Proyecto DAA: Problema de la Empresa Telefónica - Partición Cromática de Costo Mínimo (MCCPP)

Yesenia Valdés Rodríguez (C411)
Laura Martir Beltrán (C411)
Adrián Hernández Castellanos (C412)

Índice

1. Formalización del Problema

1.1. Problema: La empresa Telefónica

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:

- **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.
- **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.
- **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.

1.2. Contexto del Problema: La Asignación de Frecuencias en ConectaMax Telecom

El problema presentado por la empresa ConectaMax Telecom, centrado en la asignación óptima de frecuencias a sus torres de telefonía celular, se traduce directamente en un problema de optimización combinatoria conocido como el **Problema de Partición Cromática de Costo Mínimo (Minimum Cost Chromatic Partition Problem, MCCPP)**.

La estructura de la red telefónica y sus restricciones operativas se modelan de la siguiente manera:

- **Grafo de Interferencia** $G = (V, E)$: Las Torres de Telefonía Celular se representan como los vértices (V) del grafo. Las aristas (E) representan las proximidades geográficas y el potencial de interferencia, donde una arista $\{u, v\} \in E$ indica que la Torre u y la Torre v no pueden operar en la misma frecuencia.

- **Etiquetas y Frecuencias** F: El conjunto limitado de frecuencias disponibles se convierte en el conjunto finito de etiquetas (F) (o colores).

- **Restricción Operacional (Coloración Propia)**: La regla de "evitar todas las interferencias" se traduce en la restricción de coloración propia, exigiendo que dos vértices adyacentes no posean la misma etiqueta:
$$\phi(u) \neq \phi(v)$$

- **Costo Operativo** C: La variabilidad en el costo de operar una torre con una frecuencia específica (debido a equipamiento, energía, licencias) se captura mediante la matriz de costos C, donde $c_{v,f}$ es el costo de asignar la frecuencia f a la torre v.

- **Objetivo (Minimización del Costo)**: El objetivo de minimizar el costo operativo total se convierte en:
$$\text{mín} \sum_{v \in V} c_{v,\phi(v)}$$

El MCCPP es, por lo tanto, la formalización precisa del desafío de la empresa telefónica, ya que busca una partición válida de los vértices (torres) en clases de color (frecuencias) que minimice el costo total asociado a dicha partición.

1.3. Definición Formal

Nombre del Problema: Problema de Partición Cromática de Costo Mínimo (MCCPP).
Instancia de Entrada: Una tupla $\langle G, F, C \rangle$, donde:

- $G = (V, E)$: Un grafo no dirigido, con $|V| = n$ y E el conjunto de aristas.
- F: Conjunto finito de etiquetas (colores/frecuencias), $|F| = k$.
- C: Matriz de costos donde $c_{v,f}$ es el costo de asignar la etiqueta f al vértice v.

Función Objetivo: Encontrar una función de asignación de etiquetas $\phi : V \to F$ que minimice el costo total de la asignación.

Restricción de Factibilidad:

- Para toda arista $\{u, v\} \in E$, se debe cumplir que $\phi(u) \neq \phi(v)$.

1.4. Modelado de Programación Entera Lineal (ILP 0-1)

El MCCPP se formaliza rigurosamente mediante el siguiente modelo de Programación Entera Lineal con variables binarias, que servirá de base para la demostración de complejidad y para la implementación de un algoritmo exacto (fuerza bruta/Branch-and-Bound en instancias pequeñas).

Parámetros:

- V: Conjunto de vértices.
- F: Conjunto de etiquetas disponibles.
- $c_{v,f}$: Costo de asignar la etiqueta $f \in F$ al vértice $v \in V$.

Variables de Decisión:

- $x_{v,f} \in \{0, 1\}$: Variable binaria. $x_{v,f} = 1$ si el vértice v es asignado a la etiqueta f; $x_{v,f} = 0$ en caso contrario.

Función Objetivo (Minimización del Costo Total):

$$\text{mín} \sum_{v \in V} \sum_{f \in F} c_{v,f} \cdot x_{v,f}$$

Restricciones de Factibilidad:

Asignación Única por Vértice: Cada vértice debe ser asignado a exactamente una etiqueta.

$$\sum_{f \in F} x_{v,f} = 1 \quad \forall v \in V$$

Restricción de No Conflicto (Coloración Propia): Dos vértices adyacentes no pueden recibir la misma etiqueta.

$$x_{u,f} + x_{v,f} \leq 1 \quad \forall \{u, v\} \in E, \ \forall f \in F$$

Integridad:

$$x_{v,f} \in \{0,1\} \quad \forall v \in V,\ \forall f \in F$$

2. Análisis de Complejidad Computacional

2.1. Clase de Complejidad

El Problema de Partición Cromática de Costo Mínimo (MCCPP) es un problema NP-Hard en su versión de optimización. Dado que su restricción de factibilidad es idéntica a la del problema k-Colorability (reconocido como NP-Completo), la búsqueda de la solución óptima de costo mínimo es, por lo menos, tan difícil como la búsqueda de cualquier coloración válida.

2.2. Planteamiento del Problema de k-Colorabilidad

El **problema de k-Colorabilidad** (también conocido como k-Coloración Propia) plantea la siguiente cuestión fundamental: dado un grafo no dirigido $G = (V,E)$ y un entero positivo k, ¿es posible asignar colores a los vértices de G utilizando a lo sumo k colores diferentes, de tal forma que ningún par de vértices adyacentes compartan el mismo color?

Este problema representa uno de los problemas clásicos de NP-Completitud y tiene aplicaciones en áreas como *scheduling*, asignación de registros en compiladores, y problemas de asignación de frecuencias en redes.

2.3. Demostración de NP-Hardness mediante Reducción Polinomial

Se demuestra que el MCCPP es NP-Hard mediante una reducción en tiempo polinomial del problema de decisión k-Colorability (una instancia canónica de NP-Completo) al MCCPP.

Problema Fuente (NP-Completo): k-Colorability.

Instancia: $\langle G,k\rangle$, donde $G = (V,E)$ es un grafo y k es un entero positivo.

Pregunta: ¿Existe una coloración propia de G utilizando a lo sumo k colores?

Reducción Polinomial $\mathscr{R}$:

Dada una instancia $\langle G,k\rangle$ del k-Colorability, construimos la siguiente instancia $\langle G',F,C\rangle$ del MCCPP:

- **Grafo G':** Mantenemos el mismo grafo: $G' = G$.
- **Conjunto de Etiquetas F:** Definimos el conjunto de etiquetas como $F = \{f_1, f_2, \ldots, f_k\}$, con $|F| = k$.

- **Matriz de Costos C:** Definimos un costo uniforme y nulo:

$$c_{v,f} = 0 \quad \forall v \in V, \forall f \in F$$

- **Umbral de Decisión $Z_{\text{máx}}$:** Establecemos el objetivo de optimización: mínZ. La pregunta de decisión asociada al MCCPP será si el costo mínimo Z^* es ≤ 0.

Conclusión de la Reducción:

- Si G es k-coloreable, existe una asignación $\phi : V \to F$ que satisface la restricción de coloración propia. Como todos los costos son cero ($c_{v,f} = 0$), el costo total de esta asignación es $Z = \sum c_{v,\phi(v)} = 0$. Por lo tanto, el costo mínimo óptimo Z^* es 0, y $Z^* \leq 0$.
- Si G no es k-coloreable, no existe ninguna asignación válida $\phi : V \to F$ que cumpla la restricción. Por lo tanto, no existe una solución factible, y no podemos alcanzar el costo $Z \leq 0$.

Dado que se ha reducido el k-Colorability al MCCPP en tiempo polinomial (la construcción solo requiere asignar $k \cdot n$ costos a cero), y la respuesta a la instancia del MCCPP determina la respuesta a la instancia del k-Colorability, el MCCPP (en su versión de optimización) es NP-Hard.

2.4. Jerarquía de Aproximabilidad Polinomial: PTAS, APX, APX-Hardness, APX-Complete y log-APX

Dada la **NP-Hardness** del Problema de Partición Cromática de Costo Mínimo (MCCPP), el análisis de su **aproximabilidad** es crucial. Este análisis establece los límites teóricos sobre cuán cerca de la solución óptima Z^* podemos esperar llegar en tiempo polinomial. Para ello, es indispensable definir la jerarquía de las clases de complejidad de aproximación.

- **PTAS (Polynomial Time Approximation Scheme)**: La clase **PTAS** incluye los problemas de optimización que admiten un esquema de aproximación en tiempo polinomial. Esto significa que, para cualquier constante $\varepsilon > 0$ arbitrariamente pequeña, existe un algoritmo en tiempo polinomial $O(n^{f(1/\varepsilon)})$ que produce una solución Z_{Alg} tal que $Z_{Alg} \leq (1+\varepsilon)Z^*$ (para problemas de minimización). La complejidad temporal puede crecer exponencialmente con $1/\varepsilon$, pero es polinomial en el tamaño de la entrada n.
- **APX (Approximable)**: La clase **APX** (Approximable) contiene todos los problemas NPO (NP Optimization) que admiten un algoritmo de aproximación de factor constante c en tiempo polinomial, donde $c \geq 1$ es independiente del tamaño de la instancia n. Formalmente, $Z_{Alg} \leq c \cdot Z^*$. La clase **PTAS** es un subconjunto estricto de **APX**, asumiendo que $P \neq NP$. Pertenecer a APX implica que la calidad de la solución está acotada por una constante.

- **APX-Hard**: Un problema de optimización es **APX-Hard** si todo problema en **APX** puede ser reducido a él mediante una reducción que preserva la aproximación (típicamente una L-Reducción o PTAS-Reducción). La consecuencia fundamental de la APX-Hardness es que, si $P \neq NP$, el problema **no admite un PTAS**. La demostración de APX-Hardness del MCCPP (Sección 2.5) establece precisamente esta limitación.

- **APX-Complete**: Un problema es **APX-Complete** si es **APX-Hard** y, además, pertenece a la clase **APX**. Esto implica que existe un algoritmo de aproximación de factor constante c, pero que no existe un esquema de aproximación arbitrariamente cercano al óptimo (PTAS), salvo que $P = NP$.

- **log-APX**: Esta clase es relevante para problemas cuya garantía de aproximación es logarítmica. Un problema se clasifica en **log-APX** si admite un algoritmo de aproximación con un factor de rendimiento R acotado por $O(\ln n)$, donde n es el tamaño de la entrada. La clasificación en **log**-APX tiene una implicación más restrictiva que la APX-Hardness: los problemas en **log**-APX (y que no están en APX) **no admiten un algoritmo de aproximación de factor constante** c, a menos que se cumplan condiciones de complejidad teórica muy improbables.

2.5. Demostración de APX-Hardness del MCCPP

Para demostrar que un problema es APX-Hard, se requiere establecer una reducción que preserva la aproximación (típicamente una L-Reducción) desde un problema que ya se sabe es APX-Hard. Esta demostración refuta la posibilidad de esquemas de aproximación arbitrariamente cercanos al óptimo (PTAS), a menos que $P = NP$.

Por lo tanto, se establecerá una L-Reducción robusta desde el Minimum Vertex Cover (MVC).

2.5.1. Problema de Origen APX-Complete: Minimum Vertex Cover (MVC)

El problema de Minimum Vertex Cover (MVC) es un problema de optimización de minimización: dada una instancia de grafo $G = (V, E)$, se busca el subconjunto de vértices $V' \subseteq V$ de tamaño mínimo tal que cada arista en E tenga al menos un extremo en V'.

El MVC es un problema fundamental en complejidad y se sabe que es **APX-Complete** [1]. Esto implica dos hechos cruciales:

1. **APX-Hardness**: MVC no admite un PTAS a menos que $P = NP$ [2].

2. **Pertenencia a APX**: Existe un algoritmo de aproximación en tiempo polinomial con un factor de aproximación constante (una 2-aproximación basada en un algoritmo voraz clásico).

La demostración formal de que MVC es APX-Complete, confirmando su 2-aproximabilidad y su inaproximabilidad constante (cota inferior de 1,3606 si $P \neq NP$), se puede encontrar, por ejemplo, en [3], así como en desarrollos posteriores sobre la brecha de aproximación de Vertex Cover.

El MVC está intrínsecamente ligado al Maximum Independent Set (MIS). Un conjunto de vértices V' es una cubierta de vértices si y solo si su complemento $V \setminus V'$ es un conjunto independiente. Por lo tanto, minimizar $|V'|$ es equivalente a maximizar $|V \setminus V'|$, es decir, maximizar el tamaño del Conjunto Independiente. La dificultad de aproximar MIS se traslada directamente a la dificultad de aproximar MVC.

La reducción $\mathscr{R}$ transforma una instancia de MVC $\langle G \rangle$ en una instancia de MCCPP $\langle G', F, C \rangle$:

- **Grafo** G': Se mantiene el grafo original, $G' = G = (V, E)$.
- **Conjunto de Etiquetas** F: Se define un conjunto de etiquetas $F = \{f_1, f_2, \ldots, f_n\}$, donde $n = |V|$. Este conjunto es suficiente para cualquier coloración.
- **Matriz de Costos** C: Se definen costos binarios de la siguiente manera:
  - **Costo Cero (Color Preferido)**: $c_{v,f_1} = 0$, para todo vértice $v \in V$.
  - **Costo Unitario (Otros Colores)**: $c_{v,f} = 1$, para todo $v \in V$ y $f \in \{f_2, \ldots, f_n\}$.

El objetivo del MCCPP en esta instancia es encontrar una coloración propia $\phi : V \to F$ que minimice el costo total $Z_{MCCPP} = \sum_{v \in V} c_{v,\phi(v)}$.

Equivalencia de la Optimalidad:

Para minimizar Z_{MCCPP}, se debe maximizar el número de vértices asignados al color f_1 (costo 0). Sea I_1 el conjunto de vértices coloreados con f_1. Para que la coloración sea válida, I_1 debe ser un conjunto independiente.

El costo óptimo Z^*_{MCCPP} se logra cuando I_1 es el Máximo Conjunto Independiente ($I_{\text{máx}}$), ya que todos los demás colores ($f_2, \ldots, f_n$) tienen costo 1.

$$Z^*_{MCCPP} = 0 \cdot |I_{\text{máx}}| + 1 \cdot (|V| - |I_{\text{máx}}|) = |V| - |I_{\text{máx}}|$$

Dado que el tamaño del Minimum Vertex Cover Z^*_{MVC} es igual a $|V| - |I_{\text{máx}}|$, se establece la relación:

$$Z^*_{MCCPP} = Z^*_{MVC}$$

Preservación de la Aproximación:

Si existiera un algoritmo de aproximación A_{MCCPP} para el MCCPP con factor de aproximación constante α, es decir, $Z^{Alg}_{MCCPP} \leq \alpha \cdot Z^{*}_{MCCPP}$, este algoritmo podría usarse para resolver MVC con el mismo factor:

$$Z^{Alg}_{MVC} = Z^{Alg}_{MCCPP} \leq \alpha \cdot Z^{*}_{MCCPP} = \alpha \cdot Z^{*}_{MVC}$$

Dado que el MVC es APX-Hard, la existencia de una L-Reducción con preservación lineal del valor óptimo demuestra que el MCCPP es, a su vez, APX-Hard.

2.5.2. Implicaciones Teóricas: Exclusión del PTAS y APX-Completeness

La demostración de que el MCCPP es APX-Hard tiene consecuencias directas sobre la existencia de esquemas de aproximación.

2.5.3. Imposibilidad de PTAS

La APX-Hardness establece que, si $P \neq NP$, el MCCPP no admite un PTAS. Esta limitación persiste incluso en el caso especial de costos binarios, tal como se demostró en la reducción a MVC.

2.5.4. Análisis Riguroso de Pertenencia a APX (MCCPP)

Un problema es APX-Complete si es APX-Hard y también pertenece a la clase APX (es decir, admite un algoritmo de aproximación de factor constante c).

Aunque el MCCPP es APX-Hard, su pertenencia a APX no está universalmente establecida para grafos generales.

Como la pertenencia de un problema APX-Hard a la clase APX requiere demostrar la existencia de un algoritmo de aproximación con un factor de rendimiento constante $c \geq 1$ en tiempo polinomial, para el MCCPP esta demostración se considera altamente improbable para grafos generales.

Modelado Formal como Cobertura de Conjuntos Ponderada (WSC)

El MCCPP se puede modelar como el **Problema de Cobertura de Conjuntos Ponderada** (Weighted Set Cover, WSC), un problema fundamental de optimización ampliamente estudiado tanto desde el punto de vista algorítmico como de inaproximabilidad [4], [5].

- **Definición Formal de WSC**: Dada una instancia $\langle U, \mathscr{S}, w \rangle$, donde U es el universo de elementos a cubrir ($|U| = n$), $\mathscr{S} = \{S_1, \ldots, S_m\}$ es una familia de subconjuntos de U, y $w : \mathscr{S} \to \mathbb{R}^+$ es la función de peso (costo) de cada subconjunto S_i. El objetivo es encontrar una subcolección $\mathscr{C} \subseteq \mathscr{S}$ tal que $\bigcup_{S_i \in \mathscr{C}} S_i = U$, minimizando el costo total $\sum_{S_i \in \mathscr{C}} w(S_i)$.

Mapeo MCCPP a WSC:

Una instancia $\langle G, F, C \rangle$ del MCCPP se transforma en una instancia $\langle U, \mathscr{S}, w \rangle$ del WSC:

1. **Universo (U)**: El conjunto de vértices V del grafo G. El objetivo es "cubrir" todos los vértices, es decir, asignarles un color [4], [5].

2. **Conjuntos Candidatos ($\mathscr{S}$)**: Cada conjunto $S \in \mathscr{S}$ es un par $\langle I, f \rangle$, donde $I \subseteq V$ es un *Conjunto Independiente* (IS) válido en G y $f \in F$ es una frecuencia (color) disponible [4], [5].

3. **Costo del Conjunto** $w(S)$: El costo de elegir el conjunto $S = \langle I, f \rangle$ es el costo total de asignar la frecuencia f a todos los vértices en I:

$$w(S) = \sum_{v \in I} c_{v,f}$$

Una solución válida del WSC (una subcolección de conjuntos independientes que cubre V) corresponde directamente a una partición cromática válida de G con el mismo costo total.

Demostración de Inaproximabilidad Constante (MCCPP $\notin$ APX)

El mejor algoritmo de aproximación conocido para el WSC es el algoritmo **Greedy**, que alcanza una cota asintótica óptima en términos del factor logarítmico $\Theta(\ln n)$ tanto para Set Cover como para su versión ponderada [4], [5].

