UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Trabajo de Investigación - Graph Coloring

Integrantes: Manuel Diaz
Sebastian Escobar

Curso: Optimizacion en Ingenieria

Profesor: Monica Villanueva Ilufi

Ayudante: Cristian Sepulveda

27 de Mayo de 2023

Tabla de contenidos

1. Resumen.

El problema de la coloración de grafos se refiere a asignar colores a cada vértice de un grafo de manera que los vértices adyacentes no compartan el mismo color. Este desafío, que se encuentra entre los problemas *NP-Hard*, es parte integral de la teoría de grafos y la optimización combinatoria, y se han propuesto soluciones a través de métodos exactos, heurísticos y distribuidos. Su utilidad es amplia, abarcando áreas como la asignación de horarios, la asignación de frecuencias en redes de comunicación y logística. En este informe, se explora un método exacto mediante programación lineal entera, culminando con un análisis de los resultados y una síntesis de los hallazgos.

2. Palabras clave.

Coloración de Grafos, Graph Coloring, Optimización, Algoritmos Exactos, Conjunto potencia, Julia, Complejidad Algoritmica, NP-Hard

3. Introducción.

El problema de Coloración de Grafos o *Graph Coloring*, forma parte de la familia de problemas *NP-Hard* y existen distintas soluciones a este, ya sean exactas o aproximadas. Este se considera un problema de optimización, debido a que la solución obtenida debe garantizar ser eficiente y óptima. El objetivo principal es encontrar una asignación de colores que cumpla con la restricción de que los vértices adyacentes no compartan el mismo color, utilizando el menor número posible de colores. (Thulasiraman et al., 2016) (Cap. 19.1).

Dada su clasificación como un problema *NP-Hard*, no se puede garantizar la existencia de un método que pueda resolver este problema de coloración, en tiempo polinómico para todos los casos. Sin embargo, se han propuesto varios enfoques de solución, que incluyen algoritmos exactos, heurísticos, distribuidos, y métodos de programación lineal entera. (Gross and Yellen, 2004)

La Coloración de Grafos tiene aplicaciones en numerosas áreas como la asignación de horarios en instituciones educativas, la asignación de frecuencias en redes de comunicación móvil, su utilidad en cartografía y logística, entre otras áreas. (Roberts, 1976)

En el presente trabajo de investigación, se examinará el marco teórico que sustenta el problema de coloración de grafos, explorando tanto estrategias exactas como heurísticas existentes para su resolución. Además, se profundizará en el método de programación lineal entera, el cual constituye un método exacto. Este método se implementará como un problema con restricciones utilizando el lenguaje de programación Julia, con el propósito de analizar su rendimiento y describir los resultados obtenidos.

4. Desarrollo.

4.1. Estado del arte.

Para describir el problema de coloración de vértices en grafos debemos tener en cuenta lo siguiente. Consideremos el grafo $G = (V, E)$ no orientado, con V como un conjunto de n vértices tal que $|V| = n$ y E un conjunto de m aristas, tal que $|E| = m$. De igual forma.

- $V \neq \varnothing$
- $E \in \{x \in P(V) : |x| = 2\}$

Donde V debe contener vértices para ser distinto de vacío y cada arista $e \in E$ debe cumplir con pertenecer al conjunto potencia de V $(P(V))$, con las aristas que cumplan ser de cardinalidad 2 (Wilson, 2010) (Cap. 2).

Entonces dado un entero k, el cual se identifica como el límite cromático de colores a utilizar, y una función de coloración c (la cual nos entregara un valor entero), denotamos por cada vértice v de nuestro grafo la evaluación de c tal que $c(v) \in \{1, ...k\}$ (Galinier et al., 2013) (Sec. 1). Bajo este esquema los colores son representados mediante un valor entero, lo cual permite facilitar la identificación de la cantidad de colores únicos a utilizar.

Si sobre G utilizamos una coloración de a lo más k colores, entonces se denomina a G *k-coloreable*. En tal caso, el menor número de colores necesarios para colorear a G se llama número cromático y se denota por $X(G)$. Si a G es posible asignarle una *k-coloración* es *k-coloreable* y es *k-cromático* sí $X(G) = k$ (Galinier et al., 2013) (Sec. 1).

