquedas locales para mejorar la calidad de la solución inicial [6].

4.2. Metaheurísticas

Para superar los óptimos locales, se han aplicado extensamente técnicas como **Simulated Annealing** (Recocido Simulado), **Tabu Search** y **Algoritmos Genéticos**. Estudios comparativos (e.g., Johnson et al. [4]) han demostrado que Simulated Annealing, cuando se calibra adecuadamente con un esquema de enfriamiento lento, ofrece una de las mejores relaciones costo-calidad para grafos generales. Hertz y de Werra [3] fueron pioneros en aplicar Tabu Search a este dominio.

4.3. Avances Recientes: Graph Neural Networks

En los últimos años (2024-2025), el estado del arte ha comenzado a integrar técnicas de aprendizaje profundo. Modelos como **MCGNN** (Minimal Cost Graph Neural Network) [7] utilizan redes neuronales de grafos para aprender características topológicas complejas y guiar el proceso de coloreado. Aunque estos métodos prometen resultados superiores en instancias muy complejas, requieren un entrenamiento costoso, por lo que las heurísticas tradicionales siguen siendo preferidas en entornos donde la simplicidad y la ausencia de entrenamiento previo son críticas.

4.4. Algoritmos de Tiempo Casi Lineal

Avances teóricos recientes han logrado algoritmos de coloreado (específicamente coloración de aristas) en tiempo casi lineal ($O(m \log m)$) [2], rompiendo barreras de eficiencia que habían permanecido estáticas durante décadas. Aunque estos avances son teóricos, impulsan el desarrollo de nuevas heurísticas prácticas más rápidas.

5. Soluciones Propuestas

Para este trabajo se presenta la implementación de cinco algoritmos polinomiales que se emplean para la respuesta del problema de MCGC en vez de emplear algoritmos exactos como backtracking.

5.1. Greedy Básico

Es el enfoque constructivo más fundamental. Itera sobre los vértices V en una secuencia predeterminada $\sigma = (v_1, v_2, \ldots, v_n)$. Para cada vértice v_i, selecciona el color factible de menor costo.

5.1.1. Estrategia

1. Iterar por cada nodo v en V.
2. Identificar los colores usados por los vecinos de v.
3. Asignar a v el color de menor costo que no esté en uso por sus vecinos.

5.1.2. Formalización

Sea $C_{disp} \subseteq C$ el conjunto de colores disponibles (tal que $|C| \geq \Delta(G)+1$ para asegurar factibilidad). Para un vértice v, definimos el conjunto de colores prohibidos por sus vecinos ya coloreados como $P(v) = \{c(u) \mid u \in N(v) \cap \{v_1, \ldots, v_{i-1}\}\}$. El algoritmo selecciona el color c^* tal que:

$$c^*(v) = \arg \min_{c \in C \setminus P(v)} \{w(c, v)\}$$

5.1.3. Argumento de Funcionamiento

Correctitud: Si el conjunto de colores C es suficientemente grande ($|C| > \Delta(G)$), el conjunto $C \setminus P(v)$ nunca será vacío, garantizando siempre una asignación válida (Teorema de Brooks). **Heurística de Costo:** Al minimizar el término $w(c, v)$ localmente en cada paso, el algoritmo realiza una búsqueda "Gradient Descent" en el espacio de construcción. Aunque esta estrategia miope no garantiza el óptimo global (ya que asignar un color barato ahora podría forzar colores muy caros a vecinos futuros), es efectiva en grafos esparcidos donde las restricciones de adyacencia no se propagan densamente. **Limitaciones:** Su principal debilidad es la sensibilidad extrema al orden σ. Un mal orden puede forzar el uso de colores de alto costo innecesariamente.

5.1.4. Análisis de Complejidad

- **Tiempo**: $O(V \times D_{max} \times K)$. Para cada uno de los V nodos, se inspeccionan sus vecinos (a lo sumo D_{max}) y se busca el mínimo en K colores.
- **Espacio**: $O(V + E)$ para almacenar el grafo y la asignación.

5.2. Welsh-Powell

Es una extensión del algoritmo voraz que mitiga la sensibilidad al orden σ mediante un preprocesamiento estático de los vértices. La intuición fundamental es que los nodos con mayor grado (más restricciones potenciales) son los más çríticosz deben colorearse cuando el sistema tiene la máxima entropía (mayor libertad de elección).

5.2.1. Estrategia y Formalización

1. Calcular el grado $grad(v)$ para todo $v \in V$.
2. Ordenar los vértices en una secuencia $\sigma_{WP} = (v_1, \ldots, v_n)$ tal que $grad(v_1) \geq grad(v_2) \geq \cdots \geq grad(v_n)$.
3. Aplicar el procedimiento Greedy Básico sobre σ_{WP}.