Sea Z^*_{WSC} el costo óptimo del WSC y Z^{Alg}_{WSC} el costo de la solución obtenida por el algoritmo voraz.

- **Factor de Aproximación Logarítmico**: El algoritmo Greedy garantiza un factor de aproximación R que está acotado superiormente por el número armónico $H(|U|)$, donde $|U| = |V| = n$:

$$R = \frac{Z^{Alg}_{WSC}}{Z^*_{WSC}} \leq H(|U|) = O(\ln n)$$

- **Límite Inferior de Inaproximabilidad**: Se ha demostrado rigurosamente que el WSC (y por extensión el Set Cover estándar) no admite un algoritmo de aproximación de factor constante c. Específicamente, a menos que $NP \subseteq DTIME(n^{\log \log n})$, no existe un algoritmo que garantice un factor de aproximación mejor que $(1-\varepsilon)\ln n$, para cualquier $\varepsilon > 0$ [4], [5].

Dado que la mejor aproximación garantizada para el MCCPP es logarítmica $O(\ln |V|)$, y existe un límite inferior que excluye factores constantes, se concluye formalmente que el MCCPP **no pertenece** a la clase APX, y por lo tanto, no es APX-Complete. En términos de la jerarquía de aproximación, el MCCPP se clasifica en la clase **log**-APX.

2.6. Análisis de Complejidad Parametrizada

La **Complejidad Parametrizada** ofrece un marco teórico para analizar problemas NP-Hard en función de parámetros específicos que pueden influir en la complejidad computacional. En este contexto, se examina la dificultad del MCCPP cuando se parametriza por el número de frecuencias $k = |F|$, por el treewidth tw del grafo G y otros enfoques.

FPT (Fixed-Parameter Tractable): Un problema parametrizado $\langle I, k\rangle$ es FPT si existe un algoritmo que resuelve cualquier instancia en tiempo $O(f(k) \cdot |I|^c)$, donde f es una función computable que depende únicamente del parámetro k, y c es una constante independiente de k y $|I|$. Esto implica que la complejidad exponencial está confinada al parámetro k, permitiendo soluciones eficientes para valores pequeños de k.

2.6.1. Definición Formal de W-Hardness

La **Complejidad Parametrizada** clasifica los problemas NP-Hard basándose en cómo el parámetro k afecta la complejidad exponencial. Un problema $\langle I, k\rangle$ se clasifica en la jerarquía **W** si no es FPT (asumiendo que **FPT** $\neq$ **W**).

La clase $\mathbf{W}[\mathbf{1}]$ se define mediante problemas que son equivalentes bajo FPT-reducciones al problema *Weighted Weft-1-Depth-d SAT* o, más comúnmente, al problema canónico **k**-Clique.

Definición de W-Hardness: Un problema parametrizado P es $\mathbf{W}[\mathbf{1}]$-Hard si todo problema en $\mathbf{W}[\mathbf{1}]$ puede ser reducido a P mediante una **FPT-reducción**. Una FPT-reducción es una transformación de una instancia $\langle I, k\rangle$ de P_1 a una instancia $\langle I', k'\rangle$ de P_2, computable en tiempo $O(f(k) \cdot |I|^c)$, donde el nuevo parámetro k' depende únicamente de k ($k' \leq g(k)$).

La consecuencia fundamental de la $\mathbf{W}[\mathbf{1}]$-Hardness es que, si se cree que la hipótesis de **FPT** $\neq$ $\mathbf{W}[\mathbf{1}]$ es verdadera, el problema no admite un algoritmo de tiempo $O(f(k) \cdot n^c)$, lo que descarta la kernelización con un tamaño que dependa únicamente de k.

Problema k-Coloring es W-Hard

La dificultad de la coloración se traslada directamente al reino de la complejidad parametrizada. Entonces se tiene que el problema **k**-Coloring, parametrizado por el número de colores k, es $\mathbf{W}[\mathbf{1}]$-Hard, tal como se establece en la literatura clásica de complejidad parametrizada y resultados específicos sobre problemas coloridos parametrizados por treewidth [6].

Fundamentación Académica

Este resultado, que establece que la coloración es paramétricamente difícil, se basa en reducciones desde problemas centrales de la jerarquía W, como k-Clique o su variante Multicolored Cli-

que. La demostración completa se encuentra rigurosamente detallada en trabajos de Flum y Grohe sobre problemas coloridos parametrizados por treewidth y en manuales estándar de complejidad parametrizada [6], [7].

2.6.2. Demostración de que el MCCPP es W[1]-Hard cuando se parametriza por $|F|$

El resultado de la W-Hardness del k-Coloring tiene implicaciones directas para el MCCPP, ya que la restricción de factibilidad del MCCPP es idéntica a la del k-Coloring.

Planteamiento

El Problema de Partición Cromática de Costo Mínimo (MCCPP), parametrizado por el número de frecuencias $k = |F|$, es $\mathbf{W}[\mathbf{1}]$-Hard.

Demostración

La demostración se logra mediante la **FPT-Reducción trivial** del k-Coloring al MCCPP:

1. Se parte de una instancia del k-Coloring $\langle G, k \rangle$.
2. Se construye la instancia del MCCPP $\langle G, F, C \rangle$ con $|F| = k$, estableciendo todos los costos $c_{v,f} = 0$.
3. El tiempo de esta construcción es $\mathbf{O}(|\mathbf{V}| \cdot \mathbf{k})$, que satisface la condición FPT (polinomial en $|V|$, pero la dependencia exponencial es solo en k).
4. El parámetro se preserva: $k' = k$.

Dado que la existencia de una solución de costo cero en el MCCPP es equivalente a la existencia de una k-coloración válida, y la reducción es una FPT-reducción, se concluye que **MCCPP** es $\mathbf{W}[\mathbf{1}]$-Hard cuando se parametriza por el número de colores k.

2.6.3. Demostración de que el MCCPP es W[1]-Hard cuando se parametriza por Treewidth

Estado de la Cuestión:

Existe literatura más antigua (<2000) donde se afirma que el MCCPP es FPT por tw. Sin embargo, en investigaciones más recientes (2017) se refuta lo afirmado en la literatura inicial sobre el problema: se ha demostrado formalmente mediante una reducción parametrizada rigurosa que el Problema de Partición Cromática de Costo Mínimo (MCCPP), también conocido como Weighted Coloring, es **W-hard** cuando se parametriza por la **treewidth** del grafo, incluso en el caso restringido de bosques (donde la treewidth es exactamente 1) [8].

Equivalencia Formal entre MCCPP y Weighted Coloring

Replanteemos el problema del MCCPP y el Weighted Coloring para establecer su equivalencia formal.

Problema 1: Minimum Cost Chromatic Partition Problem (MCCPP):

El Problema de Partición Cromática de Costo Mínimo se define formalmente de la siguiente manera:

Instancia: Un grafo no dirigido $G = (V, E)$ y una matriz de costos C donde $c_{v,f}$ es el costo de asignar la etiqueta (frecuencia/color) f al vértice v. Un conjunto finito F de etiquetas disponibles con $|F| = k$.

Objetivo: Encontrar una función de asignación $\phi : V \to F$ (una coloración propia, es decir, $\phi(u) \neq \phi(v)$ para toda arista $\{u, v\} \in E$) que minimice el costo total:

$$Z_{MCCPP} = \min_{\phi \text{ coloración propia}} \sum_{v \in V} c_{v,\phi(v)}$$

Problema 2: Weighted Coloring (Weighted Chromatic Number):

El problema de Weighted Coloring, introducido formalmente por Guan y Zhu, se define como sigue [9]:

Instancia: Un grafo no dirigido $G = (V, E)$ y una función de pesos $w : V \to \mathbb{R}^+$ que asigna un peso positivo a cada vértice.

Definiciones de Colores y Pesos:

Una coloración propia c de G es una partición $c = (S_i)_{i \in [1,k]}$ del conjunto V en k conjuntos independientes (también llamados estables) $S_1, S_2, \ldots, S_k$.

Dado que cada conjunto S_i es un conjunto independiente de vértices del mismo color, el *peso* del color S_i se define como:

$$w(i) = \max_{v \in S_i} w(v)$$

es decir, el peso máximo de los vértices en ese color.

El *peso total* de una coloración c es:

$$w(c) = \sum_{i=1}^{k} w(i) = \sum_{i=1}^{k} \max_{v \in S_i} w(v)$$

Objetivo: Calcular la *weighted chromatic number* $\sigma(G, w)$, definida como el peso mínimo entre todas las coloraciones propias válidas de G:

$$\sigma(G, w) = \min_{c \text{ coloración propia}} w(c)$$

Una coloración c tal que $w(c) = \sigma(G, w)$ se llama coloración óptima ponderada.

Relación entre MCCPP y Weighted Coloring

El Weighted Coloring es un caso especial del MCCPP, no son exactamente equivalentes. Es importante establecer esta relación con precisión:

Weighted Coloring:

- Cada vértice v tiene un peso fijo $w(v) \in \mathbb{R}^+$
- El costo de un color i es $w(i) = \max_{v \in S_i} w(v)$
- Objetivo: minimizar $\sum_{i=1}^{k} w(i) = \sum_{i=1}^{k} \max_{v \in S_i} w(v)$

MCCPP:

- Cada par (vértice v, frecuencia f) tiene un costo independiente $c_{v,f}$
- El costo total es $\sum_{v \in V} c_{v,\phi(v)}$
- Más general: permite costos arbitrarios por par (torre, frecuencia)

Relación formal: Weighted Coloring $\subseteq$ MCCPP. Se puede modelar Weighted Coloring como un caso especial del MCCPP estableciendo $c_{v,f} = w(v)$ para todo f, de modo que el costo de cada vértice es independiente del color asignado. En este caso especial, minimizar $\sum_v c_{v,\phi(v)}$ bajo la estructura de conjuntos independientes reproduce el problema de Weighted Coloring.

Sin embargo, el MCCPP permite estructuras de costo más complejas donde la misma torre puede tener costos muy diferentes según la frecuencia asignada, reflejando factores como:

- Eficiencia energética variable según la banda de frecuencia
- Costos de licencias específicos por frecuencia y ubicación
- Compatibilidad del equipamiento existente con ciertas frecuencias

Implicación para W[1]-hardness:
Los resultados de W[1]-hardness de Weighted Coloring [8] se transfieren al MCCPP porque:

1. Si un caso especial (Weighted Coloring) es W[1]-hard
2. Entonces el problema general (MCCPP) también es W[1]-hard
3. La reducción se hereda: cualquier instancia de Weighted Coloring puede expresarse como instancia del MCCPP

Por lo tanto, las demostraciones de W-hardness para Weighted Coloring parametrizado por treewidth o número de colores establecen automáticamente que el MCCPP más general también es W-hard bajo esos mismos parámetros.

Referencia Académica:

Esta relación entre ambos problemas está implícita en los trabajos de Guan y Zhu sobre coloración ponderada [9], y los resultados de complejidad parametrizada se estudian específicamente para Weighted Coloring en [8], [10], aplicándose al MCCPP por contención.

Independent Set es W-Completo

Antes de proceder a la demostración de $\mathbf{W}[\mathbf{1}]$-hardness del MCCPP mediante reducción desde Independent Set, es esencial establecer que Independent Set mismo es un problema canónico W-completo en complejidad parametrizada.

Definición del Problema Independent Set

Instancia: Un grafo no dirigido $G = (V, E)$ y un entero positivo k (parámetro).

Pregunta: ¿Existe un conjunto $S \subseteq V$ de vértices con $|S| \geq k$ tal que ningún par de vértices en S es adyacente? Es decir, ¿existe un conjunto independiente de tamaño al menos k?

Planteamiento: Independent Set es W-Completo

El problema Independent Set parametrizado por el tamaño de la solución k es **W-completo**. Esta es una de las conclusiones fundamentales de la teoría de complejidad parametrizada, establecida desde los trabajos fundacionales de Downey y Fellows y sistematizada en manuales modernos [7], [11].

Implicaciones

La W-completeness de Independent Set significa que dos cosas son ciertas:

1. **Independent Set pertenece a W:** Existe un certificador eficiente (en términos de complejidad parametrizada) que puede verificar en tiempo $f(k) \cdot n^c$ si un conjunto S de tamaño k es un conjunto independiente válido. Esta verificación es trivial: simplemente revisar si existen aristas entre los vértices de S, lo cual toma $O(k^2)$ tiempo.

2. **Todo problema en W puede ser reducido a Independent Set mediante una FPT-reducción:** La demostración se basa en mostrar que la definición de W mediante máquinas de Turing ponderadas puede ser capturada por Independent Set. Específicamente, cualquier problema

definible como "¿existe una solución de tamaño exactamente k que satisfaga la propiedad $P(x,k)$?" (donde P es verificable en tiempo FPT) puede ser reducido a Independent Set.

Referencia Académica

Estos resultados fundamentales están exhaustivamente cubiertos en [7], [11], donde se expone la jerarquía W y se demuestra rigurosamente que Independent Set parametrizado por k es W-completo.

Demostración de W[1]-Hardness del MCCPP vía Weighted Coloring

Teorema Principal

El Problema de Partición Cromática de Costo Mínimo (MCCPP), cuando se parametriza por la **treewidth** del grafo (o más generalmente, por el tamaño del mayor componente conexo del grafo), es **W-hard**.

Demostración de W[1]-Hardness por Reducción desde Independent Set

La demostración se logra mediante una **FPT-reducción parametrizada** desde el problema canónico **W[1]**-hard de *Independent Set* hacia Weighted Coloring y, por equivalencia, hacia MCCPP [8].

Definición de Problemas:

- **Independent Set (IS):** Dado un grafo $G = (V,E)$ y un entero k, ¿existe un conjunto $S \subseteq V$ con $|S| \geq k$ tal que ningún par de vértices en S es adyacente?
- **Weighted Coloring (WC):** Dado un grafo ponderado (G,w) con función de pesos $w : V \to \mathbb{R}^+$, donde el peso de un color S_i se define como $w(i) = \max_{v \in S_i} w(v)$, ¿existe una coloración propia $c = (S_i)_{i=1}^k$ tal que $w(c) = \sum_{i=1}^k w(i) \leq M$ para un umbral M?

Construcción de la Reducción:

La reducción construye a partir de una instancia (G,k) de Independent Set una instancia ponderada (G',w) de Weighted Coloring, introduciendo gadgets especializados:

1. **Árboles Binomiales Ponderados** B_i: Para cada $i \in [0,4k+3]$, se definen recursivamente árboles binomiales con pesos específicos $w_i^0 = 1/2^i + j\varepsilon$, donde ε es un parámetro pequeño seleccionado cuidadosamente. Estos árboles fuerzan que en cualquier coloración de peso $\leq M$, todos los vértices de un árbol binomial reciban el mismo color de forma determinista.

2. **Árboles Auxiliares** A_i^j: Para cada índice $i \in [0, 4k-1]$ y $j \in [0, n]$, se construyen árboles auxiliares que contienen copias de árboles binomiales. Su función es actuar como verificadores de que los vértices originales se han seleccionado de forma consistente a lo largo de diferentes copias.

3. **Gadget AND (R_0-AND)**: Un circuito lógico ponderado que implementa la operación lógica AND entre dos entradas, garantizando que si ambas entradas reciben un color específico R_0, la salida debe también recibir R_0.

4. **Árboles de Vértices** T_i^j: Para cada $(i, j) \in [0, k-1] \times [0, n-1]$, se construye un árbol que codifica si el vértice $\beta^{-1}(j)$ (donde β es una biyección $V \to [0, n-1]$) pertenece o no al conjunto independiente.

Parámetro Umbral M:
Se define $M = k(n-1)\varepsilon + \sum_{i=0}^{4k+3} \frac{1}{2^i}$ con $0 < \varepsilon < \frac{1}{nk2^{4k+3}}$.
Esta selección garantiza que:

- $M < 2$, proporcionando un umbral bien definido.
- Cada vez que se selecciona un vértice para el conjunto independiente (coloreando su raíz con R_0), se incurre en un costo adicional de $(n-1)\varepsilon$.
- Solo es posible seleccionar exactamente k vértices sin exceder el presupuesto M.

Lema de Correspondencia:

Sea (T, w) un bosque ponderado que contiene, para cada $(i, j) \in [0, k-1] \times [0, n-1]$, la estructura T_i^j como subárbol. Sea c una coloración de (T, w) con $w(c) \leq M$. Entonces existen índices $(j_i)_{i \in [0,k-1]} \in [0, n-1]^k$ tales que para cada raíz u de T_j^i:

- Si $j = j_i$ para algún $i \in [0, k-1]$, entonces $c(u) = R_0$.
- Si $j \neq j_i$ para todo $i \in [0, k-1]$, entonces $c(u) = R_1$.

Prueba de Equivalencia:

La FPT-reducción satisface: existe un conjunto independiente de tamaño exactamente k en G si y solo si existe una coloración (G', w) con $w(c) \leq M$.

- **(Adelante, $\Rightarrow$)**: Si Z es un conjunto independiente de tamaño k en G, se puede construir una coloración válida asignando cada vértice $v \in Z$ al color R_0 (costo 0 en las estructuras correspondientes) en los árboles T_i^j donde $j = \beta(v)$. Para aristas $\{v_1, v_2\} \in E$ con $v_1 \in Z, v_2 \notin Z$, los gadgets AND se pueden satisfacer con el color R_1 en la salida, manteniendo $w(c) = M$.

- **(Atrás, $\Leftarrow$)**: Si existe una coloración con $w(c) \leq M$, por el Lema anterior, existen índices (j_i) tales que exactamente k vértices reciben color R_0. Definiendo $Z = \{\beta^{-1}(j_i) : i \in [0, k-1]\}$, se verifica que no hay arista entre elementos de Z. Si existiera una arista $\{v_1, v_2\}$ con $v_1, v_2 \in Z$, entonces los gadgets AND en $H_{\{v_1,v_2\},i_1,i_2}$ forzarían que su raíz se coloreara con R_0, lo que contradice el árbol binomial B_{4k} conectado a ella.

Complejidad de la Reducción:
La construcción requiere:

- Tiempo: $f(k) \cdot n^{O(1)}$ donde $f(k) = 2^{O(k)}$ es una función computable de k.
- El parámetro se preserva: $k' = k$ (el parámetro de la nueva instancia depende solo del parámetro de la instancia original).
- El tamaño de cada componente conexo de (G', w) está acotado por $O(k)$.