De esta forma, como G es *k-coloreable* podemos hacer de este *k-partito*, donde cada partición agrupa los sets de vértices, tal que $\{V_1, V_2, ..., V_k\}$ es la coloración del grafo G y es por eso que $X(G) \leq k$. Adicionalmente, $X(G) = k$ si y solo si G es *k-partito* y no es posible encontrar un $l < k$ que disminuya la cantidad de particiones k

4.1.1. Cota superior sobre $X(G)$.

Considerando a G con $|V| = n$, es posible acotar $X(G)$ mediante $1 \leq X(G) \leq n$, ya que el máximo valor cromático a utilizar se encuentra restringido entre 1 y la cantidad de vértices sobre G. De manera similar, sí consideramos $\Delta(G)$ como el grado máximo de G,

entonces $X(G) \leq \Delta(G) + 1$, esto debido a que en un grafo completo (peor escenario posible) se obtiene $\Delta(G) = n - 1$, por ende obtenemos $X(G) = n$ en peor caso (Bondy, 1969).

4.1.2. Cota inferior sobre $X(G)$.

Sea H un subgrafo de G, entonces $X(H) \leq X(G)$ y sea un clique (subgrafo completo maximal) de G, el máximo número de vértices de un clique se denota $w(g)$, entonces $X(G) \geq w(g)$, para todo grafo G (Bondy, 1969).

4.2. Métodos de solución

4.2.1. Algoritmos exactos.

Los algoritmos exactos en la coloración de grafos son métodos de resolución que buscan encontrar una solución óptima al problema. Estos algoritmos exploran exhaustivamente todas las posibles combinaciones de colores para los vértices de un grafo, determinando la asignación que cumple con las restricciones de no permitir vértices adyacentes con el mismo color y utilizando el menor número de colores posible. A continuación, se enumeran algunos de los algoritmos exactos más conocidos para la coloración de grafos.

1. **Algoritmo de Fuerza de Bruta.**

   En el algoritmo de fuerza bruta, en cada paso del algoritmo, se evalúa una combinación de colores para asignar a los vértices del grafo. Como hay K colores posibles para cada vértice, el número total de combinaciones posibles es K^n. Esto se debe a que para el primer vértice, tenemos k opciones de color; para el segundo vértice, también tenemos k opciones, y así sucesivamente hasta el último vértice. (Cormen et al., 2009)

   Por lo tanto, el algoritmo de fuerza bruta debe evaluar todas estas K^n combinaciones para encontrar la asignación óptima de colores. En el peor caso, cuando el número de colores K es relativamente grande y el número de vértices n también es grande, la complejidad del algoritmo se vuelve exponencial y es computacionalmente costosa.

2. **Algoritmo de Ramificación y Poda.**

   El algoritmo de ramificación y poda es otro enfoque exacto utilizado para resolver el problema de la coloración de grafos. Divide el problema en subproblemas más pequeños y los resuelve recursivamente. Se generan ramificaciones que representan posibles asignaciones de colores a vértices adicionales, y se realiza una poda para descartar soluciones que no cumplen con ciertos criterios de optimización. (Cormen et al., 2009)

3. **Algoritmo de Backtracking.**

   El algoritmo de Backtracking es otro algoritmo exacto, el procedimiento comienza con una asignación parcial, donde algunos vértices ya tienen un color asignado y otros están sin colorear. En cada paso, el algoritmo realiza una selección de color para un vértice no coloreado y verifica si cumple con las restricciones de adyacencia, es decir, si no tiene el mismo color que sus vértices adyacentes. Si la asignación cumple con las restricciones, el algoritmo avanza al siguiente vértice no coloreado y repite el proceso. (Cormen et al., 2009)

   Si la asignación no cumple con las restricciones, el algoritmo realiza un retroceso y prueba con otro color para el vértice anterior. Esto implica deshacer la asignación realizada y explorar otras opciones posibles. El proceso de retroceso continúa hasta que se encuentre una asignación completa y válida que cumpla con todas las restricciones de adyacencia, o hasta que se agoten todas las opciones posibles. (Cormen et al., 2009)

4. **Algoritmo de Programación Lineal Entera.**

   El Algoritmo de Programación Lineal Entera es otro enfoque utilizado para resolver el problema de la coloración de grafos. Este método emplea técnicas de programación lineal entera para modelar el problema como un problema de optimización lineal.