5.2.2. Justificación Formal

El **Principio de "Hardest-First"** sugiere que al colorear primero los nodos de alto grado, es más probable que tengan vecinos *aún no coloreados*, lo que significa que el conjunto de colores prohibidos $P(v)$ es pequeño o vacío. Esto permite asignar a los nodos más restrictivos sus colores óptimos (de menor costo). Los nodos de bajo grado, que se colorean al final, tienen más vecinos ya coloreados, pero al tener pocas aristas en total, es probable que aun existan colores baratos disponibles en su conjunto factible. Empíricamente, esto reduce la probabilidad de que el algoritmo se vea .ªcorralado"hacia colores de alto costo.

5.2.3. Análisis de Complejidad

- **Tiempo**: $O(V \log V + E)$. El ordenamiento domina con $V \log V$, y el coloreado toma tiempo proporcional a las aristas E, que en caso peor es V^2.
- **Espacio**: $O(V + E)$. Para el grafo y la lista ordenada.

5.3. DSATUR

DSATUR (Degree of Saturation) optimiza la heurística constructiva utilizando un ordenamiento **dinámico**. En lugar de una secuencia estática, el siguiente nodo a colorear se elige basándose en el estado actual de la solución parcial.

5.3.1. Definición Formal de Saturación

Para un vértice no coloreado v, definimos su **Grado de Saturación** $DSAT(v)$ como el número de colores *diferentes* utilizados en su vecindad coloreada:

$$DSAT(v) = |\{c(u) \mid u \in N(v) \wedge x_{u,\cdot} \text{ está definido}\}|$$

Intuitivamente, $DSAT(v)$ representa una cota inferior de cuántos colores "baratos"han sido prohibidos para v.

5.3.2. Estrategia

1. **Paso Inicial:** Elegir v con máximo grado estático.
2. **Selección:** Mientras existan vértices sin colorear:
   - Seleccionar $v^* = \arg\max_{v \in V_{uncolored}} \{DSAT(v)\}$.
   - Desempate: Usar el grado estático $grad(v)$.
3. **Asignación:** Asignar a v^* el color factible de mínimo costo.
4. **Actualización:** Actualizar los valores $DSAT$ de los vecinos de v^*.

5.3.3. Justificación de Efectividad

DSATUR implementa una estrategia de **Mínimo Margen de Error**. Al seleccionar el nodo con mayor saturación, estamos eligiendo el nodo que tiene *menos* colores disponibles. Posponer la decisión de este nodo aumentaría el riesgo de que sus opciones se reduzcan a cero (fallo) o solo queden opciones de costo extremo. Al atenderlo prioritariamente, maximizamos la probabilidad de encontrarle un color de costo razonable antes de que sea "demasiado tarde". En el contexto de MCGC, esto es crucial: evita que nodos muy conectados agoten todos los colores baratos (frecuencias bajas/eficientes) prematuramente.

5.3.4. Análisis de Complejidad

- **Tiempo**: $O(V^2)$ con una implementación ingenua (búsqueda lineal del máximo). Puede optimizarse a $O((V+E)\log V)$ utilizando montículos (Heaps) actualizables.
- **Espacio**: $O(V+E)$ para el grafo y estructuras auxiliares.

5.4. Simulated Annealing

Es una metaheurística probabilística inspirada en la termodinámica. Busca escapar de los "mínimos locales"(donde los algoritmos greedy se atascan) permitiendo movimientos "malos"(que aumentan el costo) temporalmente.

5.4.1. Estrategia

1. Comenzar con una solución inicial (ej. salida de DSATUR).
2. **Perturbación**: Elegir un nodo al azar y cambiar su color a otro válido al azar.
3. Calcular la diferencia de costo ΔE.
4. **Criterio de Aceptación:**
   - Si $\Delta E < 0$ (mejora), aceptar siempre.
   - Si $\Delta E > 0$ (empeora), aceptar con probabilidad $P = e^{-\Delta E/T}$.
5. **Enfriamiento**: Reducir la temperatura T gradualmente ($T = T \times \alpha$).

5.4.2. Análisis de Complejidad

- **Tiempo:**
  - Depende del número de iteraciones configurado ($Iter$).
  - Por iteración es muy rápido ($O(D_{max})$ para verificar validez y costo local).
  - Requiere comenzar con una solución factible previamente calculada $O(V^2)$ si se usa DSATOUR por ejemplo.
  - Total: $O(Iter \times D_{max} + V^2)$.
- **Espacio**: $O(V)$ para mantener la solución.

5.5. Tabu Repair

A diferencia de los anteriores que construyen una solución válida paso a paso, este enfoque comienza desde una solución **infactible en la mayoría de casos** (asignando a cada nodo su color más barato absoluto) y trata de reparar"los conflictos iterativamente, priorizando los nodos que presenten mas conflictos.