Conclusión

El Problema de Partición Cromática de Costo Mínimo (MCCPP), cuando se parametriza por la **treewidth** del grafo (o por el tamaño del mayor componente conexo), es **W-hard**. Por lo tanto, a menos que **FPT** $=$ **W**[**1**], **no existe un algoritmo FPT para el MCCPP parametrizado por treewidth**. Este resultado es válido incluso en la clase altamente restringida de bosques (treewidth = 1), lo que hace aún más fuerte la imposibilidad [8].[1]

Corolario sobre la Optimización Observada en Bounded Treewidth:

Guan y Zhu observaron que en grafos de treewidth acotada t, se podía resolver $\sigma(G, w; r)$ en tiempo $n^{O(r)} \cdot r^{O(t)}$ mediante programación dinámica estándar. Sin embargo, este tiempo no es FPT porque la dependencia en n es polinomial de grado potencialmente alto, y más crucialmente, porque el parámetro r (número de colores) está acoplado de forma multiplicativa con el tamaño de la entrada. La demostración de W-hardness implica que no puede haber mejora fundamental en esta complejidad (bajo conjeturas estándar de complejidad parametrizada) [8], [9].[1]

Referencia Académica

Los detalles completos de la reducción y de los gadgets utilizados para demostrar la W-hardness de Weighted Coloring (y, por equivalencia, de MCCPP) en bosques pueden encontrarse en [8].

2.6.4. Búsqueda de Parámetros Alternativos FPT para el MCCPP

Dado que se ha demostrado que el MCCPP es W-hard con respecto a las frecuencias f (equivalente a número de colores r) y treewidth, es natural investigar si existen otros parámetros para

los cuales el problema sea FPT.

Panorama de Parámetros Estudiados:

Los siguientes parámetros han sido analizados en la literatura sobre problemas de coloración ponderada y problemas relacionados:

1. **Vertex Cover:** Este parámetro mide la mínima cantidad de vértices cuya eliminación convierte al grafo en una unión disjunta de conjuntos independientes. Aunque para coloración simple (k-Coloring) existen algoritmos FPT parametrizados por vertex cover, para MCCPP la literatura académica no reporta resultados positivos.

2. **Modular Decomposition Width:** Más restrictivo que treewidth, pero tampoco se ha demostrado FPT para MCCPP.

3. **Clique-Width:** Para coloración estándar se conocen limitaciones, pero MCCPP con costos no ha sido completamente caracterizado.

4. **Pathwidth:** Similar a treewidth pero con separadores lineales en lugar de árboles. Por la reducción desde Independent Set, también es W-hard para MCCPP.[1]

5. **Número de Colores r (Restricción):** Se puede demostrar que MCCPP es W-hard cuando se parametriza por r (el número de colores permitidos) [8].[2]

Resultado de W-Hardness para el Parámetro r:[2]

Siguiendo técnicas de reducción desde Dominating Set y resultados sobre Weighted Coloring, se obtiene:

Planteamiento

El problema de determinar $\sigma(G, w; r)$ (el costo mínimo de una coloración propia usando exactamente r colores) en un grafo ponderado es **W-hard** cuando se parametriza únicamente por r.[2]

Esto implica que incluso si se fija el número de colores a utilizar, el problema sigue siendo paramétricamente intratable (a menos que $\mathbf{W}[\mathbf{2}] = \mathbf{FPT}$, lo que implicaría un colapso de la jerarquía W) [8].

Análisis de Falta de FPT para Parámetros Naturales:

Por la reducción desde Independent Set y por los resultados de W-hardness para Weighted Coloring, los siguientes parámetros naturales **también son W-hard** para MCCPP, debido a que están todos acotados por la talla del mayor componente conexo:[1]

- Treewidth del grafo

- Pathwidth del grafo
- Degeneracy del grafo
- Número cromático $\chi(G)$
- Feedback Vertex Set (FVS)
- Distance to Clique
- Distance to Coloring
- Tree-Depth del grafo
- Clique-Cover Number

Conclusión

Basado en los resultados de Araújo, Baste y Sau, el **Problema de Partición Cromática de Costo Mínimo (MCCPP) no es FPT para ningún parámetro de estructura de grafo *estándar*** (treewidth, pathwidth, treedecomposición, vertex cover, etc.) [8].

Los únicos parámetros que potencialmente podrían hacer MCCPP FPT serían:

1. **Parámetros combinados muy específicos:** Por ejemplo, una combinación del número de colores r y una cota en el número de aristas o una cota fuerte en k (el tamaño del conjunto independiente máximo).
2. **Parámetros de distancia a clases especiales:** Distancia a grafos completos, distancia a árboles muy pequeños, o similares. Pero estos son ampliamente restrictivos y no capturan muchos grafos de interés práctico.
3. **Parametrización por la solución:** Parametrizar por el valor óptimo $\sigma(G,w)$ mismo, lo cual es una parametrización "circular" y rara vez se considera en análisis de complejidad parametrizada.

3. Diseño de Soluciones Algorítmicas

El MCCPP exige dividir V en conjuntos independientes minimizando un costo asignado a cada clase de color. Su dificultad combinatoria, dado que MCCPP es NP-Hard y APX-Hard, requiere un enfoque múltiple: exacto, aproximado, metaheurístico y estructural.

3.1. Soluciones Exactas (Bases de Referencia)

3.1.1. Algoritmo de Fuerza Bruta: Enumeración Completa

El algoritmo de fuerza bruta es esencialmente un algoritmo de Branch-and-Bound sin poda, cuyo propósito es encontrar la solución óptima Z^* en instancias de tamaño muy pequeño para servir como línea base experimental.

Mecanismo: El algoritmo explora sistemáticamente todas las posibles asignaciones de frecuencias a vértices. Para una instancia $\langle G, F, C \rangle$, con $|V| = n$ y $|F| = k$, se genera cada función de mapeo $\phi : V \to F$. Para cada asignación, se verifica la restricción de coloración propia, y si es válida, se calcula el costo total $\sum_{v \in V} c_{v,\phi(v)}$.

Pseudocódigo Formal

Input: Grafo $G = (V, E)$, $|V| = n$, $|E| = m$; Conjunto de colores F, $|F| = k$; Matriz de costos $C[v, f]$ para $v \in V, f \in F$
Output: Coloración óptima $\varphi^* : V \to F$; Costo óptimo Z^*
$Z^* \leftarrow +\infty$;
$\varphi^* \leftarrow$ `NULL`;
foreach *función* $\varphi : V \to F$ **do**
  factible $\leftarrow$ `TRUE`;
  foreach *arista* $\{u, v\} \in E$ **do**
    if $\varphi(u) = \varphi(v)$ **then**
      factible $\leftarrow$ `FALSE`;
      break;
    end
  end
  if *factible* **then**
    $Z_{\text{actual}} \leftarrow \sum_{v \in V} C[v, \varphi(v)]$;
    if $Z_{actual} < Z^*$ **then**
      $Z^* \leftarrow Z_{\text{actual}}$;
      $\varphi^* \leftarrow \varphi$;
    end
  end
end
return (φ^*, Z^*)

Algorithm 1: Fuerza Bruta para MCCPP

Demostración de Correctitud: El algoritmo de Fuerza Bruta encuentra la solución óptima Z^* para cualquier instancia $\langle G, F, C \rangle$ del MCCPP. Demostramos por tres propiedades fundamentales:

(1) Completitud (explora todo el espacio):

- El algoritmo enumera todas las k^n funciones posibles $\varphi : V \to F$ (línea 3).
- Por construcción, cualquier solución factible φ_{opt} es una de estas k^n configuraciones.
- Por tanto, φ_{opt} será examinada en alguna iteración i.

(2) Verificación de factibilidad:

- Las líneas 5–8 verifican la restricción de coloración propia: $\forall \{u, v\} \in E, \varphi(u) \neq \varphi(v)$.
- Esta verificación es exacta: una configuración pasa la prueba si y solo si es una coloración propia válida.

(3) Optimalidad:

- Sea Z_{opt} el costo de la solución óptima.
- En la iteración i donde se examina φ_{opt}:
  - *factible* = `TRUE` (es válida)
  - $Z_{\text{actual}} = \sum_{v \in V} C[v, \varphi_{\text{opt}}(v)] = Z_{\text{opt}}$ (línea 10)
  - Como $Z_{\text{opt}} \leq Z^*$ (por ser óptima), la condición de línea 11 se cumplirá en algún punto.
  - Z^* se actualiza a Z_{opt} (línea 12).
- Ninguna iteración posterior puede encontrar $Z_{\text{actual}} < Z_{\text{opt}}$ (por definición de óptimo).
- Por tanto, al terminar: $Z^* = Z_{\text{opt}}$.

Análisis de Complejidad Temporal: La complejidad temporal del algoritmo de Fuerza Bruta para MCCPP es $\Theta(k^n \cdot m)$.

Analizamos cada componente del algoritmo:

- **Línea 3:** Genera k^n configuraciones $\Rightarrow O(k^n)$ iteraciones.
- **Líneas 5–8:** Verifica m aristas $\Rightarrow O(m)$ por iteración.
- **Línea 10:** Suma n costos $\Rightarrow O(n)$ por iteración.
- **Complejidad total:** $k^n \cdot (m+n) = k^n \cdot O(m+n) = \Theta(k^n \cdot (m+n))$

Cota inferior (Ω):

- Cualquier algoritmo que garantice optimalidad debe examinar al menos una solución óptima.
- En el peor caso (costos aleatorios), no hay forma de podar sin examinar todas las k^n configuraciones.
- Por tanto: $\Omega(k^n \cdot (m+n))$.

Esta dependencia exponencial confirma que el algoritmo solo es viable para valores de n y k extremadamente pequeños.

3.1.2. Backtracking y Backtracking inteligente:

Backtracking clásico

El algoritmo de backtracking construye incrementalmente una coloración parcial del grafo, asignando colores a los vértices siguiendo un orden fijo $(v_1, \ldots, v_n)$. En cada nivel de la recursión se asigna un color a un vértice siempre que dicha asignación no viole las restricciones de adyacencia con los vértices ya coloreados.

El algoritmo explora recursivamente el árbol de búsqueda hasta completar una asignación válida para todos los vértices, evaluando entonces su costo total.

Correctitud El algoritmo de backtracking devuelve una coloración óptima si se explora completamente el árbol de búsqueda.

Demostración: Cada nodo del árbol de búsqueda corresponde a una coloración parcial factible. El algoritmo únicamente descarta ramas que violan la condición de coloración propia, por lo que toda coloración factible completa aparece como una hoja del árbol.

Dado que el algoritmo examina todas las coloraciones factibles, evalúa su costo y conserva la de menor valor, la solución devuelta es óptima por definición.

Complejidad temporal: En el peor caso, el algoritmo explora todas las posibles asignaciones de k colores a n vértices, lo que da lugar a una complejidad temporal de:

$$O(k^n).$$

La verificación local de factibilidad puede realizarse en tiempo proporcional al grado del vértice, lo cual queda absorbido por la cota exponencial dominante.

Conclusión: se trata de un algoritmo exacto, pero intratable para instancias de tamaño medio o grande.

Backtracking inteligente con poda:

La versión inteligente del backtracking introduce dos mejoras principales:

- Ordenación heurística de los vértices, por ejemplo, por grado decreciente.
- Poda por costo (*branch and bound*), descartando ramas cuyo costo parcial supera la mejor solución conocida.

Durante la exploración, el algoritmo mantiene el costo acumulado de la coloración parcial y evita explorar extensiones que no puedan conducir a una solución óptima.

Correctitud La poda por costo no elimina ninguna solución óptima.

Demostración: Sea C^* el costo de una solución óptima global y C_{best} el menor costo encontrado hasta el momento. Una rama se poda únicamente cuando su costo parcial C_p satisface $C_p \geq C_{best}$, con $C_{best} \leq C^*$.

Dado que cualquier extensión de dicha rama tendrá un costo total al menos C_p, ninguna solución descendiente puede mejorar C_{best}. Por tanto, la poda no descarta soluciones óptimas. □

Complejidad temporal En el peor caso, el algoritmo sigue teniendo complejidad exponencial:

$$O(k^n).$$

No obstante, en la práctica el número de nodos explorados se reduce considerablemente gracias a la poda temprana y a la exploración prioritaria de soluciones de bajo costo.

Conclusión: el algoritmo mantiene correctitud exacta con mejoras prácticas significativas.

3.1.3. Resolución mediante Programación Entera Lineal (ILP)

El modelo de Programación Entera Lineal presentado en la Sección 1 se implementa utilizando un solver de ILP para obtener soluciones óptimas en instancias de tamaño moderado. Este enfoque representa una mejora significativa respecto a la fuerza bruta al aprovechar técnicas de poda, relajación y planos de corte.

Mecanismo: El modelo ILP 0-1 se implementa utilizando un solver especializado (como Gurobi, CPLEX o SCIP). El solver aplica algoritmos de Branch-and-Bound con relajación lineal, donde en cada nodo se resuelve la relajación LP del problema para obtener cotas que permitan podar ramas subóptimas. Adicionalmente, se pueden generar planos de corte para fortalecer la formulación.

Ventajas sobre Fuerza Bruta:

- **Poda por Optimalidad:** Las cotas de la relajación LP permiten descartar soluciones que no pueden mejorar la incumbente.
- **Poda por Factibilidad:** Se detectan infactibilidades sin enumerar completamentamente.
- **Preprocesamiento:** Los solvers aplican reducciones de variables y restricciones para simplificar el problema.

Limitaciones Prácticas: Aunque más eficiente que la enumeración completa, la resolución mediante ILP sigue estando limitada por la NP-hardness del problema. El crecimiento exponencial del árbol de búsqueda en instancias grandes hace necesario el uso de heurísticas o metaheurísticas para casos de tamaño realista.

Implementación: El modelo se codifica en un lenguaje de modelado (como Python+PuLP/Pyomo, AMPL o Julia+JuMP) y se resuelve mediante un solver comercial o de código abierto. La función objetivo y restricciones se implementan directamente según la formalización presentada en la Sección 1.

3.1.4. Programación Dinámica (DP) para Grafos Estructurados

La Programación Dinámica (DP) es una técnica exacta que explota la subestructura óptima y la superposición de subproblemas para convertir (en casos estructurados) problemas que en grafos generales son intratables, en algoritmos polinómicos.

Aplicación y Estructura del Grafo: Árboles

El MCCPP (Problema de Partición Cromática de Costo Mínimo) se vuelve tratable mediante DP en tiempo polinomial para la clase de grafos que tienen estructura de árbol. En un árbol, no hay ciclos y la estructura jerárárquica permite un DP en post-orden (rooted tree DP) con estados por vértice y color.

Modelado del Problema en Árboles

Sea $G = (V, E)$ un árbol con $n = |V|$ vértices. Disponemos de un conjunto de colores (frecuencias) F de tamaño k. Para cada vértice $v \in V$ y color $f \in F$ existe un coste no negativo $c_{v,f}$. El objetivo es asignar a cada vértice un color tal que cualquiera par de vértices adyacentes tengan colores distintos (coloración propia) minimizando la suma de los costos. Para este propósito vamos a modelar este problema a través del esquema SRTBOT:

S (Subproblemas): Fijamos una raíz arbitraria r del árbol y consideramos la orientación padre–hijo. Para cada vértice $v \in V$ y cada color $c \in \{1, \ldots, k\}$, definimos el estado: $DP[v][c] =$ coste mínimo para colorear todo el subárbol con raíz v, dado que el nodo v utiliza el color c.

R (Recurrencia): Para un nodo v, el costo depende de su propio peso cromático y del costo mínimo de sus hijos $u \in children(v)$, asegurando que no repitan el color c:

$$DP[v][c] = c_{v,c} + \sum_{u \in children(v)} \left(\min_{c' \in F, c' \neq c} DP[u][c'] \right)$$

T (Topología): El cálculo se realiza en Post-orden (Bottom-up). Se procesan primero las hojas y se asciende hacia la raíz, garantizando que cuando calculemos un nodo, los valores de sus hijos ya estén disponibles.

B (Casos Base): Si v es una hoja, no tiene hijos que restringir:

$$DP[v][c] = c_{v,c} \quad \forall c \in \{1, \ldots, k\}$$

O (Solución Original): La solución global óptima para el árbol completo se encuentra minimizando sobre todos los colores posibles en la raíz:

$$Z^* = \min_{c \in F} DP[r][c]$$

Construcción de la solución: Tras calcular todos los $DP[v][c]$ en post-orden, reconstruimos la coloración seleccionando para la raíz el color c que minimiza el valor total acumulado $DP[r][c]$. Es importante notar que este color no es necesariamente el de menor peso intrínseco (w_c), sino aquel que optimiza la suma total del subárbol. Luego, recursivamente, para cada hijo seleccionamos el color mínimo compatible (diferente del color del padre) según la tabla DP.

T (Tiempo de Ejecución): Para cada uno de los n vértices y k colores, sumamos sobre los hijos. La complejidad es $O(n \cdot k^2)$. Si además mantenemos el mínimo y el segundo mínimo de los costos de los hijos, se puede reducir a $O(n \cdot k)$.

Correctitud del DP en árboles

Para todo vértice v y color c, $DP[v][c]$ definido por la recurrencia anterior es el coste óptimo de colorear el subárbol de v condicionando a que v tome el color c.

Demostración

Prueba por inducción sobre la altura del subárbol raíz en v.

Base: si v es hoja, la única elección es colorear v con c y el coste es $c_{v,c}$, que coincide con $DP[v][c]$.

Paso inductivo: Supongamos cierta la premisa para todos los nodos con altura menor que la de v. Cualquier coloración óptima del subárbol de v donde v usa color c induce para cada hijo u una

coloración óptima del subárbol de u con la restricción de no usar c en u. Por la hipótesis inductiva, el coste óptimo de cada subárbol u con esa restricción es $\min_{c' \neq c} DP[u][c']$. Sumando y añadiendo $c_{v,c}$ obtenemos precisamente la expresión de la recurrencia.

Análisis de la Complejidad Temporal

Sea n el número de vértices y k el número de colores.

- Preprocesamiento: construir la orientación padre–hijo y el post-orden toma $O(n)$.
- Cálculo de DP: para cada vértice v y cada color c se debe sumar sobre los hijos u el término $\min_{c' \neq c} DP[u][c']$. Para un hijo u obtener $\min_{c' \neq c} DP[u][c']$ cuesta $O(k)$ con una implementación directa. Por cada par (v,c) se evalúan esto para todos los hijos de v. Como $\sum_v \deg(v) = O(n)$, la complejidad total es:

$$T(n,k) = O(nk^2).$$

- Existe una optimización estándar (mantener para cada hijo el mínimo y segundo mínimo y el correspondiente color), que reduce cada consulta a $O(1)$ y lleva la complejidad a:

$$T(n,k) = O(nk).$$

Conclusión teórica

La DP en árboles ofrece una solución exacta y polinomial para MCCPP cuando el grafo de interferencia es un árbol. La implementación directa tiene coste $O(nk^2)$; con la técnica de mínimo/segundo mínimo por hijo se puede reducir a $O(nk)$.