   Se definen variables binarias que representan la asignación de colores a los vértices del grafo. Cada variable binaria indica si un vértice determinado está asignado a un color específico. Se establecen restricciones lineales que aseguran que los vértices adyacentes no tengan el mismo color. Además, se establecen objetivos lineales para minimizar el número total de colores utilizados. (Wolsey, 1998)

4.2.2. Resumen Algoritmos Exactos.

En la siguiente tabla se muestra un resumen de los algoritmos exactos mencionados anteriormente.

Algoritmo	**Complejidad de Tiempo (Peor Caso)**	**¿Determinístico?**	**¿Solución Óptima?**	**Espacio de Búsqueda**
Fuerza Bruta	$O(K^n)$	Sí	Sí	Completo
Ramificación y Poda	Variable, hasta $O(K^n)$	Sí	Sí	Parcial/Completo
Backtracking	Variable, hasta $O(K^n)$	Sí	Sí	Parcial/Completo
Programación Lineal Entera	Dependiente de la implementación	Sí	Sí	Completo

Cuadro 1: Resumen de los algoritmos exactos de coloración de grafos.

La complejidad de tiempo en el peor caso se proporciona para evaluar el rendimiento computacional de cada algoritmo. Además, se indica si los algoritmos son determinísticos, lo que significa que producen la misma solución para una instancia dada, y si proporcionan soluciones óptimas, que son las soluciones de coloración con el menor número posible de colores. Finalmente se menciona el espacio de búsqueda utilizado por cada algoritmo. Algunos algoritmos exploran todo el espacio de búsqueda, mientras que otros algoritmos utilizan estrategias más limitadas y exploran solo una parte del espacio de búsqueda.

Los algoritmos exactos en la coloración de grafos ofrecen garantías de encontrar la solución óptima, pero pueden ser computacionalmente costosos en casos de grafos grandes. Por esta razón, en muchos casos se utilizan enfoques heurísticos y aproximados, que ofrecen soluciones rápidas pero no necesariamente óptimas, para abordar de manera eficiente problemas de coloración de grafos en la práctica.

4.2.3. Algoritmos heuristicos.

Existen varios algoritmos heurísticos utilizados en coloración de grafos. A continuación, se describen algunos de ellos:

Nombre	Descripción
Algoritmo DSatur	Este algoritmo utiliza una heurística basada en el grado de saturación de los vértices. Comienza seleccionando el vértice con el mayor grado y le asigna un color. Luego, selecciona el siguiente vértice de mayor grado de saturación, que representa el número de colores diferentes utilizados por sus vecinos, y se le asigna el color disponible más bajo. Se repite el proceso hasta colorear todos. (Brélaz, 1979)
Algoritmo de Ordenación de Grados	Este algoritmo se basa en ordenar los vértices del grafo según su grado en orden descendente. Luego, los vértices se colorean uno por uno en el orden dado, asignándoles el color más bajo que no esté siendo utilizado por sus vecinos ya coloreados. (Wilson, 2010)
Algoritmo de Búsqueda Local	Este enfoque consiste en partir de una asignación inicial de colores y realizar cambios locales para mejorar la calidad de la solución. Se exploran diferentes intercambios de colores entre los vértices adyacentes, buscando reducir el número de colores utilizados o mejorar alguna medida de calidad definida. (Aarts and Lenstra, 2003)
Algoritmo de Coloración Greedy	Este algoritmo asigna colores a los vértices en un orden determinado, sin considerar los colores de los vértices adyacentes. Se selecciona un vértice y se le asigna el color disponible más bajo. Luego, se procede con los vértices restantes, asignándoles el color más bajo posible que no esté siendo utilizado por sus vecinos ya coloreados. Este proceso continúa hasta que todos los vértices estén coloreados. (Wilson, 2010)