5.5.1. Estrategia

1. **Inicio:** Asignar a cada nodo el color de costo mínimo absoluto, ignorando conflictos.
2. **Ciclo de Reparación**
   - Identificar nodos en conflicto (vecinos con el mismo color).
   - Seleccionar al nodo que presenta más conflictos (En caso de empate emplear heuristicas de DSATUR o Welsh-Powell).
   - **Evaluar movimientos**: Cambiar el color de un nodo en conflicto.
   - **Función Objetivo**: Minimizar $F = (K \times \text{Conflictos}) + \text{Costo Total}$. ($K$ es muy grande para priorizar factibilidad).
   - **Lista Tabú**: Evitar revertir un cambio reciente para no caer en ciclos.

5.5.2. Análisis de Complejidad

- **Tiempo**: El algoritmo es iterativo y no garantiza convergencia en tiempo polinomial estricto sin un límite de iteraciones ($MaxIter$).
  - En cada iteración, identificar el nodo con máximo conflicto toma $O(V)$ inspeccionando la lista de conflictos.
  - Evaluar los movimientos para ese nodo implica probar hasta K colores, verificando conflictos con sus vecinos, lo que toma $O(D_{max} \times K)$.
  - Por tanto, la complejidad es $O(MaxIter \times (V + D_{max} \times K))$. Aunque cada paso es rápido, se requieren muchas iteraciones para escapar de mínimos locales profundos.
- **Espacio**: $O(V + E)$. Se necesita almacenar la matriz o lista de adyacencia para verificar conflictos rápidamente, el arreglo de asignaciones actuales y la lista Tabú (que suele ser de tamaño fijo o promocional a V).

6. Resultados y Discusión

Para comparar los algoritmos, se generaron grafos sintéticos con densidad $D = 0{,}5$ y tamaños $N = \{10, 20, 50, 100\}$, ejecutando 10 muestras para cada tamaño. Se midió el costo de la solución encontrada, la validez de la misma y el tiempo de ejecución en milisegundos.

Los resultados revelan patrones claros de rendimiento:

Instancias Pequeñas ($N = 10$) Como se observa en la Tabla 1 (y datos en `benchmark_exact.csv`), el algoritmo exacto **Backtracking** logra consistentemente los costos mínimos (ej. 1340.0 vs 1350.0 del Greedy en la muestra #1), demostrando que las heurísticas no siempre alcanzan el óptimo global. Sin embargo, el costo temporal es prohibitivo: Backtracking tarda entre 30,000 ms (30 segundos) y 116,000 ms (casi 2 minutos) para un solo grafo de 10 nodos. En contraste, los algoritmos Greedy y DSATUR resuelven estas instancias en menos de 0,1 ms. Esto confirma que los métodos exactos son inviables para cualquier aplicación de tiempo real o grafos mayores.

Escalabilidad y Metaheurísticas ($N = 20, 50, 100$) Para grafos medianos y grandes, la diferencia de tiempos se magnifica.

1. **Velocidad Extrema:** Los algoritmos constructivos (Basic Greedy, Welsh-Powell, DSATUR) mantienen tiempos de ejecución casi despreciables. Incluso para $N = 100$, completan la tarea en aproximadamente 1,5 ms (DSATUR) a 0,5 ms (Greedy).
2. **Calidad de Solución:**
   - **Simulated Annealing (SA)** y **DSATUR** suelen ofrecer los mejores costos. En las pruebas de $N = 50$ y $N = 100$, SA a menudo iguala o mejora ligeramente el costo de DSATUR, aunque con un tiempo de ejecución mayor (aprox. $15 - 20$ ms para $N = 100$).
   - **Tabu Repair** mostró un comportamiento menos estable en las pruebas realizadas, con tiempos de ejecución significativamente más altos ($200 - 300$ ms para $N = 100$) y costos a menudo superiores a SA o DSATUR. Esto sugiere que para el MCGC, comenzar con una solución factible y mejorarla (como hace SA) es más eficiente que reparar una infactible.

En resumen, para $N = 100$, un enfoque como DSATUR ofrece una solución de alta calidad en 1.5 ms, mientras que Simulated Annealing puede refinarla marginalmente invirtiendo 10 veces más tiempo (17 ms), lo cual sigue siendo muy rápido para la mayoría de las aplicaciones.