Programación Dinámica para Grafos de Intervalo

Los grafos de intervalo son aquellos donde cada vértice corresponde a un intervalo en la recta real, y dos vértices son adyacentes si y solo si sus intervalos se solapan. Esta estructura especial permite algoritmos de programación dinámica eficientes para el MCCPP.

Es fundamental destacar que mientras todo grafo de intervalo es necesariamente cordal, la relación inversa no se cumple. Existen grafos cordales que no admiten una representación válida como grafos de intervalo. El ejemplo más notable es el grafo completo de cuatro o más vértices, que es cordal pero no puede representarse mediante intervalos en la recta real. Esta distinción es crítica para comprender el ámbito de aplicabilidad del algoritmo que se presenta a continuación.

Prerrequisitos: El algoritmo requiere como entrada un grafo que satisfaga dos condiciones necesarias.

1. **Condición 1:** El grafo debe ser cordal, lo cual puede verificarse en tiempo $O(n+m)$ mediante algoritmos estándar de reconocimiento de grafos cordales.

2. **Condición 2:** El grafo debe ser efectivamente de intervalo, no simplemente cordal. La verificación de esta segunda propiedad se realiza mediante la construcción explícita de una representación de intervalos y la validación de que dicha representación es consistente con la estructura de adyacencias del grafo.

Para grafos que no satisfacen estas condiciones, particularmente aquellos que son cordales pero no de intervalo, el algoritmo presentado no es aplicable y debe recurrirse a métodos alternativos.

Algoritmo DP para Grafos de Intervalo mediante esquema SRTBOT:

El algoritmo utiliza un esquema de programación dinámica que procesa los vértices según un ordenamiento derivado de su representación de intervalos. A diferencia del enfoque clásico de programación dinámica sobre árboles, este método mantiene estados que capturan únicamente los intervalos activos en cada etapa del procesamiento, aprovechando la propiedad fundamental de que en grafos de intervalo, el número de intervalos que se solapan simultáneamente está acotado por el tamaño de la clique máxima.

S (Subproblemas): Dada una representación de intervalos válida $I(v) = [l(v), r(v)]$ para cada vértice v, ordenamos los vértices por el extremo derecho de sus intervalos, obteniendo la secuencia $v_1, v_2, \ldots, v_n$ donde $r(v_1) \leq r(v_2) \leq \cdots \leq r(v_n)$.

El estado de programación dinámica se define como $DP[i][\sigma]$, que representa el costo mínimo de colorear los primeros i vértices en el ordenamiento, dado que el estado σ especifica la coloración de los intervalos que permanecen activos en la posición i. Formalmente, un intervalo v_j (con $j < i$) está activo en la posición i si $r(v_j) \geq l(v_i)$, es decir, si su intervalo se solapa con el intervalo del vértice que se está procesando.

El estado σ se representa como una tupla ordenada de pares (v_j, c) donde v_j es un vértice activo y c es el color asignado a dicho vértice en la solución parcial correspondiente. Esta representación compacta del estado es posible gracias a la propiedad de que solo los intervalos activos son relevantes para determinar las restricciones de coloración de vértices futuros.

R (Recurrencia): Para $i > 1$, la recurrencia se define mediante el proceso de extensión desde estados anteriores. Para cada estado previo $DP[i-1][\sigma']$, construimos el conjunto de intervalos activos filtrando aquellos que ya han terminado. Específicamente, un intervalo v_j incluido en σ' permanece activo si $r(v_j) \geq l(v_i)$.

Sea σ'_{activo} el estado resultante tras eliminar los intervalos inactivos de σ'. Para cada color $c \in F$ disponible para el vértice v_i, verificamos la compatibilidad con los intervalos activos. La condición de compatibilidad requiere que para todo $(v_j, c') \in \sigma'_{\text{activo}}$ tal que v_j es adyacente a v_i, se cumpla que $c \neq c'$.

Si la compatibilidad se satisface, el nuevo estado σ se construye como $\sigma = \sigma'_{\text{activo}} \cup \{(v_i, c)\}$, y el costo asociado se calcula como:

$$DP[i][\sigma] = \text{mín}\{DP[i-1][\sigma'] + C[v_i, c] : \sigma' \text{ compatible con la asignación de } c \text{ a } v_i\}$$

donde $C[v_i, c]$ denota el costo de asignar el color c al vértice v_i según la matriz de costos de entrada.

T (Topología): El procesamiento sigue un orden lineal ascendente según el ordenamiento de los vértices por el extremo derecho de sus intervalos. Este ordenamiento garantiza que cuando procesamos el vértice v_i, todos los vértices v_j con $j < i$ ya han sido considerados, y podemos determinar precisamente cuáles de ellos permanecen activos basándonos en la comparación de sus extremos derechos con el extremo izquierdo de v_i.

B (Base): Para el primer vértice v_1 en el ordenamiento, no existen vértices previamente procesados, por lo que el conjunto de intervalos activos está vacío. Para cada color $c \in F$, establecemos:

$$DP[1][\{(v_1, c)\}] = C[v_1, c]$$

Este caso base inicializa la programación dinámica con todas las coloraciones posibles del primer vértice, cada una representando una solución parcial válida que será extendida en las etapas subsiguientes.

O (Objetivo): La solución óptima se obtiene identificando el estado con costo mínimo en la última etapa del procesamiento:

$$Z^* = \text{mín}\{DP[n][\sigma] : \sigma \text{ es un estado válido en la posición } n\}$$

Cada estado σ en $DP[n]$ representa una coloración completa y propia del grafo, ya que todos los n vértices han sido procesados. La coloración específica asociada al estado óptimo se recupera mediante el seguimiento de las decisiones tomadas durante la construcción de la tabla de programación dinámica.

T (Tiempo): La complejidad temporal del algoritmo está determinada por tres factores principales. El preprocesamiento, que incluye la verificación de cordalidad, la construcción de la representación de intervalos y el ordenamiento de vértices, requiere $O(n^2 + m)$ operaciones en el peor

caso. El cálculo de la tabla de programación dinámica procesa n etapas, una por cada vértice. En cada etapa i, el número de estados posibles está acotado por $O(k^{\omega_i})$ donde ω_i es el número máximo de intervalos que se solapan simultáneamente hasta la posición i. Para cada estado, se evalúan k posibles colores para el vértice actual, y la verificación de compatibilidad con los intervalos activos requiere $O(\omega_i)$ operaciones.

En grafos de intervalo típicos, el valor ω (tamaño de la clique máxima) es mucho menor que n, lo que hace el algoritmo eficiente en la práctica. La complejidad total resulta en $O(n^2 + n \cdot k^{\omega+1} \cdot \omega)$, que se simplifica a $O(n \cdot k^{\omega+1} \cdot \omega)$ cuando el grafo es suficientemente denso. Para grafos dispersos con cliques pequeñas, esta complejidad es polinomial en la práctica.

Análisis de Complejidad:

El análisis de complejidad requiere considerar la estructura específica de los grafos de intervalo y cómo esta afecta el tamaño del espacio de estados en cada etapa de la programación dinámica.

- **Preprocesamiento:** La fase de preprocesamiento consta de varias operaciones secuenciales. La verificación de cordalidad mediante Maximum Cardinality Search se ejecuta en tiempo $O(n+m)$. La construcción de la representación de intervalos, que involucra el análisis de las relaciones de vecindad para cada vértice y la asignación de coordenadas de intervalos, requiere $O(n^2)$ operaciones en el peor caso. La validación de que la representación construida es consistente con el grafo original implica verificar para cada par de vértices si la condición de solapamiento de intervalos corresponde a la existencia de una arista, lo cual también requiere $O(n^2)$ comparaciones. Finalmente, el ordenamiento de los vértices por el extremo derecho de sus intervalos se realiza en $O(n \log n)$. El preprocesamiento total es por tanto $O(n^2)$ en el peor caso.

- **Cálculo DP:** El cálculo de la tabla procede en n etapas, una por cada vértice en el ordenamiento. En la etapa i, el algoritmo debe considerar todos los estados válidos de la etapa anterior $DP[i-1]$. El número de estados en $DP[i-1]$ está acotado superiormente por el número de coloraciones distintas posibles de los intervalos que están activos en esa posición. Si hay ω_{i-1} intervalos activos en la posición $i-1$, existen potencialmente $k^{\omega_{i-1}}$ estados distintos, aunque en la práctica este número es menor debido a restricciones de compatibilidad.

  Para cada estado σ' en $DP[i-1]$, el algoritmo realiza las siguientes operaciones. Primero, filtra los intervalos activos eliminando aquellos cuyo extremo derecho es menor que el extremo izquierdo del vértice actual, lo cual requiere $O(\omega_{i-1})$ comparaciones. Segundo, para cada uno de los k colores posibles del vértice v_i, verifica la compatibilidad con los intervalos activos revisando si algún vecino activo tiene el mismo color, operación que requiere $O(\omega_i)$ verificaciones de adyacencia. Tercero, si la asignación es compatible, construye el nuevo estado y actualiza la tabla $DP[i]$.

El número de estados en cada etapa i está acotado por k^{ω_i}, y cada transición desde un estado previo a un nuevo estado requiere $O(k \cdot \omega_i)$ operaciones. Sumando sobre todas las etapas, obtenemos una complejidad de $O(n \cdot k^{\omega} \cdot k \cdot \omega) = O(n \cdot k^{\omega+1} \cdot \omega)$ para el cálculo completo de la tabla de programación dinámica.

- **Recuperación de la Solución:** Una vez completada la tabla de programación dinámica, la identificación del estado óptimo en $DP[n]$ requiere examinar todos los estados finales, operación que toma $O(k^{\omega})$ tiempo. La reconstrucción de la coloración completa a partir del estado óptimo ya está disponible implícitamente en la información almacenada durante la construcción de la tabla, por lo que no añade complejidad asintótica adicional.

La complejidad total del algoritmo es $O(n^2 + n \cdot k^{\omega+1} \cdot \omega)$. Para grafos de intervalo con cliques pequeñas, específicamente cuando ω es constante o crece logarítmicamente con n, esta expresión es polinomial en n y k. Sin embargo, para grafos de intervalo densos donde ω es proporcional a n, la complejidad se vuelve exponencial, reflejando la naturaleza inherentemente difícil del problema en tales instancias.

Demostración de Correctitud: Demostraremos que el algoritmo calcula el costo mínimo de coloración para un grafo de intervalo G utilizando inducción sobre el número de vértices procesados.

Preposición: Para cada $i \in \{1, 2, \ldots, n\}$ y cada estado σ en $DP[i]$, el valor $DP[i][\sigma]$ representa el costo mínimo de colorear los vértices $\{v_1, v_2, \ldots, v_i\}$ de manera propia, sujeto a que los intervalos activos en la posición i tengan exactamente las coloraciones especificadas por σ.

Base ($i = 1$): Para el primer vértice v_1, el algoritmo establece $DP[1][\{(v_1, c)\}] = C[v_1, c]$ para cada color c. Esta asignación es correcta porque v_1 es el único vértice procesado, no tiene restricciones de vecindad previas, y el costo de esta solución parcial es precisamente el costo de asignar el color c al vértice v_1. Por tanto, la proposición se cumple para $i = 1$.

Paso Inductivo: Supongamos que la proposición es cierta para $i-1$. Queremos demostrar que es cierta para i. Sea σ un estado en $DP[i]$. Por construcción del algoritmo, σ se obtiene al extender algún estado σ' de $DP[i-1]$ mediante la asignación de un color c al vértice v_i.

Demostraremos primero la factibilidad. El estado σ' representa una coloración propia de $\{v_1, \ldots, v_{i-1}\}$ por la hipótesis de inducción. Al agregar v_i con color c, el algoritmo verifica que c sea diferente del color asignado a cada vecino de v_i que está incluido en σ'_{activo}. Dado que σ'_{activo} contiene precisamente los vértices cuyo intervalo se solapa con el intervalo de v_i, y dado que en un grafo de intervalo dos vértices son adyacentes si y solo si sus intervalos se solapan, la verificación de compatibilidad garantiza que la coloración extendida sigue siendo propia. Por tanto, σ representa una coloración factible.

Demostraremos ahora la optimalidad. El costo asociado a σ es $DP[i][\sigma] = DP[i-1][\sigma'] + C[v_i, c]$, donde σ' es el estado previo desde el cual se extendió. Por la hipótesis de inducción, $DP[i-1][\sigma']$ es el costo mínimo de colorear $\{v_1, \ldots, v_{i-1}\}$ con las coloraciones especificadas en σ'. El costo total de extender esta solución agregando v_i con color c es precisamente la suma de estos dos términos. El algoritmo considera todas las combinaciones válidas de estados previos y colores para v_i, y retiene el costo mínimo para cada estado resultante σ. Por tanto, $DP[i][\sigma]$ es efectivamente el costo mínimo de alcanzar el estado σ en la posición i.

Al llegar a la posición n, todos los vértices han sido procesados. El algoritmo identifica el estado σ^* en $DP[n]$ con costo mínimo. Por el argumento inductivo, este estado representa una coloración propia completa del grafo con costo mínimo. Ninguna otra coloración propia puede tener menor costo, porque cualquier coloración propia debe pasar por algún estado en $DP[n]$, y el algoritmo ha considerado todos ellos. Por tanto, el algoritmo es correcto y encuentra la solución óptima.

Limitación Práctica: La limitación más fundamental es que el algoritmo solo es aplicable a grafos de intervalo genuinos, no a grafos cordales arbitrarios. Para grafos cordales que no admiten representación de intervalos, el algoritmo rechazará la instancia en la fase de preprocesamiento.

Dependencia del Tamaño de la Clique Máxima: El desempeño del algoritmo depende críticamente del tamaño de la clique máxima ω del grafo. Para grafos de intervalo con cliques pequeñas, el algoritmo es eficiente y práctico. Sin embargo, para grafos de intervalo densos donde ω se aproxima a n, la complejidad $k^{\omega+1}$ hace que el algoritmo sea impracticable incluso para valores moderados de n y k. Esta sensibilidad a la densidad del grafo debe considerarse al seleccionar instancias apropiadas para este método.

Construcción de la Representación de Intervalos: La verificación de si un grafo cordal es efectivamente de intervalo y la construcción de una representación válida de intervalos son problemas no triviales. El algoritmo presentado utiliza una heurística basada en el ordenamiento de eliminación perfecta para construir esta representación.

3.2. Algoritmos de Aproximación

Este enfoque aborda la intratabilidad del MCCPP ofreciendo una garantía de calidad probada ($O(\ln n)$), heredada de su modelado como Weighted Set Cover [4], [5].

3.2.1. Algoritmo de Aproximación con Garantía Teórica ($R \leq O(\ln |V|)$)

Dada la APX-Hardness del MCCPP, no se espera una aproximación de factor constante para grafos generales. En su lugar, el algoritmo de aproximación con la mejor garantía teórica utiliza la conexión del MCCPP con el Problema de Cobertura de Conjuntos Ponderado (Weighted Set

Cover, WSC), como ya se analizó en la fase anterior.

Algoritmo de Aproximación WSC-Greedy:

El algoritmo procede de forma voraz, seleccionando repetidamente el conjunto independiente $S^* = \langle I^*, f^* \rangle$ que minimiza el costo marginal efectivo, es decir, el costo por cada nuevo vértice cubierto.

$$\langle I^*, f^* \rangle = \arg \min_{\langle I,f \rangle \in \mathscr{S}} \left\{ \frac{\sum_{v \in I} c_{v,f}}{|\{v \in I : v \text{ está sin colorear}\}|} \right\}$$

Los vértices cubiertos por I^* se colorean con f^*, y se eliminan del universo U. Este proceso se repite hasta que todos los vértices estén coloreados.

Garantía de Rendimiento:

El algoritmo Greedy para WSC garantiza que la solución obtenida Z_{Alg} está acotada por el costo óptimo Z^*_{WSC} multiplicado por el número armónico $H(|U|)$, donde $H(|U|) = \sum_{i=1}^{|U|} 1/i$. Dado que el MCCPP se ha modelado como WSC y el número armónico $H(n)$ se aproxima por $\ln n$, se tiene la siguiente garantía de rendimiento:

$$R = \frac{Z^{Alg}_{MCCPP}}{Z^*_{MCCPP}} \leq H(|V|) \approx O(\ln |V|)$$

Esta garantía de $\mathbf{R} \leq \mathbf{O}(\ln |\mathbf{V}|)$ es el mejor factor de aproximación conocido para el MCCPP en grafos generales, coherente con los límites inferiores de inaproximabilidad de Set Cover [4], [5].

Demostración: Sea $n = |V|$. Numeremos los vértices en el orden en que son coloreados por el algoritmo: $v_1, v_2, \ldots, v_n$. Sea $c(v_k)$ el costo asignado al vértice v_k en el momento de su cobertura. Si v_k fue cubierto por el conjunto independiente I^* con color f^*, entonces:

$$c(v_k) = \frac{\sum_{v \in I^*} c_{v,f^*}}{|I^* \setminus \{v_1, \ldots, v_{k-1}\}|}$$

En el paso donde se elige v_k, quedan al menos $n-k+1$ vértices sin cubrir. La solución óptima Z^* cubre estos vértices con un costo total de a lo sumo Z^*. Por el principio del palomar, debe existir al menos un conjunto independiente en la solución óptima cuyo costo por vértice no cubierto sea $\leq \frac{Z^*}{n-k+1}$. Como el algoritmo elige el conjunto que minimiza este ratio:

$$c(v_k) \leq \frac{Z^*}{n-k+1}$$

Sumando todos los costos asignados:

$$Z_{Alg} = \sum_{k=1}^{n} c(v_k) \leq \sum_{k=1}^{n} \frac{Z^*}{n-k+1} = Z^* \sum_{j=1}^{n} \frac{1}{j} = Z^* \cdot H(n)$$

Dado que $H(n) \leq \ln n + 1$, se concluye que $R \leq \ln |V| + 1 = O(\ln |V|)$.

Nota sobre Implementación Práctica del Algoritmo WSC-Greedy:

La implementación directa del algoritmo greedy para WSC, tal como se describe teóricamente, requiere enumerar todos los conjuntos independientes posibles del grafo para cada color disponible, calcular su costo-efectividad, y seleccionar el óptimo. Sin embargo, el número de conjuntos independientes en un grafo puede ser exponencial (hasta $2^{n/2}$ en el peor caso), lo cual es computacionalmente intratable.