Cuadro 2: Algunos algoritmos heuristicos

4.2.4. Enfoques distribuidos y paralelizados.

Los enfoques distribuidos y paralelizados para la coloración de grafos aprovechan la capacidad de procesamiento distribuido y paralelo para acelerar el proceso de coloración en grafos de gran tamaño. En los algoritmos distribuidos, el grafo se divide en partes más pequeñas asignadas a diferentes nodos o procesadores en un sistema distribuido. Cada nodo realiza la coloración de su parte del grafo de manera independiente, coordinando la asignación de colores para evitar conflictos. Por otro lado, los algoritmos paralelizados ejecutan tareas de coloración simultáneamente en diferentes núcleos de procesamiento o sistemas multiprocesador. Cada tarea se realiza de forma independiente, utilizando técnicas de sincronización y comunicación entre los núcleos para garantizar la consistencia en la asignación de colores. Estos enfoques permiten una coloración eficiente de grafos grandes al reducir el tiempo de ejecución y abordar problemas de escalabilidad. (Perron and Malick, 2009)

Estos enfoques distribuidos y paralelizados pueden combinar diferentes algoritmos heurísticos o exactos para lograr una coloración eficiente de grafos. Al distribuir o paralelizar el proceso de coloración, se logra una reducción significativa del tiempo de ejecución y se abordan problemas de escalabilidad en grafos grandes. (Perron and Malick, 2009)

4.2.5. Problemas similares.

Similar a la coloración de vértices en grafos, existen problemas similares que nacen de las definiciones de coloración de vértices como casos particulares o más bien abarcan un rango más general sobre las definiciones ya establecidas.

1. **Polinomio cromático.**

   Se conoce que un grafo G es *k-coloreable* si existe una coloración para G, con un set de k colores y que el valor mínimo de k corresponde al número cromático de G y es denotado por $X(G)$. Con esto último en cuenta, consideremos ahora un λ entero positivo, tal que sobre G se pueden aplicar λ-*coloraciones* y $P(G, \lambda)$ se denominara el polinomio cromático de G, es decir, el polinomio cromático son todas las coloraciones factibles que puedo aplicar sobre G (Al-Gounmeein, 2012).

2. **Coloración de aristas.**

   Similar a la coloración de vértices, el número cromático de aristas sobre G, es el menor número de colores requeridos para colorear $e \in E$, tal que no existan dos aristas adyacentes de igual coloración. El número cromático de aristas sobre G se denota mediante $X'(G)$ (Jaradat, 2005).

3. **Coloración total.**

   La coloración total de grafos se denota mediante $X''(G)$, esta consiste en componer las reglas de coloración de aristas junto con la coloración de vértices, respetando que, dos vértices adyacentes no pueden ser coloreados de igual forma, dos aristas adyacentes no pueden ser coloreadas de igual forma y las aristas no pueden estar coloreadas de igual manera que sus vértices terminales (Muthuramakrishnan and Govindasamy, 2021).

4.2.6. Aplicaciones.

A continuación identificamos algunos contextos en los cuales es posible utilizar coloración de grafos, esto con tal de encontrar una solución factible al problema planteado.

1. **Radiofrecuencias.**

   Dentro de un territorio específico, se encuentran distribuidas N estaciones de radio. Cada una está definida por su posición en coordenadas (x, y) y tiene un radio de alcance R. El objetivo es asignar frecuencias de manera que ninguna estación con rangos de cobertura superpuestos comparta la misma frecuencia, procurando minimizar la cantidad total de frecuencias utilizadas. (Dobre et al., 2015)

   El problema se representa mediante un grafo G, donde cada vértice del grafo representa un círculo centrado en las coordenadas (x, y) con un radio R. Dos vértices se consideran adyacentes si las áreas de los círculos correspondientes se superponen. Las frecuencias se asignan utilizando colores distintos. En la siguiente figura se aprecia este caso.