Se presentan a continuación las gráficas que ilustran estas comparativas:

Instance	Solver	Cost	Valid	Time(ms)
"Random(N=10,D=0.5,#1)"	BasicGreedy	1350.0	True	0.04
"Random(N=10,D=0.5,#1)"	Backtracking	1340.0	True	36593.52
"Random(N=10,D=0.5,#2)"	BasicGreedy	1340.0	True	0.02
"Random(N=10,D=0.5,#2)"	Backtracking	1330.0	True	37165.57
"Random(N=10,D=0.5,#3)"	BasicGreedy	1380.0	True	0.03
"Random(N=10,D=0.5,#3)"	Backtracking	1350.0	True	116174.77
"Random(N=10,D=0.5,#4)"	BasicGreedy	1400.0	True	0.04
"Random(N=10,D=0.5,#4)"	Backtracking	1385.0	True	32983.57
"Random(N=10,D=0.5,#5)"	BasicGreedy	1375.0	True	0.03
"Random(N=10,D=0.5,#5)"	Backtracking	1350.0	True	69297.09
"Random(N=10,D=0.5,#6)"	BasicGreedy	1400.0	True	0.02
"Random(N=10,D=0.5,#6)"	Backtracking	1370.0	True	29940.25
"Random(N=10,D=0.5,#7)"	BasicGreedy	1385.0	True	0.02
"Random(N=10,D=0.5,#7)"	Backtracking	1380.0	True	50220.84
"Random(N=10,D=0.5,#8)"	BasicGreedy	1420.0	True	0.02
"Random(N=10,D=0.5,#8)"	Backtracking	1415.0	True	75984.06
"Random(N=10,D=0.5,#9)"	BasicGreedy	1480.0	True	0.02
"Random(N=10,D=0.5,#9)"	Backtracking	1470.0	True	65715.76
"Random(N=10,D=0.5,#10)"	BasicGreedy	1370.0	True	0.02
"Random(N=10,D=0.5,#10)"	Backtracking	1360.0	True	46541.42

Figura 1: Comparativa de resultados entre Backtracking y Greedy Básico

Nota: Solo se probó el algoritmo de backtracking con grafos de hasta 10 nodos, ya que con dicho tamaño, para cada grafo el algoritmo tardaba en promedio 1 minuto en ejecutarse.

Instance	Solver	Cost	Valid	Time(ms)
"Random(N=10, D=0.5, #1)"	Basic Greedy	1375.0	True	0.05
"Random(N=10, D=0.5, #1)"	DSATUR	1385.0	True	0.1
"Random(N=10, D=0.5, #1)"	Welsh-Powell	1385.0	True	0.06
"Random(N=10, D=0.5, #1)"	Sim. Annealing	1370.0	True	12.39
"Random(N=10, D=0.5, #1)"	Tabu Repair	1360.0	True	0.66
"Random(N=10, D=0.5, #2)"	Basic Greedy	1430.0	True	0.03
"Random(N=10, D=0.5, #2)"	DSATUR	1450.0	True	0.09
"Random(N=10, D=0.5, #2)"	Welsh-Powell	1450.0	True	0.05
"Random(N=10, D=0.5, #2)"	Sim. Annealing	1425.0	True	13.39
"Random(N=10, D=0.5, #2)"	Tabu Repair	1420.0	True	0.75
"Random(N=10, D=0.5, #3)"	Basic Greedy	1465.0	True	0.02
"Random(N=10, D=0.5, #3)"	DSATUR	1455.0	True	0.09
"Random(N=10, D=0.5, #3)"	Welsh-Powell	1455.0	True	0.05
"Random(N=10, D=0.5, #3)"	Sim. Annealing	1445.0	True	12.27
"Random(N=10, D=0.5, #3)"	Tabu Repair	1450.0	True	0.96
"Random(N=10, D=0.5, #4)"	Basic Greedy	1450.0	True	0.03
"Random(N=10, D=0.5, #4)"	DSATUR	1455.0	True	0.07
"Random(N=10, D=0.5, #4)"	Welsh-Powell	1455.0	True	0.03
"Random(N=10, D=0.5, #4)"	Sim. Annealing	1435.0	True	13.05
"Random(N=10, D=0.5, #4)"	Tabu Repair	1445.0	True	0.64
"Random(N=10, D=0.5, #5)"	Basic Greedy	1475.0	True	0.03
"Random(N=10, D=0.5, #5)"	DSATUR	1475.0	True	0.07
"Random(N=10, D=0.5, #5)"	Welsh-Powell	1475.0	True	0.05
"Random(N=10, D=0.5, #5)"	Sim. Annealing	1465.0	True	12.8
"Random(N=10, D=0.5, #5)"	Tabu Repair	1475.0	True	0.64