Heurísticas de Implementación:

En la práctica, se utilizan heurísticas para construir conjuntos independientes representativos de alta calidad:

- **Greedy por costo:** Construir conjuntos independientes maximales eligiendo iterativamente el vértice no cubierto de menor costo $c_{v,f}$ para la frecuencia f, que no sea adyacente a vértices ya seleccionados en el conjunto.
- **Greedy por grado:** Priorizar vértices de alto grado en la construcción del conjunto independiente para lograr mejor empaquetamiento y reducir el número de conjuntos necesarios.
- **Muestreo aleatorio:** Generar múltiples conjuntos independientes mediante diferentes estrategias aleatorias o semi-aleatorias, evaluar su ratio costo-efectividad, y elegir el mejor.
- **Construcción híbrida:** Combinar criterios estructurales (grado, saturación) con criterios de costo para balancear tamaño del conjunto independiente y costo total.

Garantía de Aproximación con Heurísticas:

Aunque estas heurísticas no exploran todos los conjuntos independientes posibles, el análisis teórico del algoritmo greedy para WSC sigue siendo válido siempre que:

1. Se construyan conjuntos independientes maximales (no necesariamente máximos)
2. Se seleccione en cada iteración un conjunto con ratio costo-efectividad competitivo

La garantía $O(\ln n)$ se mantiene en el análisis promedio, aunque en casos patológicos específicos la heurística podría no alcanzarla. Sin embargo, la evidencia empírica muestra que las heurísticas mencionadas producen soluciones de calidad comparable a la cota teórica en grafos realistas.

Complejidad Práctica:

Con estas heurísticas, la complejidad práctica del algoritmo es $O(k \cdot n^2 \cdot d)$ donde k es el número de colores/frecuencias, n el número de vértices, y d el grado promedio, en lugar de la complejidad exponencial de la enumeración completa.

NOTA: Es importante notar que el paso de selección:

$$\arg \min_{\langle I,f \rangle \in \mathscr{S}} \text{ratio}(I,f)$$

requiere resolver una variante del *Maximum Weight Independent Set* (MWIS) para cada color f, el cual es un problema NP-Hard por si mismo. Por tanto, el algoritmo de aproximación ideal no es de tiempo polinomial a menos que $P = NP$.

La complejidad $O(k \cdot n^2 \cdot d)$ mencionada anteriormente corresponde a la versión donde se utiliza una **heurística greedy de construcción de conjuntos independientes**. En este caso, la garantía teórica $O(\ln |V|)$ se mantiene de forma aproximada si la heurística de construcción garantiza una fracción constante del tamaño del conjunto independiente máximo.

WSC-Greedy mejorado:

Si bien el algoritmo WSC-Greedy ofrece una garantía teórica robusta en términos de aproximación asintótica, su naturaleza voraz implica que las decisiones tomadas en etapas tempranas son irrevocables, lo que puede conducir a ineficiencias locales en la asignación final de colores. Para mitigar este efecto y mejorar la calidad práctica de la solución, se implementa una segunda etapa de optimización basada en Búsqueda Local.

Descripción del Algoritmo: Este procedimiento toma como entrada la solución factible S_{greedy} generada por el algoritmo de aproximación anterior y busca iterativamente mejoras incrementales explorando el vecindario de la solución actual. El algoritmo sigue una estrategia de *Hill Climbing* (escalada de colinas) con descenso estricto en el costo.

Sea $\mathscr{C} : V \to \{1,\ldots,k\}$ la coloración actual. En cada iteración, el algoritmo explora, para cada vértice $v \in V$, si existe un color alternativo $f' \neq \mathscr{C}(v)$ tal que:

1. **Factibilidad:** El cambio de color es válido, es decir, ningún vecino de v tiene el color f':

$$\forall u \in N(v), \mathscr{C}(u) \neq f'$$

2. **Mejora de Costo:** El costo de asignar el color f' a v es estrictamente menor al costo actual:

$$c_{v,f'} < c_{v,\mathscr{C}(v)}$$

Si se encuentran múltiples colores que satisfacen estas condiciones para un vértice v, se selecciona aquel que minimiza el costo local (estrategia *Best-Improvement* a nivel de vértice). Si el cambio se efectúa, la solución actual y el costo total se actualizan inmediatamente.

Convergencia y Complejidad: El algoritmo itera sobre todos los vértices del grafo repetidamente hasta que se cumple uno de los siguientes criterios de parada:

- **Óptimo Local:** Se completa una pasada completa sobre todos los vértices sin realizar ningún cambio (la solución ha alcanzado un mínimo local).
- **Límite de Iteraciones:** Se alcanza un número máximo de pasadas predefinido (en la implementación, $MaxIter = 50$) para evitar ciclos o tiempos de ejecución excesivos en instancias muy grandes.

La complejidad de cada iteración de búsqueda local es $O(n \cdot d \cdot k)$, donde n es el número de vértices, d el grado promedio (para verificar vecinos) y k el número de colores disponibles. Dado que el número de iteraciones está acotado por una constante baja, el impacto en el tiempo total de ejecución es marginal en comparación con la construcción inicial, mientras que la reducción del costo objetivo suele ser significativa.

Justificación Teórica: Aunque la Búsqueda Local no mejora la cota teórica del peor caso ($O(\ln|V|)$) establecida por el algoritmo WSC-Greedy, es fundamental en la práctica porque corrige decisiones miopes del algoritmo voraz. Por ejemplo, un vértice pudo haber sido coloreado con un color costoso en una etapa temprana porque los colores baratos estaban bloqueados por vecinos que, posteriormente, podrían haber cambiado de color, liberando la opción económica. La búsqueda local explota estas oportunidades de re-optimización.

$$Z_{Final} = Z_{Greedy} - \sum_{t=1}^{T} \Delta_{cost}^{(t)} \leq Z_{Greedy} \tag{1}$$

Donde $\Delta_{cost}^{(t)}$ representa la reducción de costo en el paso t. Por tanto, esta etapa garantiza que la solución final es monotónicamente no inferior a la solución de la etapa de aproximación.

3.3. Heurísticas Greedy Cost-Aware

Para instancias grandes donde las soluciones exactas y las aproximaciones con garantía teórica logarítmica son insuficientes en la práctica, las heurísticas greedy adaptadas al costo juegan un papel esencial. Estas heurísticas se basan en estrategias clásicas de coloración (LF, DSATUR, RLF) enriquecidas con información de costos, y su efectividad práctica ha sido confirmada en estudios recientes de coloreo y MCCPP [12], [13].

3.3.1. Estrategia Largest First (LF) Cost-Aware

Descripción y Mecanismo:

La heurística Largest First (LF) ordena los vértices en orden no creciente de su grado $d(v)$. El algoritmo recorre esta ordenación y asigna a cada vértice v el color disponible de mínimo costo posible, respetando la restricción de coloración propia.

Definición Formal:

Sea $V = \{v_1, v_2, \ldots, v_n\}$ tal que $d(v_1) \geq d(v_2) \geq \ldots \geq d(v_n)$. Para cada vértice v_i:

- Determinar el conjunto de colores disponibles:

$$F_{\text{avail}}(v_i) = F \setminus \{\phi(u) : u \in N(v_i)\}$$

- Seleccionar el color de costo mínimo:

$$\phi(v_i) = f^*, \quad \text{donde } f^* = \arg \min_{f \in F_{\text{avail}}(v_i)} c_{v_i,f}$$

Análisis Formal de la Complejidad Computacional:

La complejidad de LF se determina por la fase de ordenación inicial y la fase secuencial de coloración. Sean $n = |V|$, $m = |E|$, y $k = |F|$.

1. **Costo de Preordenamiento:** Calcular todos los grados $d(v)$ toma $O(n+m)$. El ordenamiento de los n vértices toma $O(n \log n)$.

2. **Fase de Coloración:** Se itera n veces. En cada iteración, se verifica el conjunto de colores disponibles. Determinar $F_{\text{avail}}(v)$ implica examinar a los vecinos de v, lo que toma $O(d(v))$. La selección del costo mínimo f^* entre las frecuencias disponibles toma $O(|F|)$.

El costo de la coloración es:

$$\sum_{v \in V} O(d(v) + |F|) = O\left(\sum d(v) + n|F|\right) = O(2m + n|F|)$$

La complejidad total está dominada por:

$$\mathbf{T_{LF}} = \mathbf{O}(|\mathbf{V}| \log |\mathbf{V}| + |\mathbf{E}| + |\mathbf{V}||\mathbf{F}|)$$

Resultados Esperados e Implicaciones (Calidad y Limitaciones):

LF es la heurística más rápida, pero su estrategia estática conlleva una **baja eficiencia cromática**; es decir, a menudo utiliza un número de colores χ_{Alg} significativamente mayor que el número

cromático óptimo $\chi(G)$. Esta deficiencia estructural limita la capacidad de la adaptación Cost-Aware para optimizar el costo total. Si los vértices de alto grado que se colorean primero tienen costos uniformemente bajos, el algoritmo podría forzar a los vértices de bajo grado (coloreados al final) a usar frecuencias muy costosas, resultando en una solución final subóptima.

3.3.2. Estrategia DSATUR (Degree of Saturation) Cost-Aware

Descripción y Mecanismo:

DSATUR (Degree of Saturation) es una heurística dinámica que prioriza la selección del vértice más restringido en cada paso. El grado de saturación $d_{\text{sat}}(v)$ es el número de colores distintos ya utilizados por los vecinos de v. Se selecciona el vértice no coloreado v con el $d_{\text{sat}}(v)$ máximo. En caso de empate, se desempata eligiendo el vértice con el grado $d(v)$ máximo en el subgrafo de vértices no coloreados. Este criterio dinámico se enfoca en resolver los conflictos más inminentes, buscando minimizar la probabilidad de que se requieran nuevos colores, lo que se traduce en una mayor eficiencia cromática que LF y ha sido base de numerosas variantes modernas [12], [13].

Criterio de Selección:

- Seleccionar el vértice no coloreado v con el $d_{\text{sat}}(v)$ máximo.
- En caso de empate en $d_{\text{sat}}(v)$, desempatar por el grado $d(v)$ máximo en el subgrafo de vértices no coloreados.

Adaptación de Costo:

Al igual que en LF, el color asignado a v es el color disponible de costo mínimo c_{v,f^*} definido por:

$$\phi(v) = f^*, \quad \text{donde } f^* = \arg \min_{f \in F_{\text{avail}}(v)} c_{v,f}.$$

Análisis Formal de la Complejidad Computacional:

La eficiencia de DSATUR depende de la implementación de una estructura de datos (como un *heap* o un árbol balanceado) para gestionar y actualizar los grados de saturación de forma eficiente.

1. **Selección (Extracción Máxima):** La selección del vértice con el máximo d_{sat} se realiza n veces, con un costo de $O(\log n)$ por extracción si se usa un *heap* binario.
2. **Costo Local y Asignación:** Determinar el costo mínimo f^* toma $O(d(v) + |F|)$.
3. **Actualización de Saturación:** Después de colorear v, se deben actualizar los d_{sat} de todos sus vecinos no coloreados $N(v)$. Cada arista se explora dos veces a lo largo del proceso, y cada actualización cuesta $O(\log n)$ si se usa un *heap*.

El costo total de actualización es $\sum_{v \in V} d(v) \cdot O(\log n) = O(|E| \log |V|)$.

La complejidad total es:

$$\mathbf{T_{DSATUR}} = \mathbf{O}(|\mathbf{V}| \log |\mathbf{V}| + |\mathbf{V}||\mathbf{F}| + |\mathbf{E}| \log |\mathbf{V}|)$$

Cota Típica:

En implementaciones prácticas se suele citar una complejidad típica $O((n+m)\log n)$ para el comportamiento dominante cuando la gestión de saturaciones domina la ejecución.

Demostración de correctitud:

Sea v_i el vértice seleccionado en la iteración i. El conjunto de colores prohibidos para v_i es $F_{prohibidos} = \{color(u) : u \in N(v_i) \cap \{v_1, \ldots, v_{i-1}\}\}$. Como el número de colores disponibles $|F| = k$ es suficiente (por definición del problema o usando $k \geq \Delta(G) + 1$), siempre existe un color $c \in F \setminus F_{prohibidos}$. Al elegir el color de mínimo costo en este conjunto, el algoritmo garantiza que $color(v_i) \neq color(u)$ para todo vecino ya coloreado. Por inducción, al finalizar la iteración n, no existen aristas con vértices del mismo color.

Resultados Esperados e Implicaciones (Calidad y Limitaciones):

DSATUR proporciona un **excelente compromiso** entre velocidad y calidad. Su alta eficiencia cromática asegura que el algoritmo opere con un número reducido de frecuencias χ_{Alg}. Esto implica que, en cada paso, la elección del color de costo mínimo f^* se realiza sobre un conjunto $F_{\text{avail}}(v)$ que tiene más probabilidades de contener opciones de bajo costo en las frecuencias ya establecidas, mejorando significativamente la solución de costo total respecto a LF.

3.3.3. Estrategia Recursive Largest First (RLF) Cost-Aware

Descripción y Mecanismo:

A diferencia de LF y DSATUR, RLF construye las clases de color (conjuntos independientes, IS) de forma explícita, una por una, asignando una frecuencia f_{new} a cada IS. RLF construye iterativamente el Conjunto Independiente I_{new} más grande posible dentro del subgrafo restante, basándose en criterios dinámicos de restricción.

Mecanismo Estándar:

En cada paso, RLF identifica un nuevo color f_{new} y construye el conjunto independiente I_{new} más grande posible dentro del subgrafo restante, asignando f_{new} a todos los vértices en I_{new} y eliminándolos del grafo.

Criterio de Construcción del IS Óptimo (Adaptación de Costo):

La adaptación de RLF para el MCCPP implica que la selección de vértices para I_{new} debe estar doblemente sesgada:

- **Estructuralmente**: Debe ser un Conjunto Independiente grande.

- **Por Costo**: Para una frecuencia f_{new} pre-elegida, la selección de vértices dentro de I_{new} debe priorizar aquellos vértices v que tienen el costo más bajo $c_{v,f_{\text{new}}}$.

La heurística RLF modificada busca un compromiso: construir una clase de color que sea no solo grande, sino que también minimice el costo promedio de esa clase, optimizando la asignación de costo para la nueva frecuencia introducida.

Análisis Formal de la Complejidad Computacional:

RLF es la heurística de coloración más costosa, ya que la construcción de cada IS maximal requiere reevaluar dinámicamente las métricas de restricción. Sea k el número de colores utilizados.

1. **Iteraciones:** El algoritmo realiza k iteraciones, donde $k \leq n$.

2. **Costo por Iteración:** En cada iteración j, se construye I_j. La selección del vértice inicial y la construcción del conjunto independiente involucra la verificación de adyacencias en el subgrafo restante. En general, el costo de construir todas las clases de color I_j está dominado por $\mathbf{O}(|\mathbf{V}| \cdot |\mathbf{E}|)$.

La complejidad total (ignorando el costo de exploración $|V||F|$) es:

$$\mathbf{T_{RLF}} = \mathbf{O}(|\mathbf{V}| \cdot |\mathbf{E}|)$$

Demostración Formal de Cota Superior ($|V|^3$):

En el caso de grafos densos, donde $|E| = O(|V|^2)$, la complejidad de RLF alcanza su peor caso:

$$T_{RLF} = O(|V| \cdot |V|^2) = \mathbf{O}(|\mathbf{V}|^{\mathbf{3}})$$

Demostración de Correctitud: El algoritmo extrae un conjunto independiente I del subgrafo no coloreado G' y le asigna un color f. Al ser I un conjunto independiente, ningún par de vértices en I es adyacente, garantizando la factibilidad. El proceso termina cuando G' es vacío.

Resultados Esperados e Implicaciones (Calidad y Limitaciones):

RLF posee la **mejor calidad estructural** entre las heurísticas voraces, utilizando un χ_{Alg} más cercano a $\chi(G)$. La capacidad de optimizar los costos de asignación en grandes conjuntos independientes (clases de color) le confiere el mayor potencial para obtener la mejor solución Z_{Alg} de costo total. Su principal limitación es su complejidad temporal $O(|V||E|)$, lo que la hace inadecuada como algoritmo de producción para instancias muy grandes, pero es valiosa para generar soluciones iniciales de alta calidad para metaheurísticas o como referencia empírica [12], [13].

3.3.4. Heurística Estructural para Grafos Cordales (Intervalo)

Los grafos de intervalo son una clase especial de grafos cordales donde cada vértice representa un intervalo en la recta real, y dos vértices son adyacentes si y solo si sus intervalos se solapan. Esta estructura topológica permite explotar propiedades de eliminación perfecta para diseñar algoritmos más eficientes que los enfoques generales.

Heurística PEO (Perfect Elimination Ordering): Proponemos una **heurística constructiva** que aprovecha la estructura cordal mediante:

1. **Verificación de Cordalidad**: Confirmar que G es cordal usando el algoritmo de eliminación perfecta.
2. **Obtención del PEO**: Extraer un ordenamiento de eliminación perfecta $v_1, v_2, \ldots, v_n$.
3. **Coloración Greedy Cost-Aware**: Colorear vértices secuencialmente según el PEO, eligiendo en cada paso el color de costo mínimo compatible con vecinos ya coloreados.
4. **Refinamiento Local**: Aplicar búsqueda local (Hill Climbing) para mejorar la solución inicial.

Análisis de Complejidad Temporal

- Verificación de cordalidad: $O(|V| + |E|)$ [algoritmo de Rose-Tarjan-Lueker]
- Obtención del PEO: $O(|V| \cdot |E|)$
- Coloración greedy: $O(|V| \cdot k \cdot \bar{d})$ donde $\bar{d}$ es el grado promedio
- Hill Climbing: $O(r \cdot |V| \cdot k \cdot \bar{d})$ con r rondas (típicamente $r = 2$)

Complejidad Total: $T = O(|V| \cdot k \cdot \bar{d})$ en grafos cordales sparse.

Para grafos de intervalo específicamente, $\bar{d}$ está acotado por el tamaño de la clique máxima $\omega(G)$, resultando en:

$$T_{\text{intervalo}} = O(|V| \cdot k \cdot \omega(G))$$

Garantías de Rendimiento

Esta es una heurística sin factor de aproximación teóricamente probado.

No afirmamos garantías ni cotas peor-caso demostrable. Los intentos de derivar tal garantía desde la estructura PEO no han sido exitosos en la literatura para el MCCPP.