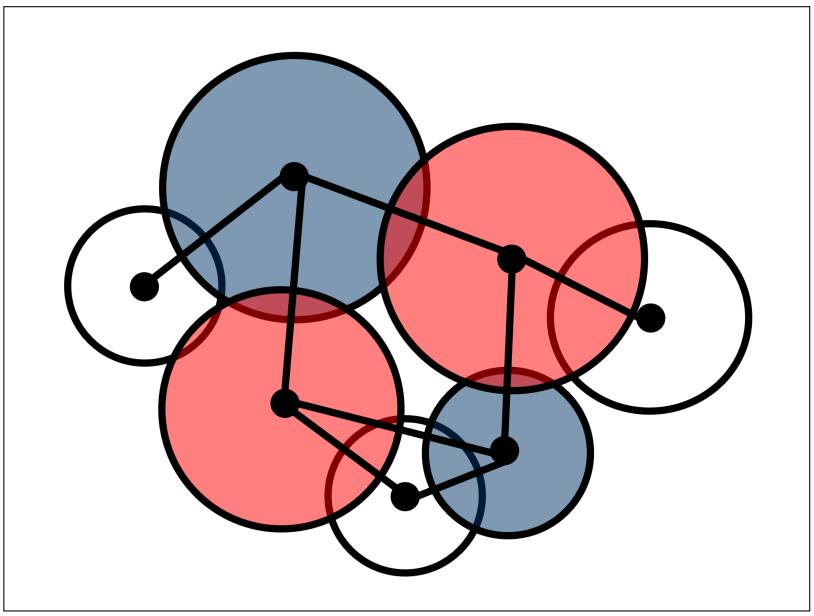

Figura 1: Frecuencias inalámbricas: minimizando interferencias con coloración de grafos.

El objetivo es encontrar una coloración óptima para el grafo G, es decir, una asignación de colores a los vértices de manera que ningún par de vértices adyacentes comparta el mismo color. La cantidad mínima de frecuencias utilizadas se conoce como el número cromático de G. Al minimizar el número de frecuencias necesarias, podemos maximizar la capacidad de transmisión sin interferencias entre las estaciones.

2. **Cartografía y Logística.**

   La coloración de grafos en mapas es una técnica utilizada en cartografía para asignar colores a regiones adyacentes de forma que las fronteras entre ellas sean claramente distinguibles. Esto permite una representación visual ordenada y comprensible de las distintas áreas geográficas. En la Figura 2 se presenta un ejemplo para las regiones de Estados Unidos.

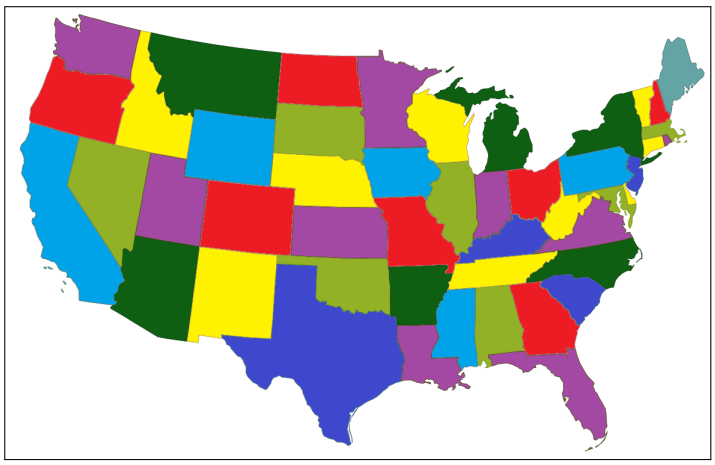

Figura 2: Mapa de Estados Unidos: Distintos colores para regiones adyacentes.

Además de su aplicación en la cartografía, la coloración de grafos en mapas tiene diversas utilidades prácticas, como la planificación de rutas eficientes y la asignación de recursos en logística. Al asignar colores diferentes a las regiones limítrofes, se facilita la identificación de caminos óptimos y la distribución adecuada de recursos, mejorando la eficiencia en diversas aplicaciones geográficas. (Wilson, 2002)

3. **Calendarización de exámenes.**

   Una universidad debe generar el calendario anual de exámenes para todos sus cursos ofertados. Es necesario que la examinación de dos diferentes cursos no tenga estudiantes en común. El problema de encontrar una calendarización para los exámenes con la menor cantidad de bloques horarios a utilizar, puede ser reducido a un problema de coloración en grafos. Definimos el grafo G, cuyo conjunto de vértices es el conjunto de todos los cursos y dos cursos están unidos por un borde si tienen estudiantes en común. El número mínimo de bloques horarios exigido para el calendario del examen es el número cromático de G (Thulasiraman et al., 2016).