Figura 2: Resultados de los cinco algoritmos para grafos de 10 nodos

Instance	Solver	Cost	Valid	Time(ms)
"Random(N=10, D=0.5, #6)"	Basic Greedy	1480.0	True	0.02
"Random(N=10, D=0.5, #6)"	DSATUR	1500.0	True	0.09
"Random(N=10, D=0.5, #6)"	Welsh-Powell	1500.0	True	0.06
"Random(N=10, D=0.5, #6)"	Sim. Annealing	1490.0	True	14.98
"Random(N=10, D=0.5, #6)"	Tabu Repair	1470.0	True	0.69
"Random(N=10, D=0.5, #7)"	Basic Greedy	1445.0	True	0.02
"Random(N=10, D=0.5, #7)"	DSATUR	1435.0	True	0.08
"Random(N=10, D=0.5, #7)"	Welsh-Powell	1435.0	True	0.08
"Random(N=10, D=0.5, #7)"	Sim. Annealing	1430.0	True	14.03
"Random(N=10, D=0.5, #7)"	Tabu Repair	1420.0	True	0.65
"Random(N=10, D=0.5, #8)"	Basic Greedy	1405.0	True	0.02
"Random(N=10, D=0.5, #8)"	DSATUR	1415.0	True	0.11
"Random(N=10, D=0.5, #8)"	Welsh-Powell	1420.0	True	0.03
"Random(N=10, D=0.5, #8)"	Sim. Annealing	1410.0	True	13.55
"Random(N=10, D=0.5, #8)"	Tabu Repair	1410.0	True	0.68
"Random(N=10, D=0.5, #9)"	Basic Greedy	1320.0	True	0.02
"Random(N=10, D=0.5, #9)"	DSATUR	1340.0	True	0.06
"Random(N=10, D=0.5, #9)"	Welsh-Powell	1340.0	True	0.05
"Random(N=10, D=0.5, #9)"	Sim. Annealing	1315.0	True	13.82
"Random(N=10, D=0.5, #9)"	Tabu Repair	1320.0	True	0.68
"Random(N=10, D=0.5, #10)"	Basic Greedy	1405.0	True	0.02
"Random(N=10, D=0.5, #10)"	DSATUR	1410.0	True	0.06
"Random(N=10, D=0.5, #10)"	Welsh-Powell	1410.0	True	0.03
"Random(N=10, D=0.5, #10)"	Sim. Annealing	1400.0	True	12.54
"Random(N=10, D=0.5, #10)"	Tabu Repair	1400.0	True	0.67

Figura 3: Resultados de los cinco algoritmos para grafos de 10 nodos

Instance	Solver	Cost	Valid	Time(ms)
"Random(N=20, D=0.5, #1)"	Basic Greedy	2915.0	True	0.07
"Random(N=20, D=0.5, #1)"	DSATUR	2865.0	True	0.18
"Random(N=20, D=0.5, #1)"	Welsh-Powell	2910.0	True	0.08
"Random(N=20, D=0.5, #1)"	Sim. Annealing	2865.0	True	12.96
"Random(N=20, D=0.5, #1)"	Tabu Repair	2850.0	True	7.69
"Random(N=20, D=0.5, #2)"	Basic Greedy	2930.0	True	0.06
"Random(N=20, D=0.5, #2)"	DSATUR	2920.0	True	0.17
"Random(N=20, D=0.5, #2)"	Welsh-Powell	2935.0	True	0.08
"Random(N=20, D=0.5, #2)"	Sim. Annealing	2910.0	True	13.37
"Random(N=20, D=0.5, #2)"	Tabu Repair	2890.0	True	8.67
"Random(N=20, D=0.5, #3)"	Basic Greedy	2865.0	True	0.05
"Random(N=20, D=0.5, #3)"	DSATUR	2875.0	True	0.16
"Random(N=20, D=0.5, #3)"	Welsh-Powell	2895.0	True	0.06
"Random(N=20, D=0.5, #3)"	Sim. Annealing	2845.0	True	12.08
"Random(N=20, D=0.5, #3)"	Tabu Repair	2875.0	True	7.83
"Random(N=20, D=0.5, #4)"	Basic Greedy	3045.0	True	0.05
"Random(N=20, D=0.5, #4)"	DSATUR	3040.0	True	0.13
"Random(N=20, D=0.5, #4)"	Welsh-Powell	3040.0	True	0.08
"Random(N=20, D=0.5, #4)"	Sim. Annealing	3030.0	True	12.38
"Random(N=20, D=0.5, #4)"	Tabu Repair	3045.0	True	8.68
"Random(N=20, D=0.5, #5)"	Basic Greedy	2915.0	True	0.05
"Random(N=20, D=0.5, #5)"	DSATUR	2895.0	True	0.15
"Random(N=20, D=0.5, #5)"	Welsh-Powell	2920.0	True	0.07
"Random(N=20, D=0.5, #5)"	Sim. Annealing	2895.0	True	12.99
"Random(N=20, D=0.5, #5)"	Tabu Repair	2905.0	True	7.87