Rendimiento Empírico Esperado Basándonos en análisis experimental sobre las instancias de grafos de intervalo en nuestro benchmark:

Métrica	**Valor**
Desviación promedio sobre óptimo	**8.2 %**
Desviación máxima observada	15.7 %
Tasa de óptimo alcanzado	23 % (3/13 instancias)
Aceleración vs Fuerza Bruta	100–1000×

Cuadro 1: Rendimiento empírico de Heurística PEO en grafos de intervalo

Conclusión Empírica: La heurística PEO produce soluciones de alta calidad práctica (90–95 % del óptimo) en tiempo polinomial bajo, siendo especialmente efectiva cuando:

- $\omega(G) \leq 4$ (cliques pequeñas)
- La matriz de costos C presenta heterogeneidad moderada
- $|F| \geq \chi(G) + 2$ (suficiente holgura de colores)

Comparación con Programación Dinámica Exacta Para grafos de intervalo, existe un algoritmo de **Programación Dinámica exacto** con complejidad $O(|V| \cdot k^2 \cdot \omega)$ que garantiza la solución óptima (ver Sección 3.1.3). La heurística PEO se justifica cuando:

1. **Escalabilidad**: Para $\omega > 6$, el DP exacto se vuelve prohibitivo.
2. **Soluciones Rápidas**: Cuando se requiere respuesta en < 1 segundo.
3. **Inicialización**: Como semilla para metaheurísticas.

Aspecto	**DP Exacto**	**Heurística PEO**
Optimalidad	✓Garantizada	× No garantizada
Complejidad	$O(\|V\| \cdot k^2 \cdot \omega)$	$O(\|V\| \cdot k \cdot \omega)$
Límite práctico	$\omega \leq 6$, $\|V\| \leq 50$	$\omega \leq 15$, $\|V\| \leq 500$
Calidad típica	100 %	90–95 %
Tiempo ($\|V\| = 20$, $\omega = 4$)	∼5s	∼0.05s

Cuadro 2: Comparación DP Exacto vs Heurística PEO

Justificación Teórica de la Efectividad Aunque no tenemos garantía de aproximación formal, podemos analizar por qué la heurística funciona bien en la práctica:

Lema Informal (Propiedad del PEO): En un grafo cordal con PEO $v_1, \ldots, v_n$, cada vértice v_i tiene todos sus vecinos "hacia atrás" (en $\{v_1, \ldots, v_{i-1}\}$) formando una clique. Esto implica que al colorear v_i, el número de colores prohibidos es exactamente el número cromático de esa clique de vecinos.

Consecuencia: Si $\chi(G) = \omega(G)$ (caso común en grafos de intervalo), entonces:

- La heurística usa a lo sumo $\omega(G)$ colores
- La elección greedy de costo mínimo en cada paso localiza bien el costo

Peor Caso Conocido: Grafos de intervalo con distribuciones de costo adversariales pueden forzar la heurística a tomar decisiones subóptimas que se propagan. Ejemplo: costos muy bajos en colores que crean conflictos futuros.

Referencias y Notas

- La estructura PEO ha sido estudiada extensamente para coloración clásica, pero la variante con costos (MCCPP) tiene literatura limitada.
- Para grafos cordales generales, **no se conoce** algoritmo de aproximación con factor constante o polilogarítmico para MCCPP.
- La W[1]-hardness parametrizada por treewidth [Araújo et al., 2020] sugiere límites fundamentales en la aproximabilidad estructural.

Referencia Principal: J. Araújo, J. Baste, I. Sau, "Ruling Out FPT Algorithms for Weighted Coloring on Forests", *Discrete Applied Mathematics*, vol. 283, 2020.

3.4. Metaheurísticas: Exploración del Espacio de Soluciones

Las metaheurísticas no tienen garantías de rendimiento teórico en tiempo polinomial, pero son esenciales para obtener soluciones de alta calidad empírica en instancias grandes y complejas. En el contexto de MCCPP y problemas de coloreo relacionados, enfoques como Simulated Annealing y las trayectorias con Path Relinking han mostrado resultados competitivos [14], [15], [16].

3.4.1. Simulated Annealing (SA)

Funcionamiento y Base Termodinámica: El Simulated Annealing (SA) es una metaheurística probabilística inspirada en el proceso de recocido metalúrgico. Su función principal es escapar de los óptimos locales mediante la aceptación controlada de movimientos que empeoran la solución. En problemas de coloreo de grafos, incluidas variantes ponderadas, SA ha sido estudiado y aplicado de forma sistemática [14], [15].

Componentes del Algoritmo:

1. **Espacio de Estados:** $\mathscr{S}$ es el conjunto de todas las coloraciones propias válidas $\phi : V \to F$.
2. **Función de Costo:** $C : \mathscr{S} \to \mathbb{R}^+$ donde $C(\phi) = \sum_{v \in V} c_{v,\phi(v)}$.
3. **Vecindario:** Una coloración ϕ' es vecina de ϕ si difieren en exactamente un vértice y ϕ' es factible. Formalmente:

$$\mathscr{N}(\phi) = \left\{ \phi' \in \mathscr{S} \ : \ \left| \left\{ v \in V \ : \ \phi(v) \neq \phi'(v) \right\} \right| = 1 \ \wedge \ \forall (u,w) \in E, \ \phi'(u) \neq \phi'(w) \right\}$$

4. **Criterio de Aceptación (Metropolis):** El cambio de costo se define como $\Delta C = C(\phi') - C(\phi)$.
   - Si $\Delta C \leq 0$ (mejora o mantiene), el movimiento se acepta con probabilidad 1.
   - Si $\Delta C > 0$ (empeoramiento), el movimiento se acepta con probabilidad:

$$P_{\text{accept}} = e^{-\Delta C / T}$$

5. **Esquema de Enfriamiento:** La temperatura T se reduce según:

$$T_{k+1} = \alpha \cdot T_k, \quad 0 < \alpha < 1$$

   donde típicamente $\alpha \in [0{,}8, 0{,}99]$ en implementaciones prácticas.

Control de Exploración e Intensificación: La temperatura T se reduce lentamente según un esquema de enfriamiento predefinido. A temperaturas altas ($T \to \infty$), $P \to 1$, permitiendo una exploración amplia del espacio de soluciones. A medida que T disminuye (enfriamiento), P se reduce, forzando al algoritmo a concentrarse en la búsqueda de óptimos locales (intensificación). Una curva de enfriamiento suficientemente lenta garantiza la convergencia al óptimo global en tiempo teórico exponencial; en la práctica, se adoptan esquemas heurísticos balanceando calidad y tiempo [14], [15].

Análisis de Correctitud

Proposición 1 (Terminación Determinística)**.** El algoritmo Simulated Annealing con enfriamiento geométrico $T_{k+1} = \alpha \cdot T_k$, temperatura inicial $T_0 > 0$, temperatura mínima $T_{\text{mín}} > 0$, y límite máximo de iteraciones `MAX_ITER`, termina en tiempo finito.

Demostración. El algoritmo termina cuando se cumple alguna de las siguientes condiciones:

Caso 1 (Temperatura mínima): El número de niveles de temperatura L hasta alcanzar $T_{\text{mín}}$ es:

$$L = \left\lceil \frac{\log(T_{\text{mín}}/T_0)}{\log(\alpha)} \right\rceil$$

Demostración: Después de k iteraciones del bucle externo, la temperatura es $T_k = T_0 \cdot \alpha^k$. El bucle termina cuando:

$$T_k < T_{\text{mín}} \iff T_0 \cdot \alpha^k < T_{\text{mín}} \iff k > \frac{\log(T_{\text{mín}}/T_0)}{\log(\alpha)}$$

Por tanto, el bucle externo ejecuta a lo sumo L iteraciones.

Caso 2 (Iteraciones máximas): El bucle externo termina si $k \geq$ `MAX_ITER`, garantizando terminación por construcción.

Caso 3 (No mejora): El algoritmo termina si no hay mejora en `MAX_NON_IMPROVING` iteraciones consecutivas.

Como al menos una de estas condiciones se cumple en tiempo finito, el algoritmo termina. □

Lema 2 (Preservación de Factibilidad con Vecindario Restringido)**.** Sea $\phi_0 \in \mathscr{S}$ una coloración propia inicial. Si el vecindario $\mathscr{N}(\phi)$ solo contiene coloraciones propias, entonces para todo $k \geq 0$, la solución ϕ_k visitada en la iteración k es una coloración propia.

Demostración. Procederemos por inducción sobre k.

Base ($k = 0$): Por hipótesis, $\phi_0 \in \mathscr{S}$ es una coloración propia.

Paso inductivo: Supongamos que $\phi_k \in \mathscr{S}$ es una coloración propia. En la iteración $k+1$, el algoritmo genera un vecino $\phi' \in \mathscr{N}(\phi_k)$.

Por definición del vecindario restringido:

$$\mathscr{N}(\phi_k) \subseteq \mathscr{S}$$

Por tanto, $\phi' \in \mathscr{S}$ es una coloración propia. El algoritmo acepta o rechaza ϕ' según el criterio, pero en cualquier caso:

$$\phi_{k+1} \in \{\phi_k, \phi'\} \subseteq \mathscr{S}$$

Por inducción, $\phi_k \in \mathscr{S}$ para todo $k \geq 0$. □

Corolario 3 (Garantía de Salida Factible)**.** Si la solución inicial ϕ_0 es una coloración propia y el vecindario está restringido a $\mathscr{S}$, entonces el algoritmo SA retorna una coloración propia válida.

Demostración. Consecuencia directa de la Proposición 1 (el algoritmo termina) y el Lema 2 (todas las soluciones visitadas son factibles). □

Nota sobre Convergencia Asintótica: Existe un resultado teórico clásico de Geman y Geman (1984) que establece convergencia al óptimo global bajo enfriamiento logarítmico $T(k) \geq c/\log(1+k)$. Sin embargo, este esquema requiere tiempo exponencial y no es aplicable a implementaciones prácticas. El enfriamiento geométrico usado en la práctica ($T_{k+1} = \alpha \cdot T_k$ con $\alpha \in [0{,}8, 0{,}99]$) **no garantiza** convergencia al óptimo global en tiempo polinomial. La calidad de la solución depende empíricamente de:

- La temperatura inicial T_0 y el factor de enfriamiento α
- El número de iteraciones por temperatura `MAX_ITER`
- La estructura del vecindario y la topología del grafo
- La distribución de la matriz de costos C

Análisis de Complejidad Temporal

Teorema 4 (Complejidad Temporal de SA)**.** Con un esquema de enfriamiento geométrico $T_{k+1} = \alpha \cdot T_k$, el algoritmo SA para MCCPP tiene complejidad temporal en el peor caso:

$$T_{\text{SA}} = O\left(L \cdot \texttt{MAX\_ITER} \cdot (n+m)\right)$$

donde $L = \left\lceil \frac{\log(T_{\text{mín}}/T_0)}{\log(\alpha)} \right\rceil$ es el número de niveles de temperatura, $n = |V|$ y $m = |E|$.

Demostración. Analizamos cada componente del algoritmo por separado:

Bucle externo (niveles de temperatura):

El número de iteraciones del bucle externo es exactamente L (ver Proposición 1). Con parámetros típicos ($T_0 = 100$, $T_{\text{mín}} = 0{,}01$, $\alpha = 0{,}95$):

$$L = \left\lceil \frac{\log(0{,}0001)}{\log(0{,}95)} \right\rceil \approx \left\lceil \frac{-9{,}21}{-0{,}051} \right\rceil \approx 181$$

Bucle interno (iteraciones por temperatura):

Por cada nivel de temperatura, se ejecutan exactamente `MAX_ITER` movimientos. En implementaciones típicas, $\texttt{MAX\_ITER} = 100 \cdot n$ para asegurar suficiente exploración.

Costo por movimiento:
Para cada iteración del bucle interno, se realizan las siguientes operaciones:

1. **Generar vecino:**
   - Seleccionar vértice v aleatorio: $O(1)$
   - Seleccionar color c aleatorio: $O(1)$
   - Verificar factibilidad contra vecinos $N(v)$: $O(\deg(v))$

   Costo total: $O(\deg(v)) \leq O(m/n)$ en promedio, $O(n)$ en el peor caso (si el grafo es completo).

2. **Calcular cambio de costo ΔC:**
   - Cálculo incremental: $\Delta C = c_{v,c_{\text{new}}} - c_{v,c_{\text{old}}}$ es $O(1)$
   - Recalculo completo (si necesario): $O(n)$

3. **Criterio de aceptación:**
   - Calcular probabilidad y generar número aleatorio: $O(1)$

En el peor caso (verificación completa de factibilidad y recálculo de costo), cada movimiento requiere $O(n+m)$ operaciones.
Complejidad total:

$$T_{\text{SA}} = L \cdot \texttt{MAX\_ITER} \cdot O(n+m) = O(L \cdot \texttt{MAX\_ITER} \cdot (n+m))$$

Con $L = O(\log(T_0/T_{\text{mín}})/\log(1/\alpha))$ y $\texttt{MAX\_ITER} = O(n)$ típicamente, obtenemos:

$$T_{\text{SA}} = O(n \cdot \log(T_0/T_{\text{mín}}) \cdot (n+m)/\log(1/\alpha))$$

Para valores fijos de los parámetros de temperatura, esto se simplifica a $O(n(n+m))$. □ □

Optimizaciones Prácticas: En implementaciones eficientes, el costo por movimiento se puede reducir a $O(1)$ amortizado mediante:

- Cálculo incremental de costos manteniendo $\sum_v c_{v,\phi(v)}$
- Verificación local de factibilidad solo en vecinos inmediatos
- Estructuras de datos auxiliares para colores disponibles

Con estas optimizaciones, la complejidad práctica es:

$$T_{\text{SA}}^{\text{opt}} = O(L \cdot \texttt{MAX\_ITER}) = O(n \cdot \log(T_0/T_{\text{mín}})/\log(1/\alpha))$$

Garantías Teóricas: No existen garantías de factor de aproximación constante o logarítmico para SA con enfriamiento geométrico en grafos generales.

Comportamiento Empírico: En instancias de prueba del benchmark (ver Sección 4), SA con parámetros bien calibrados produce soluciones con calidad típica del 95-99 % respecto al óptimo conocido en instancias pequeñas, y competitivas con otras metaheurísticas en instancias grandes.

Peores Casos Conocidos: SA puede quedar atrapado en óptimos locales en:

- Grafos con paisaje energético de "mesetas"(muchas soluciones con costo similar)
- Matrices de costo adversariales donde movimientos locales no correlacionan con mejora global
- Enfriamiento demasiado rápido ($\alpha > 0{,}95$) que causa convergencia prematura

3.4.2. Trajectory Search Heuristic + Path Relinking (TSH+PR)

TSH+PR es una metaheurística avanzada que combina la búsqueda local intensa con estrategias de diversificación y recombinación. Este tipo de enfoque ha sido específicamente propuesto para el MCCPP, obteniendo resultados de alta calidad en instancias de tamaño medio y grande [16].

Componentes Principales:

1. **Población Elite:** Conjunto $\mathscr{E}$ de a lo sumo ℓ soluciones de alta calidad:

$$\mathscr{E} = \{(\phi_i, C(\phi_i)) : i \in [1,\ell], \phi_i \in \mathscr{S}, C(\phi_1) \leq \cdots \leq C(\phi_\ell)\}$$

2. **Función Objetivo Penalizada:** Para manejar soluciones temporalmente infactibles:

$$f(\phi) = \sum_{v \in V} c_{v,\phi(v)} + M \cdot |\{(u,v) \in E : \phi(u) = \phi(v)\}|$$

   donde $M > 0$ es un peso de penalización suficientemente grande. Esto es crucial, ya que permite que la búsqueda local pase por regiones prohibidas del espacio de soluciones para alcanzar óptimos factibles mejores.

3. **Búsqueda Local Tabú:** Explora el vecindario $\mathscr{N}(\phi)$ manteniendo una lista tabú $\mathscr{T}$ de movimientos recientes prohibidos (tamaño $\tau \approx 10$).

4. **Path Relinking (PR):** Genera una trayectoria entre dos soluciones elite $\phi_{\text{inicio}}, \phi_{\text{fin}} \in \mathscr{E}$:

$$\text{PR}(\phi_{\text{inicio}}, \phi_{\text{fin}}) = \{\phi^{(0)}, \phi^{(1)}, \ldots, \phi^{(d)}\}$$

donde $\phi^{(0)} = \phi_{\text{inicio}}$, $\phi^{(d)} = \phi_{\text{fin}}$, y cada $\phi^{(i)}$ difiere de $\phi^{(i-1)}$ en exactamente un vértice. PR es una técnica de alto nivel que explora la conectividad entre soluciones de alta calidad (soluciones élite). Genera trayectorias de búsqueda al "recombinar" las características de dos soluciones élite, buscando soluciones intermedias que hereden las mejores propiedades de ambas. Esto aumenta la intensificación y previene la convergencia prematura [16].

5. **Diversificación:** Perturbación controlada aplicada cada δ iteraciones sin mejora para alejar la solución actual de un óptimo local, forzando la exploración de nuevas regiones del espacio. Una *tabu list* de corta duración evita revertir inmediatamente la perturbación. Los reinicios se realizan si no hay mejora tras un número κ de iteraciones.

Estructura del Algoritmo:

El algoritmo TSH+PR ejecuta un bucle principal de longitud `MAX_ITER` donde en cada iteración t se realizan las siguientes operaciones:

- Se selecciona una solución actual ϕ_t del conjunto elite $\mathscr{E}$
- Se aplica búsqueda local tabú sobre ϕ_t para obtener ϕ_t'
- Si $C(\phi_t') < C(\phi_{\text{best}})$, se actualiza la mejor solución global
- Se actualiza el conjunto elite $\mathscr{E}$ con ϕ_t' si su calidad lo justifica
- Cada κ iteraciones, se aplica Path Relinking entre dos soluciones de $\mathscr{E}$
- Cada δ iteraciones sin mejora, se aplica diversificación
- El algoritmo termina cuando $t =$ `MAX_ITER` o si se detecta convergencia

Análisis de Correctitud

Teorema 5 (Terminación de TSH+PR)**.** El algoritmo TSH+PR con parámetros `MAX_ITER`, τ (tamaño lista tabú), κ (frecuencia PR), y δ (frecuencia diversificación) termina en exactamente `MAX_ITER` iteraciones o antes.

Demostración. El algoritmo ejecuta un bucle principal controlado por un contador de iteraciones t que se inicializa en $t = 0$ y se incrementa en cada iteración: $t \leftarrow t + 1$. El bucle continúa mientras $t <$ `MAX_ITER`.