Figura 4: Resultados de los cinco algoritmos para grafos de 20 nodos

Instance	Solver	Cost	Valid	Time(ms)
"Random(N=20, D=0.5, #6)"	Basic Greedy	2835.0	True	0.05
"Random(N=20, D=0.5, #6)"	DSATUR	2855.0	True	0.13
"Random(N=20, D=0.5, #6)"	Welsh-Powell	2865.0	True	0.07
"Random(N=20, D=0.5, #6)"	Sim. Annealing	2850.0	True	12.67
"Random(N=20, D=0.5, #6)"	Tabu Repair	2825.0	True	8.15
"Random(N=20, D=0.5, #7)"	Basic Greedy	2835.0	True	0.05
"Random(N=20, D=0.5, #7)"	DSATUR	2845.0	True	0.13
"Random(N=20, D=0.5, #7)"	Welsh-Powell	2860.0	True	0.08
"Random(N=20, D=0.5, #7)"	Sim. Annealing	2840.0	True	13.02
"Random(N=20, D=0.5, #7)"	Tabu Repair	2815.0	True	7.93
"Random(N=20, D=0.5, #8)"	Basic Greedy	2980.0	True	0.05
"Random(N=20, D=0.5, #8)"	DSATUR	2935.0	True	0.14
"Random(N=20, D=0.5, #8)"	Welsh-Powell	2950.0	True	0.1
"Random(N=20, D=0.5, #8)"	Sim. Annealing	2935.0	True	12.66
"Random(N=20, D=0.5, #8)"	Tabu Repair	2915.0	True	8.38
"Random(N=20, D=0.5, #9)"	Basic Greedy	2840.0	True	0.06
"Random(N=20, D=0.5, #9)"	DSATUR	2840.0	True	0.16
"Random(N=20, D=0.5, #9)"	Welsh-Powell	2845.0	True	0.07
"Random(N=20, D=0.5, #9)"	Sim. Annealing	2825.0	True	12.6
"Random(N=20, D=0.5, #9)"	Tabu Repair	2805.0	True	7.38
"Random(N=20, D=0.5, #10)"	Basic Greedy	2845.0	True	0.06
"Random(N=20, D=0.5, #10)"	DSATUR	2905.0	True	0.16
"Random(N=20, D=0.5, #10)"	Welsh-Powell	2885.0	True	0.08
"Random(N=20, D=0.5, #10)"	Sim. Annealing	2895.0	True	13.18
"Random(N=20, D=0.5, #10)"	Tabu Repair	2870.0	True	7.61

Figura 5: Resultados de los cinco algoritmos para grafos de 20 nodos

Instance	Solver	Cost	Valid	Time(ms)
"Random(N=50, D=0.5, #1)"	Basic Greedy	8130.0	True	0.17
"Random(N=50, D=0.5, #1)"	DSATUR	8055.0	True	0.67
"Random(N=50, D=0.5, #1)"	Welsh-Powell	8095.0	True	0.22
"Random(N=50, D=0.5, #1)"	Sim. Annealing	8055.0	True	15.41
"Random(N=50, D=0.5, #1)"	Tabu Repair	8230.0	True	70.51
"Random(N=50, D=0.5, #2)"	Basic Greedy	7705.0	True	0.15
"Random(N=50, D=0.5, #2)"	DSATUR	7640.0	True	0.51
"Random(N=50, D=0.5, #2)"	Welsh-Powell	7775.0	True	0.2
"Random(N=50, D=0.5, #2)"	Sim. Annealing	7640.0	True	13.08
"Random(N=50, D=0.5, #2)"	Tabu Repair	8005.0	True	67.82
"Random(N=50, D=0.5, #3)"	Basic Greedy	7715.0	True	0.15
"Random(N=50, D=0.5, #3)"	DSATUR	7700.0	True	0.5
"Random(N=50, D=0.5, #3)"	Welsh-Powell	7700.0	True	0.18
"Random(N=50, D=0.5, #3)"	Sim. Annealing	7700.0	True	13.23
"Random(N=50, D=0.5, #3)"	Tabu Repair	7995.0	True	66.72
"Random(N=50, D=0.5, #4)"	Basic Greedy	7885.0	True	0.14
"Random(N=50, D=0.5, #4)"	DSATUR	7730.0	True	0.53
"Random(N=50, D=0.5, #4)"	Welsh-Powell	7865.0	True	0.18
"Random(N=50, D=0.5, #4)"	Sim. Annealing	7730.0	True	12.88
"Random(N=50, D=0.5, #4)"	Tabu Repair	8010.0	True	66.26
"Random(N=50, D=0.5, #5)"	Basic Greedy	7725.0	True	0.15
"Random(N=50, D=0.5, #5)"	DSATUR	7845.0	True	0.5
"Random(N=50, D=0.5, #5)"	Welsh-Powell	7840.0	True	0.18
"Random(N=50, D=0.5, #5)"	Sim. Annealing	7845.0	True	14.0
"Random(N=50, D=0.5, #5)"	Tabu Repair	7975.0	True	67.63

Figura 6: Resultados de los cinco algoritmos para grafos de 50 nodos

Instance	Solver	Cost	Valid	Time(ms)
"Random(N=50, D=0.5, #6)"	Basic Greedy	8010.0	True	0.15
"Random(N=50, D=0.5, #6)"	DSATUR	7905.0	True	0.51
"Random(N=50, D=0.5, #6)"	Welsh-Powell	8020.0	True	0.2
"Random(N=50, D=0.5, #6)"	Sim. Annealing	7905.0	True	13.58
"Random(N=50, D=0.5, #6)"	Tabu Repair	8175.0	True	66.41
"Random(N=50, D=0.5, #7)"	Basic Greedy	7725.0	True	0.15
"Random(N=50, D=0.5, #7)"	DSATUR	7485.0	True	0.54
"Random(N=50, D=0.5, #7)"	Welsh-Powell	7675.0	True	0.2
"Random(N=50, D=0.5, #7)"	Sim. Annealing	7485.0	True	13.81
"Random(N=50, D=0.5, #7)"	Tabu Repair	7820.0	True	72.65
"Random(N=50, D=0.5, #8)"	Basic Greedy	7805.0	True	0.22
"Random(N=50, D=0.5, #8)"	DSATUR	7790.0	True	0.65
"Random(N=50, D=0.5, #8)"	Welsh-Powell	7930.0	True	0.25
"Random(N=50, D=0.5, #8)"	Sim. Annealing	7790.0	True	15.22
"Random(N=50, D=0.5, #8)"	Tabu Repair	8010.0	True	73.62
"Random(N=50, D=0.5, #9)"	Basic Greedy	8085.0	True	0.16
"Random(N=50, D=0.5, #9)"	DSATUR	8120.0	True	0.66
"Random(N=50, D=0.5, #9)"	Welsh-Powell	8135.0	True	0.22
"Random(N=50, D=0.5, #9)"	Sim. Annealing	8120.0	True	15.33
"Random(N=50, D=0.5, #9)"	Tabu Repair	8285.0	True	77.12
"Random(N=50, D=0.5, #10)"	Basic Greedy	8080.0	True	0.19
"Random(N=50, D=0.5, #10)"	DSATUR	8055.0	True	0.65
"Random(N=50, D=0.5, #10)"	Welsh-Powell	8075.0	True	0.24
"Random(N=50, D=0.5, #10)"	Sim. Annealing	8055.0	True	14.87
"Random(N=50, D=0.5, #10)"	Tabu Repair	8295.0	True	69.33

Figura 7: Resultados de los cinco algoritmos para grafos de 50 nodos

Instance	Solver	Cost	Valid	Time(ms)
"Random(N=100, D=0.5, #1)"	Basic Greedy	17775.0	True	0.48
"Random(N=100, D=0.5, #1)"	DSATUR	17690.0	True	1.95
"Random(N=100, D=0.5, #1)"	Welsh-Powell	17600.0	True	0.53
"Random(N=100, D=0.5, #1)"	Sim. Annealing	17690.0	True	17.48
"Random(N=100, D=0.5, #1)"	Tabu Repair	18895.0	True	260.74
"Random(N=100, D=0.5, #2)"	Basic Greedy	17665.0	True	0.49
"Random(N=100, D=0.5, #2)"	DSATUR	17390.0	True	1.53
"Random(N=100, D=0.5, #2)"	Welsh-Powell	17830.0	True	0.51
"Random(N=100, D=0.5, #2)"	Sim. Annealing	17390.0	True	18.13
"Random(N=100, D=0.5, #2)"	Tabu Repair	19180.0	True	271.5
"Random(N=100, D=0.5, #3)"	Basic Greedy	18035.0	True	0.43
"Random(N=100, D=0.5, #3)"	DSATUR	17830.0	True	1.52
"Random(N=100, D=0.5, #3)"	Welsh-Powell	17925.0	True	0.47
"Random(N=100, D=0.5, #3)"	Sim. Annealing	17830.0	True	26.38
"Random(N=100, D=0.5, #3)"	Tabu Repair	19720.0	True	290.39
"Random(N=100, D=0.5, #4)"	Basic Greedy	17650.0	True	0.44
"Random(N=100, D=0.5, #4)"	DSATUR	17475.0	True	1.61
"Random(N=100, D=0.5, #4)"	Welsh-Powell	17930.0	True	0.51
"Random(N=100, D=0.5, #4)"	Sim. Annealing	17475.0	True	26.05
"Random(N=100, D=0.5, #4)"	Tabu Repair	19270.0	True	272.62
"Random(N=100, D=0.5, #5)"	Basic Greedy	17475.0	True	0.4
"Random(N=100, D=0.5, #5)"	DSATUR	17190.0	True	1.51
"Random(N=100, D=0.5, #5)"	Welsh-Powell	17500.0	True	0.46
"Random(N=100, D=0.5, #5)"	Sim. Annealing	17190.0	True	16.45
"Random(N=100, D=0.5, #5)"	Tabu Repair	19010.0	True	268.01