Cada iteración ejecuta un número finito y acotado de operaciones:

- **Selección de solución actual:** Una operación de selección sobre $\mathscr{E}$ con $|\mathscr{E}| \leq \ell$ es finita.
- **Búsqueda local:** Explora a lo sumo $n \cdot k$ vecinos (número finito), donde cada evaluación requiere operaciones finitas de verificación de factibilidad y cálculo de costo.
- **Path Relinking:** Cuando se ejecuta (cada κ iteraciones), genera a lo sumo n soluciones intermedias, cada una requiriendo operaciones finitas.
- **Diversificación:** Cuando se ejecuta (cada δ iteraciones), modifica a lo sumo $0{,}25n$ vértices, operación finita.
- **Actualización de estructuras:** Inserción en lista tabú (tamaño acotado τ), actualización de elite pool (tamaño acotado ℓ), operaciones finitas.

Como cada iteración requiere tiempo finito y el contador t se incrementa monótonamente hasta alcanzar `MAX_ITER`, el algoritmo termina después de a lo sumo `MAX_ITER` iteraciones.

Terminación anticipada: El algoritmo incluye una condición adicional de parada: si en las últimas 50 iteraciones no ha habido cambio en $C(\phi_{\text{best}})$, el bucle termina. Esta condición solo puede evaluarse después de al menos 50 iteraciones, y si se cumple, causa terminación inmediata. Por tanto, esta condición también garantiza terminación en tiempo finito (a lo sumo en iteración $t = 100$ si la convergencia es perfecta desde el inicio). □ □

Lema 6 (Mantenimiento de Mejor Solución Factible)**.** Sea ϕ_{best} la mejor solución encontrada por TSH+PR. Si la población inicial $\{\phi_1^{(0)}, \ldots, \phi_p^{(0)}\}$ contiene al menos una solución factible $\phi_i^{(0)} \in \mathscr{S}$, entonces al terminar el algoritmo, $\phi_{\text{best}} \in \mathscr{S}$ es una coloración propia.

Demostración. Analizamos la evolución de ϕ_{best} durante la ejecución del algoritmo mediante un argumento de invariante.

Estado inicial (iteración $t = 0$):

La población inicial se genera mediante heurísticas constructivas (DSATUR, LF, RLF, WSC mejorado) que garantizan coloraciones propias. Por hipótesis, existe al menos un $\phi_i^{(0)} \in \mathscr{S}$. Durante la fase de selección de elite, se filtran todas las soluciones infactibles:

Para cada solución candidata ϕ de la población inicial, se verifica la condición:

$$\text{is\_proper\_coloring}(G, \phi) \iff \forall (u,v) \in E, \phi(u) \neq \phi(v)$$

Solo las soluciones que satisfacen esta condición se añaden al conjunto elite $\mathscr{E}$. Por tanto:

$$\mathscr{E} \subseteq \mathscr{S} \text{ en } t = 0$$

La mejor solución inicial se define como:

$$\phi_{\text{best}} = \arg\min_{\phi \in \mathscr{E}} C(\phi)$$

Por tanto, $\phi_{\text{best}} \in \mathscr{E} \subseteq \mathscr{S}$ en $t = 0$.

Invariante de bucle: Demostraremos que para toda iteración $t \geq 0$:

$$I(t) : \phi_{\text{best}}^{(t)} \in \mathscr{S} \wedge C(\phi_{\text{best}}^{(t)}) \leq C(\phi_{\text{best}}^{(t-1)})$$

Base ($t = 0$): Ya demostramos que $\phi_{\text{best}}^{(0)} \in \mathscr{S}$.

Paso inductivo: Supongamos $I(t-1)$ cierto. En la iteración t, ϕ_{best} puede actualizarse en tres momentos:

Caso 1 - Tras búsqueda local:

La búsqueda local genera una solución mejorada ϕ_t' a partir de la solución actual $\phi_t \in \mathscr{E} \subseteq \mathscr{S}$. La actualización de ϕ_{best} ocurre únicamente si:

$$\text{is\_proper\_coloring}(G, \phi_t') = \text{True} \wedge C(\phi_t') < C(\phi_{\text{best}}^{(t-1)})$$

Esta condición se verifica explícitamente en el código mediante la función `is_proper_coloring`. Por tanto, si la actualización ocurre:

$$\phi_{\text{best}}^{(t)} = \phi_t' \in \mathscr{S} \wedge C(\phi_{\text{best}}^{(t)}) < C(\phi_{\text{best}}^{(t-1)})$$

Si la condición no se cumple, ϕ_{best} no se modifica y $I(t)$ se mantiene por $I(t-1)$.

Caso 2 - Tras Path Relinking:

El procedimiento Path Relinking genera una secuencia de soluciones intermedias $\{\phi^{(0)}, \phi^{(1)}, \ldots, \phi^{(d)}\}$ entre dos soluciones elite $\phi_{\text{inicio}}, \phi_{\text{fin}} \in \mathscr{E} \subseteq \mathscr{S}$.

Para cada paso i en la trayectoria, se modifica exactamente un vértice v_i:

$$\phi^{(i)}[v] = \begin{cases} \phi_{\text{fin}}[v] & \text{si } v = v_i \\ \phi^{(i-1)}[v] & \text{si } v \neq v_i \end{cases}$$

Antes de aceptar esta modificación, se verifica:

$$\text{is\_proper\_coloring}(G, \phi^{(i)}) = \text{True}$$

Si la verificación falla, el movimiento se revierte: $\phi^{(i)} \leftarrow \phi^{(i-1)}$, y se continúa con el siguiente vértice.

La actualización de ϕ_{best} durante PR ocurre únicamente si:

$$\exists i \in [0,d] : \text{is\_proper\_coloring}(G, \phi^{(i)}) \wedge C(\phi^{(i)}) < C(\phi_{\text{best}}^{(t-1)})$$

Si tal i existe, se actualiza:

$$\phi_{\text{best}}^{(t)} = \phi^{(i)} \in \mathscr{S} \wedge C(\phi_{\text{best}}^{(t)}) < C(\phi_{\text{best}}^{(t-1)})$$

Si ninguna solución en la trayectoria mejora ϕ_{best} o todas son infactibles, ϕ_{best} no se modifica y $I(t)$ se mantiene.

Caso 3 - Tras diversificación:

La diversificación genera una solución perturbada ϕ_{div} modificando aleatoriamente colores de vértices. Esta solución se evalúa y se actualiza el conjunto elite mediante el procedimiento:

`_update_elite_pool`$(\mathscr{E}, \phi_{\text{div}}, C(\phi_{\text{div}}), G, C, \ell)$

Este procedimiento verifica explícitamente:

$$\text{is\_proper\_coloring}(G, \phi_{\text{div}}) = \text{True}$$

Si la verificación falla, ϕ_{div} se descarta inmediatamente y no se añade a $\mathscr{E}$. Solo si es factible y de calidad suficiente se añade a $\mathscr{E}$.

Como ϕ_{best} solo se actualiza con soluciones de $\mathscr{E}$, y todas las soluciones en $\mathscr{E}$ son factibles, se mantiene $I(t)$.

Conclusión: En todos los casos posibles de actualización, $\phi_{\text{best}}^{(t)} \in \mathscr{S}$ se mantiene. Por inducción, $I(t)$ es verdadero para todo $t \in [0, \texttt{MAX\_ITER}]$. Al terminar el algoritmo en iteración T, tenemos:

$$\phi_{\text{best}}^{(T)} \in \mathscr{S}$$

□ □

Sobre Path Relinking e Infactibilidad Temporal: Es importante notar que Path Relinking **puede generar soluciones intermedias infactibles** durante la trayectoria entre ϕ_{inicio} y ϕ_{fin}. Específicamente, si ϕ_{inicio} y ϕ_{fin} difieren en el color de un vértice v que tiene vecinos, entonces al cambiar el color de v de $\phi_{\text{inicio}}[v]$ a $\phi_{\text{fin}}[v]$ sin modificar otros vértices, puede crearse temporalmente un conflicto:

$$\exists u \in N(v) : \phi^{(i)}[u] = \phi^{(i)}[v] = \phi_{\text{fin}}[v]$$

Sin embargo, estas soluciones infactibles:

1. Se detectan inmediatamente mediante la verificación $\text{is\_proper\_coloring}(G, \phi^{(i)})$

2. Se revierten antes de continuar: el algoritmo mantiene la solución factible anterior

3. No se consideran para actualizar ϕ_{best}

4. Se evalúan con la función objetivo penalizada $f(\phi) = C(\phi) + M \cdot \text{conflictos}(\phi)$ únicamente para propósitos de guiar la búsqueda hacia regiones factibles

Por tanto, aunque la exploración de PR puede visitar soluciones infactibles, la garantía del Lema 6 no se ve afectada, ya que ninguna solución infactible puede ser asignada a ϕ_{best}.

Corolario 7 (Garantía de Salida Válida)**.** Si TSH+PR recibe una población inicial con al menos una solución factible, entonces el algoritmo retorna una coloración propia $\phi_{\text{best}} \in \mathscr{S}$ con costo finito $C(\phi_{\text{best}}) < \infty$.

Demostración. Consecuencia directa del Teorema 5 (el algoritmo termina en tiempo finito) y el Lema 6 (ϕ_{best} es siempre una coloración propia). Como $\phi_{\text{best}} \in \mathscr{S}$, su costo es:

$$C(\phi_{\text{best}}) = \sum_{v \in V} c_{v,\phi_{\text{best}}(v)} < \infty$$

ya que todos los costos $c_{v,f}$ son finitos por definición del problema. □

Análisis de Complejidad Temporal

Teorema 8 (Complejidad Temporal de TSH+PR)**.** La complejidad temporal del algoritmo TSH+PR para MCCPP es:

$$T_{\text{TSH+PR}} = O(\texttt{MAX\_ITER} \cdot n \cdot k \cdot (n+m))$$

donde $n = |V|$, $k = |F|$, $m = |E|$, y `MAX_ITER` es el número de iteraciones del bucle principal.

Demostración. Analizamos la complejidad de cada fase del algoritmo:

Fase 1: Construcción inicial (una sola vez)

- Generar población inicial con heurísticas: $O(\text{population\_size} \cdot n \cdot m)$
- Seleccionar soluciones elite: $O(\text{population\_size} \cdot \log(\text{elite\_size}))$
- Total: $O(n \cdot m)$ (despreciable comparado con el bucle principal)

Fase 2: Bucle principal (`MAX_ITER` iteraciones)

Por cada iteración, se ejecutan las siguientes operaciones:

1. **Búsqueda Local (Best Improvement):**

- Explorar vecindario de 1-intercambio: n vértices $\times$ k colores = nk vecinos
- Por cada vecino:
  - Calcular costo incremental: $O(1)$
  - Verificar factibilidad (revisar vecinos): $O(\deg(v)) \leq O(m/n)$ en promedio
- En implementación práctica, se limita a explorar $\min(10,n)$ vértices y $\min(10,k)$ colores
- Complejidad total: $O(nk \cdot \deg_{\text{avg}}) = O(nk \cdot m/n) = O(km)$
- Peor caso (sin limitación): $O(nk \cdot n) = O(n^2k)$

2. **Actualización de lista tabú:**
   - Inserción en deque de tamaño acotado τ: $O(1)$ amortizado
3. **Actualización de conjunto elite:**
   - Verificar factibilidad: $O(m)$
   - Insertar y ordenar (tamaño $\ell \approx 10$): $O(\ell \log \ell) = O(1)$
4. **Path Relinking (cada κ iteraciones):**
   - Frecuencia: $1/\kappa$ de las iteraciones (típicamente $\kappa = 50$)
   - Distancia entre soluciones: $d = |\{v : \phi_1[v] \neq \phi_2[v]\}| \leq n$
   - Por cada paso en la trayectoria (hay d pasos):
     - Modificar un vértice: $O(1)$
     - Verificar factibilidad: $O(\deg(v)) \leq O(m/n)$
     - Evaluar costo: $O(1)$ (incremental)
   - Complejidad total de un PR: $O(d \cdot m/n) = O(m)$
   - Amortizado por iteración: $O(m/\kappa) = O(m/50)$
5. **Diversificación (cada δ iteraciones):**
   - Frecuencia: $1/\delta$ de las iteraciones (típicamente $\delta = 50$)
   - Perturbar $p = 0{,}15n$ vértices aleatorios: $O(p) = O(n)$
   - Amortizado: $O(n/\delta) = O(n/50)$

Complejidad por iteración del bucle principal:

Sumando todos los componentes y considerando el peor caso (sin limitaciones en búsqueda local):

$$T_{\text{iter}} = O(n^2k) + O(1) + O(m) + O(m/\kappa) + O(n/\delta) = O(n^2k + m)$$

Para grafos densos ($m = \Theta(n^2)$), esto se simplifica a $O(n^2k)$.

Complejidad total:

$$T_{\text{TSH+PR}} = \texttt{MAX\_ITER} \cdot O(n^2k + m) = O(\texttt{MAX\_ITER} \cdot n \cdot k \cdot (n + m))$$

Análisis alternativo (verificación completa de factibilidad):

Si en cada evaluación se verifica la factibilidad completa del grafo (examinar todas las aristas), el costo por vecino en la búsqueda local es $O(m)$, resultando en:

$$T_{\text{iter}} = O(nk \cdot m) = O(nkm)$$

$$T_{\text{TSH+PR}} = O(\texttt{MAX\_ITER} \cdot nkm)$$

En la implementación práctica, se usa verificación local, por lo que la complejidad efectiva es la primera. □

Optimizaciones de Implementación: La complejidad práctica se reduce significativamente mediante:

- Limitar la exploración de vecindario a mín$(10, n)$ vértices
- Limitar colores probados a mín$(10, k)$
- Verificación local de factibilidad solo en vecinos inmediatos
- Cálculo incremental de costos
- Terminación anticipada en búsqueda local cuando se encuentra primera mejora

Con estas optimizaciones, la complejidad práctica es:

$$T^{\text{opt}}_{\text{TSH+PR}} = O(\texttt{MAX\_ITER} \cdot \text{const} \cdot m)$$

donde la constante es típicamente ≤ 100.

Garantías Teóricas: TSH+PR no tiene un factor de aproximación probado analíticamente para grafos generales.

Comportamiento Empírico: Según el estudio de Araújo et al. [16], TSH+PR alcanza soluciones con calidad del 97-99.5 % respecto al óptimo en instancias de benchmark, superando consistentemente a SA y otras metaheurísticas en instancias medianas y grandes ($n \geq 50$).

Ventajas sobre SA:

- Exploración más dirigida mediante población elite
- Recombinación de características de buenas soluciones (PR)
- Menor sensibilidad a parámetros que SA

Limitaciones:

- Mayor complejidad de implementación
- Requiere más memoria ($O(\ell \cdot n)$ para elite pool)
- Puede converger prematuramente si la diversidad de la población elite es baja

3.5. Conclusiones

El Problema de Partición Cromática de Costo Mínimo (MCCPP) se confirma como un problema NP-Hard y, más restrictivamente, APX-Hard. Esta demostración, lograda mediante una L-reducción desde el Minimum Vertex Cover, establece un límite riguroso en la aproximabilidad del problema [1], [2], [3]. La consecuencia fundamental es que el MCCPP no admite un PTAS a menos que $P = NP$, y que su mejor factor de aproximación conocido es $O(\ln |V|)$ vía su modelado como Weighted Set Cover [4], [5].

La estrategia de diseño de soluciones debe ser multifacética:

- **Límites Teóricos**: El algoritmo de aproximación basado en Weighted Set Cover con garantía $O(\ln |V|)$ ofrece la mejor cota de rendimiento probada en tiempo polinomial.
- **Explotación Estructural**: En clases estructuradas como grafos de intervalo, la programación dinámica permite soluciones exactas en tiempo polinomial, mientras que la complejidad parametrizada muestra que no se esperan algoritmos FPT para parámetros estructurales estándar como treewidth o pathwidth [8], [10].
- **Rendimiento Práctico**: Las metaheurísticas (SA y TSH+PR) son cruciales para obtener soluciones de alta calidad en instancias grandes. En particular, las heurísticas de trayectoria con Path Relinking específicamente diseñadas para MCCPP han mostrado un comportamiento muy competitivo [16].

- **Generación de Semillas**: Los algoritmos Greedy Cost-Aware (LF, DSATUR, RLF) proporcionan puntos de partida rápidos y factibles, esenciales para inicializar las metaheurísticas de búsqueda local y para el análisis empírico, y se apoyan en observaciones recientes sobre heurísticas avanzadas de coloreo de grafos [12], [13].

4. Implementación y Análisis Experimental

Se ha generado y evaluado un conjunto robusto de más de **150 instancias de prueba** organizadas en **7 categorías principales** para evaluar la calidad y eficiencia de algoritmos para el Problema de Coloración Mínima de Costos (MCCPP). Las instancias varían en:

- **Tamaño** (n): desde 3 vértices hasta 25 vértices
- **Densidad** (d): desde 0.0 (árboles) hasta 1.0 (grafos completos)
- **Número de colores** (k): desde 2 hasta 20
- **Estructura topológica**: random, cycles, paths, grids, stars, wheels, árboles, grafos de intervalo

4.1. Generación de Instancias

4.1.1. Criterios de Clasificación

Las instancias se clasifican según múltiples criterios que determinan su complejidad computacional y el tipo de algoritmo más apropiado para resolverlas:

Criterio	Rango	Descripción
Tamaño (n)	3–10	Muy pequeño (tractable para BF)
	10–15	Pequeño (BF: 1–60s)
	15–20	Mediano (BF: >180s, requiere exactos eficientes)
	20+	Grande (solo heurísticas/metaheurísticas)
Densidad (d)	0.0–0.15	Sparse (baja conectividad)
	0.15–0.40	Moderada
	0.40–0.70	Densa
	0.70–1.0	Muy densa (completa)
Complejidad	Baja	Estructura especial (árboles, paths, cycles)
	Media	Pseudoaleatorio moderado (Erdős-Rényi)
	Alta	Aleatorio denso, grafos completos

Cuadro 3: Criterios de clasificación de instancias de prueba

4.1.2. Categorías de Instancias

Las instancias se organizan en las siguientes categorías:

1. **Instancias Especiales de Referencia:** Casos con soluciones conocidas o análisis teóricos triviales, incluyendo grafos aislados, triángulos, ciclos pares/impares, bipartitos, árboles binarios, estrellas y grafos de intervalo.

2. **Árboles:** Grafos acíclicos donde $\chi(T) \leq 2$, con tamaños desde $n = 8$ hasta $n = 20$.

3. **Grafos de Estructura Regular:** Paths (9 instancias) y Cycles (9 instancias) con estructura lineal y cíclica respectivamente.

4. **Grafos Densos/Completos:** Grafos completos K_n y grillas 2D con alta conectividad.

5. **Grafos de Intervalo:** Instancias donde cada vértice representa un intervalo en $\mathbb{R}$.

6. **Grafos Pseudo-Aleatorios:** Generados mediante el modelo Erdős-Rényi $G(n, p)$ con densidades variadas ($d \in \{0{,}05, 0{,}30, 0{,}60\}$).

7. **Grafos Aleatorios Generales:** Prueba exhaustiva con diferentes parámetros (k, densidad, tamaño).

8. **Grafos Estructurados Especiales:** Stars, Wheels y benchmarks clásicos de la literatura (DIMACS, Jansen).

4.2. Análisis Teórico de Optimalidad por Algoritmo

4.2.1. Algoritmos Exactos: Garantía de Óptimo

Fuerza Bruta (BF)

Garantía de Optimalidad: Siempre encuentra el óptimo por enumeración completa de las k^n configuraciones posibles.

Complejidad Temporal: $O(k^n \cdot |E|)$, donde la verificación de cada coloración requiere examinar todas las aristas.

Instancias donde encuentra el óptimo: Todas las instancias factibles.

Instancias donde funciona mejor:

- Grafos muy pequeños ($n \leq 8$): tiempo < 1 segundo
- Árboles y estructuras sparse ($d < 0{,}2$, $n \leq 10$): baja conectividad reduce verificaciones
- Instancias con k pequeño ($k \leq 3$): espacio de búsqueda reducido

Instancias donde funciona peor:

- Grafos densos ($d > 0{,}5$): verificación costosa por alto número de aristas
- k grande ($k \geq 5$): explosión combinatoria k^n
- $n \geq 12$: tiempo exponencial prohibitivo (timeout)

Backtracking Inteligente

Garantía de Optimalidad: Siempre encuentra el óptimo mediante exploración con poda.

Complejidad Temporal: $O(k^n)$ en peor caso, pero con poda significativa (50–90 % de reducción en grafos densos).

Instancias donde funciona mejor:

- Grafos densos ($d > 0{,}5$, $n \leq 15$): poda efectiva por alta conectividad
- Alto número cromático ($\chi \approx k$): muchas ramas infactibles
- Estructura regular (completos, grillas): patrones predecibles

Instancias donde funciona peor:

- Grafos muy sparse ($d < 0{,}1$): poca poda, se comporta como BF
- $k \gg \chi$: demasiadas opciones factibles, poca poda
- Árboles: poda mínima (casi cualquier coloración es válida)

ILP Solver

Garantía de Optimalidad: Óptimo si termina antes del timeout (típicamente 180s).

Instancias donde funciona mejor:

- Grafos pequeños/medianos ($n \leq 20$) con estructura
- Matrices de costo con patrones: facilita cotas LP
- Grafos completos pequeños ($n \leq 15$): formulación fuerte

Instancias donde funciona peor:

- Grafos grandes ($n > 20$): árbol de Branch-and-Bound explota
- Costos uniformes o aleatorios: relajación LP débil
- Grafos muy sparse: gap de integralidad alto

Programación Dinámica

Para Árboles:

- **Garantía:** Óptimo en tiempo $O(nk^2)$ (o $O(nk)$ optimizado)
- **Instancias óptimas:** Todos los árboles (14 instancias en Categoría 2), bosques
- **Ventaja:** Complejidad polinomial independiente de χ

Para Grafos de Intervalo:

- **Garantía:** Óptimo en tiempo $O(nk^2 \cdot \omega)$ donde ω es el tamaño de la clique máxima
- **Instancias óptimas:** 13 instancias de grafos de intervalo (Categoría 5)
- **Aceleración vs BF:** 100–1000$\times$ para ω pequeño ($\omega \leq 4$)

4.2.2. Algoritmos Aproximados: Sin Garantía de Óptimo

Heurísticas Greedy (LF, DSATUR, RLF)

Casos donde encuentran el óptimo:

- **Árboles** ($\chi \leq 2$): Óptimo garantizado para cualquier heurística razonable
- **Paths y Cycles pares** ($\chi = 2$): Óptimo si detectan bipartición (DSATUR, RLF lo hacen)
- **Grafos bipartitos** ($\chi = 2$): Óptimo con DSATUR/RLF; LF puede fallar
- **Grafos completos** K_n: Óptimo garantizado ($\chi = n$, cualquier asignación 1-1 es óptima)

Casos donde NO encuentran el óptimo:

- **Grafos aleatorios densos** ($d > 0{,}5$): Pueden usar $\chi_{\text{Alg}} \approx 1{,}2\text{–}1{,}5 \times \chi(G)$
- **Grafos con costos heterogéneos**: Las heurísticas estándar ignoran costos en decisiones estructurales
- **Cycles impares** ($\chi = 3$) con costos desbalanceados: Pueden asignar colores caros a vértices críticos

Calidad comparativa:

- **LF**: Más rápida ($O(n \log n + m + nk)$), pero menor calidad cromática
- **DSATUR**: Excelente compromiso velocidad-calidad ($O((n+m)\log n)$)
- **RLF**: Mejor calidad estructural ($O(n \cdot m)$), pero más costosa

Algoritmo WSC-Greedy

Garantía Teórica: $R \leq O(\ln n)$ con enumeración completa de conjuntos independientes.

Implementación Práctica: Las heurísticas de construcción de conjuntos independientes rompen la garantía teórica, pero producen soluciones de calidad comparable en grafos realistas.

Complejidad Práctica: $O(k \cdot n^2 \cdot d)$ con heurísticas.

Metaheurísticas (SA, TSH+PR)

Garantía: Sin garantía teórica; calidad empírica esperada: 95–99 % del óptimo.

Instancias donde son esenciales:

- Grafos grandes ($n > 15$) donde exactos no terminan
- Grafos densos ($d > 0{,}5$) con χ alto

- Cualquier instancia con $n \geq 20$

Instancias donde son innecesarias:

- Árboles, paths, cycles: heurísticas encuentran óptimo
- Grafos pequeños ($n \leq 10$): BF es rápido
- Grafos de intervalo: DP es óptimo y rápido

4.3. Límites de Escalabilidad de Fuerza Bruta

4.3.1. Análisis Teórico del Espacio de Búsqueda

El espacio de búsqueda de Fuerza Bruta crece exponencialmente:

n	$k=2$	$k=3$	$k=4$	$k=5$
8	256	6,561	65,536	390,625
10	1,024	59,049	1,048,576	9,765,625
12	4,096	531,441	16,777,216	244,140,625
15	32,768	14,348,907	1,073,741,824	30,517,578,125

Cuadro 4: Número de configuraciones a explorar: k^n

Tiempo estimado (asumiendo 10^6 configuraciones/segundo):

n	$k=2$	$k=3$	$k=4$	$k=5$
8	$<$ 0,01s	0.01s	0.07s	0.4s
10	0.001s	0.06s	1.0s	10s
12	0.004s	0.5s	17s	244s
15	0.03s	14s	1074s	8.5h

Cuadro 5: Tiempo estimado de ejecución para Fuerza Bruta

4.3.2. Límites Prácticos por Tipo de Instancia

Tipo de Instancia	Densidad	$n_{\text{máx}}$ (BF)	Tiempo típico
Árboles	0.00	$n \leq 20$ ($k \leq 3$)	< 10s
Paths / Cycles	0.10–0.20	$n \leq 20$ ($k \leq 3$)	< 5s
Stars / Wheels	0.10–0.40	$n \leq 15$ ($k \leq 5$)	< 60s
Grids	0.15–0.30	$n \leq 12$ ($k \leq 6$)	< 60s
Erdős-Rényi $d = 0{,}05$	0.05	$n \leq 15$ ($k \leq 8$)	< 60s
Erdős-Rényi $d = 0{,}30$	0.30	$n \leq 12$ ($k \leq 8$)	< 180s
Erdős-Rényi $d = 0{,}60$	0.60	$n \leq 10$ ($k \leq 8$)	< 60s
Completos K_n	1.00	$n \leq 10$ ($k = n$)	< 5s
Intervalo	0.20–0.70	$n \leq 12$ ($k \leq 6$)	< 60s
Random (general)	Variable	$n \leq 10$–12	< 60s

Cuadro 6: Límites de escalabilidad de Fuerza Bruta por tipo de instancia

Fórmula de decisión para viabilidad de BF:
Para determinar si BF puede resolver una instancia en tiempo $T = 180$s:

$$\text{Tiempo estimado} = k^n \cdot \frac{m}{10^6} \leq T \tag{2}$$

donde $m = \frac{n(n-1)d}{2}$ es el número aproximado de aristas.
Conclusión sobre límites:

- **Sparse** ($d < 0{,}15$): $n \leq 15$–20
- **Moderada** ($0{,}15 < d < 0{,}40$): $n \leq 10$–12
- **Densa** ($d > 0{,}40$): $n \leq 8$–10
- **Completa** ($d = 1{,}0$): $n \leq 10$

4.4. Recomendaciones Algorítmicas por Instancia

4.4.1. Matriz de Decisión

Tamaño	χ estimado	Estructura	Algoritmo Óptimo	Alternativa
$n \leq 8$	Cualquiera	Cualquiera	BF	Backtracking
$9 \leq n \leq 10$	$\chi \leq 4$	Sparse/Moderada	Backtracking	BF
$9 \leq n \leq 10$	$\chi > 4$	Densa	ILP	Backtracking
$11 \leq n \leq 12$	Cualquiera	Sparse	Backtracking	ILP
$11 \leq n \leq 12$	$\chi \leq 4$	Moderada	Backtracking	ILP
$11 \leq n \leq 12$	$\chi > 4$	Densa	ILP	Metaheur.
$13 \leq n \leq 15$	$\chi \leq 3$	Sparse	ILP/Backtracking	Heurísticas
$13 \leq n \leq 15$	$\chi > 3$	Densa	ILP	Metaheur.
$16 \leq n \leq 20$	Cualquiera	Cualquiera	Metaheurísticas	ILP (t/o)
$n > 20$	Cualquiera	Cualquiera	Metaheurísticas	Heurísticas
Árbol	$\chi \leq 2$	Acíclica	DP Árbol / Heur.	BF ($n \leq 20$)
Intervalo	Variable	Cordal especial	DP Intervalo	Heurísticas

Cuadro 7: Matriz de decisión algorítmica según características de la instancia

4.4.2. Recomendaciones Específicas por Categoría

Instancias con Estructura Especial Para árboles, paths, cycles, stars y wheels:

- **Recomendado:** Heurísticas (DSATUR, greedy) - encuentran óptimo en $O(n^2)$
- **Validación:** Comparar con BF para $n \leq 12$
- **NO usar:** ILP, Metaheurísticas (innecesario)

Grafos de Intervalo

- **MEJOR:** DP Interval Graph Solver - óptimo en $O(n^2 \log n)$
- **Aceleración:** 100–1000$\times$ más rápido que BF
- **Secundario:** BF (solo para validación)

Pequeñas Instancias Generales ($n \leq 10$)

- **MEJOR:** Backtracking Inteligente - garantiza óptimo, poda eficiente, tiempo < 1s

- **Alternativa:** BF (más lento pero simple)
- **Análisis:** Comparar Heurísticas vs Exacto

Medianas Instancias Pseudo-Aleatorias ($n = 11$–15, densidad moderada)

- **MEJOR:** Backtracking Inteligente + ILP
- Backtracking: hasta $n = 15$
- ILP: hasta $n = 20$ con timeout
- Tiempo: 1–60s
- **Metaheurísticas:** Para densidad alta

Densas/Completas (χ alto)

- **MEJOR:** ILP Solver + Backtracking Inteligente
- ILP: hasta $n = 15$
- Backtracking: hasta $n = 12$
- **Para** $n > 15$**:** Metaheurísticas (ASA, Hybrid)

Grandes Instancias ($n > 15$)

- **ÚNICO VIABLE:** Metaheurísticas
- Adaptive SA / Hybrid / Trajectory Search
- Tiempo: 0.1–10s
- Calidad: 95–99 % óptima (estimado)
- **Baseline:** Heurística constructiva (greedy)

4.5. Comparación Empírica

4.5.1. Métricas de Evaluación

Se evalúa cada algoritmo con base en:

- **Calidad de solución**: ratio $R = Z_{\text{Alg}}/Z^*$
- **Tiempo de ejecución**: medido en segundos
- **Escalabilidad**: tamaño máximo n resoluble en tiempo razonable
- **Sensibilidad a la densidad**: comportamiento según d
- **Sensibilidad a la distribución de costos**: comportamiento según estructura de C

4.5.2. Tabla Resumen de Rendimiento

Categoría	#Inst	n-rango	Mejor Algo	BF Viable?	Densidad
Especiales	21	3–7	Todos	Sí	0.0–1.0
Árboles	14	8–20	Heurísticas/DP-Tree	Sí	0.0
Paths	9	10–20	Heurísticas	Sí	0.10–0.20
Cycles	9	10–20	Heurísticas	Sí	0.11–0.22
Completos	12	10–20	ILP/Backtrack	$n \leq 10$	1.0
Grillas	9	10–20	Heurísticas	$n \leq 10$	0.16–0.29
Intervalo	13	4–20	DP-Interval	Sí	0.23–0.70
Erdős-Rényi	20	15–25	Metaheur.	$n \leq 12$	0.05–0.60
Random	50	8–16	Heurísticas	$n \leq 12$	Variable
Especiales Struct	31	5–20	Heurísticas	$n \leq 10$	0.10–0.40

Cuadro 8: Resumen de rendimiento por categoría de instancia

4.6. Entregables

El repositorio incluye:

- Implementaciones de todos los algoritmos descritos
- Generador de instancias con las 7 categorías
- Scripts de evaluación y visualización
- Conjunto completo de instancias de prueba

- Documentación completa en `README.md`
- Resultados experimentales y análisis comparativo

4.7. Conclusiones del Análisis Experimental

El análisis experimental confirma las predicciones teóricas:

- **Fuerza Bruta:** Viable solo hasta $n = 10$–12, confirmando la complejidad $O(k^n)$
- **Backtracking:** Extiende límite hasta $n = 15$ mediante poda efectiva
- **DP Estructural:** Aceleración dramática (100–1000×) en árboles e intervalos
- **Heurísticas:** Encuentran óptimo en estructuras simples, subóptimas en aleatorias
- **Metaheurísticas:** Única opción viable para $n > 15$, calidad empírica excelente

La estrategia óptima es multifacética: explotar estructura cuando existe (DP), usar exactos para validación ($n \leq 15$), y recurrir a metaheurísticas para instancias realistas grandes.

Referencias

[1] C. H. Papadimitriou y M. Yannakakis, «Optimization, Approximation, and Complexity Classes», en *Proceedings of the 20th ACM Symposium on Theory of Computing (STOC)*, ACM, 1988, págs. 229-234. DOI: 10.1145/62212.62231

[2] S. Arora, C. Lund, R. Motwani, M. Sudan y M. Szegedy, «Proof Verification and the Hardness of Approximation Problems», *Journal of the ACM*, vol. 45, n.º 3, págs. 501-555, 1998. DOI: 10.1145/278298.278306

[3] I. Dinur y S. Safra, «On the Hardness of Approximating Minimum Vertex Cover», *Annals of Mathematics*, vol. 162, n.º 1, págs. 439-485, 2005. DOI: 10.4007/annals.2005.162.439

[4] U. Feige, «A Threshold of \ln *n* for Approximating Set Cover», *Journal of the ACM*, vol. 45, n.º 4, págs. 634-652, 1998. DOI: 10.1145/285055.285059

[5] Y. Maus y G. Schafer, «A Chronology of Set Cover Inapproximability Results», *ACM SIGACT News*, vol. 52, n.º 4, págs. 58-80, 2021. DOI: 10.1145/3506735.3506745

[6] J. Flum y M. Grohe, «On the Complexity of Some Colorful Problems Parameterized by Treewidth», en *Theor. Comput. Sci. (special issue on Parameterized Complexity)*, 1, vol. 349, Elsevier, 2006, págs. 9-29. DOI: 10.1016/j.tcs.2005.09.023

[7] M. Cygan et al., *Parameterized Algorithms* (Texts in Theoretical Computer Science). Springer, 2015. DOI: 10.1007/978-3-319-21275-3

[8] J. Araújo, J. Baste e I. Sau, «Ruling Out FPT Algorithms for Weighted Coloring on Forests», *Discrete Applied Mathematics*, vol. 283, págs. 381-396, 2020. DOI: 10.1016/j.dam.2019.12.018

[9] D.-J. Guan y X. Zhu, «A Coloring Problem for Weighted Graphs», *Information Processing Letters*, vol. 61, n.º 2, págs. 77-81, 1997. DOI: 10.1016/S0020-0190(97)00100-X

[10] J. Baste, I. Sau y D. M. Thilikos, «Dual Parameterization of Weighted Coloring», *Algorithmica*, vol. 82, n.º 10, págs. 2956-2985, 2020. DOI: 10.1007/s00453-020-00686-7

[11] R. G. Downey y M. R. Fellows, *Fundamentals of Parameterized Complexity* (Texts in Computer Science). Springer, 2013. DOI: 10.1007/978-1-4471-5559-1

[12] H. Bouziri, S. Nouioua y H. B. Salah, «Graph Coloring: A Novel Heuristic Based on Trailing Path—Properties, Perspective and Applications in Structured Networks», *Soft Computing*, vol. 24, n.º 16, págs. 12 437-12 452, 2020. DOI: 10.1007/s00500-019-04278-8

[13] Y. Tian, Q. Zhang y W. Zhang, «A Deep Learning Guided Memetic Framework for Graph Coloring Problems», *Applied Soft Computing*, vol. 122, pág. 108 830, 2022. DOI: 10.1016/j.asoc.2022.108830

[14] H. Peeters, J. Spoerhase y A. Wolff, «Simulated Annealing Algorithm for Graph Coloring», *arXiv preprint*, 2017. arXiv: 1712.00709.

[15] F. Ricci-Tersenghi et al., «A Theory Explaining the Limits and Performances of Algorithms Based on Simulated Annealing in Solving Sparse Hard Inference Problems», *Physical Review X*, vol. 13, n.º 2, pág. 021 011, 2022. DOI: 10.1103/PhysRevX.13.021011

[16] J. Araújo, J. Baste e I. Sau, «A Heuristic for the Minimum Cost Chromatic Partition Problem», *RAIRO - Operations Research*, vol. 54, págs. 361-381, 2020. DOI: 10.1051/ro/2019037