Figura 8: Resultados de los cinco algoritmos para grafos de 100 nodos

Instance	Solver	Cost	Valid	Time(ms)
"Random(N=100, D=0.5, #6)"	Basic Greedy	17735.0	True	0.42
"Random(N=100, D=0.5, #6)"	DSATUR	17415.0	True	1.56
"Random(N=100, D=0.5, #6)"	Welsh-Powell	17860.0	True	0.49
"Random(N=100, D=0.5, #6)"	Sim. Annealing	17415.0	True	16.91
"Random(N=100, D=0.5, #6)"	Tabu Repair	19635.0	True	266.94
"Random(N=100, D=0.5, #7)"	Basic Greedy	17680.0	True	0.42
"Random(N=100, D=0.5, #7)"	DSATUR	17300.0	True	1.49
"Random(N=100, D=0.5, #7)"	Welsh-Powell	17425.0	True	0.47
"Random(N=100, D=0.5, #7)"	Sim. Annealing	17300.0	True	16.73
"Random(N=100, D=0.5, #7)"	Tabu Repair	19080.0	True	273.28
"Random(N=100, D=0.5, #8)"	Basic Greedy	16880.0	True	0.39
"Random(N=100, D=0.5, #8)"	DSATUR	16980.0	True	1.49
"Random(N=100, D=0.5, #8)"	Welsh-Powell	17490.0	True	0.46
"Random(N=100, D=0.5, #8)"	Sim. Annealing	16980.0	True	17.33
"Random(N=100, D=0.5, #8)"	Tabu Repair	18195.0	True	267.72
"Random(N=100, D=0.5, #9)"	Basic Greedy	17780.0	True	0.45
"Random(N=100, D=0.5, #9)"	DSATUR	17575.0	True	1.5
"Random(N=100, D=0.5, #9)"	Welsh-Powell	18105.0	True	0.48
"Random(N=100, D=0.5, #9)"	Sim. Annealing	17575.0	True	16.66
"Random(N=100, D=0.5, #9)"	Tabu Repair	19390.0	True	273.12
"Random(N=100, D=0.5, #10)"	Basic Greedy	17715.0	True	0.44
"Random(N=100, D=0.5, #10)"	DSATUR	17305.0	True	1.6
"Random(N=100, D=0.5, #10)"	Welsh-Powell	17930.0	True	0.49
"Random(N=100, D=0.5, #10)"	Sim. Annealing	17305.0	True	16.53
"Random(N=100, D=0.5, #10)"	Tabu Repair	19090.0	True	276.67

Figura 9: Resultados de los cinco algoritmos para grafos de 100 nodos

7. Conclusión y Trabajo Futuro

El Problema de Coloreado de Grafos de Costo Mínimo presenta un desafío complejo donde el equilibrio entre tiempo de cómputo y calidad de la solución es crítico. Nuestros experimentos permiten concluir que:

1. Para aplicaciones de **tiempo real** o en sistemas embebidos con recursos limitados, **DSATUR** es la opción superior. Su complejidad polinomial baja y su capacidad para tomar decisiones informadas sobre la marcha le permiten superar a los enfoques greedy simples sin el costo computacional de las metaheurísticas.
2. Cuando la calidad de la solución (minimización de costos operativos) es la prioridad absoluta y el tiempo de cálculo permite unos milisegundos extra, **Simulated Annealing** es una herramienta robusta para escapar de los óptimos locales en los que pueden caer los métodos constructivos.
3. Los métodos exactos como Backtracking son puramente académicos para este problema, siendo inútiles para grafos con $N > 15$ debido a su crecimiento factorial.

Como trabajo futuro, proponemos:

- Investigar algoritmos híbridos que utilicen DSATUR para generar la población inicial de un **Algoritmo Genético**, combinando la buena calidad inicial con la capacidad de exploración global de la evolución.
- Explorar la paralelización de la evaluación de vecinos en Tabu Search para reducir su tiempo de ejecución en instancias densas.
- Incorporar restricciones del mundo real más complejas, como interferencias no binarias (gradientes de señal) o costos dinámicos dependientes del tiempo, para acercar el modelo aún más a las necesidades de la industria de telecomunicaciones.

Referencias

[1] Daniel Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.

[2] A. Costa and M. S. Near-linear time algorithms for edge coloring. *Journal of Algorithms*, 2024.

[3] Alain Hertz and Dominique De Werra. Using tabu search techniques for graph coloring. *Computing*, 39:345–351, 1987.

[4] David S Johnson, Cecilia R Aragon, Lyle A McGeoch, and Catherine Schevon. Optimization by simulated annealing: an experimental evaluation; part i, graph partitioning. *Operations research*, 37(6):865–892, 1989.

[5] Richard M Karp. Reducibility among combinatorial problems. *Complexity of computer computations*, pages 85–103, 1972.

[6] Dominic JA Welsh and Martin B Powell. An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal*, 10(1):85–86, 1967.

[7] Y. Zhou and et al. Minimal cost graph neural network for graph coloring. *IEEE Transactions on Neural Networks and Learning Systems*, 2024. (Preprint).