4. **Almacenamiento de químicos.**

   Un conjunto de n químicos industriales $c_1, c_2, ..., c_n$ deben ser almacenados en un depósito. Ciertos pares de químicos pueden provocar explosiones si entran en contacto y es por eso que deben almacenarse en distintos compartimientos del depósito. Entonces el

número de compartimientos dentro del depósito debe ser al menos equivalente al número cromático de G, donde $V(G) = \{c_1, c_2, ..., c_n\}$ y las aristas que unen $c_i c_j \in E(G)$ deben cumplir que si c_i y c_j entran en contacto provocan una explosión (Thulasiraman et al., 2016).

4.3. Implementación programación lineal entera.

La programación lineal entera es una técnica comúnmente empleada para el desarrollo de problemas de optimización, en esta situación se utilizan variables de valores enteros. Es posible emplear este enfoque para el problema de *graph coloring*, pero puede ser ineficiente al momento de escalar la cantidad de vértices y aristas sobre en el problema de entrada.

El modelo de programación lineal es el siguiente, sea $G = (V, E)$ un grafo con n vértices y m aristas, se denominan las variables, x_{ij} donde $i \in (1, ..., n)$ y $j \in (1, ..., m)$, donde es posible representar a x de la siguiente manera (Seijas, 2017).

$$x_{ij} = \begin{cases} 1, & \text{Si el vértice } v_i \text{ es coloreado mediante } j \\ 0, & \text{Si no se le ha asignado color alguno} \end{cases} \tag{1}$$

Como fue enunciado previamente, una cota superior para la coloración en grafos, se encuentra dada por el grado máximo de g con $\Delta(G)$, tal que es posible acotar el valor de m a $\Delta(G)$.

Como función objetivo para este modelo, es fácil identificar que la asignación de colores a utilizar debe lo menor posible, por ende la función objetivo la identificamos mediante.

$$min \sum_{i=1}^{n} \sum_{j=1}^{\Delta(G)} x_{ij}$$

Luego identificamos las siguientes restricciones. La suma de los x_{ij}, con $j \in (1, ..., \Delta(G))$ debe ser 1, ya que un vértice no puede tener asignado más de un color en simultáneo.

$$\sum_{i=1}^{\Delta(G)} x_{ij} = 1, \forall v_i \in V$$

De igual forma, para cada i se deben restringir los vértices adyacentes, tal que no exista otro color j igual sobre el vecindario dé v_i.

$$x_{ij} + x_{kj} \leq 1; \forall (v_i, v_k) \in E, \forall j \in (1, ..., \Delta(G))$$

4.3.1. Ambiente de pruebas e implementación

La implementación se diseñó en Julia mediante Jupyter Notebook, esto principalmente dado a las facilidades que entrega este lenguaje respecto al ambiente de ciencias de datos. A nivel de infraestructura se cuenta con un equipo con las siguientes especificaciones.

- Procesador: Intel i7 10510U
- RAM: 16GB
- Sistema Operativo: Windows 11 x64

Para la toma muestras se emplearon los conjuntos de instancias dispuestos en el sitio web https://mat.tepper.cmu.edu/COLOR/instances.html, adjuntos al documento ".tar". Mientras que el procedimiento fue el siguiente.

1. Declaración del modelo

En primera instancia se declara la función de evaluación del modelo a implementar, esta recibe como parámetro un grafo como matriz de adyacencia, sobre este se determinan el número de vértices y el grado máximo del grafo, se enumeran las variables de decisión x_{ij}, se enuncian las restricciones junto con la función objetivo y luego se ejecuta el modelo.

2. Pre Procesamiento de las instancias

El siguiente paso consistió en realizar una lectura de cada fichero de las instancias propuestas con formato ".col", realizando un análisis previo del formato se tomó en cuenta que al inicio de cada línea se declara la función de esa línea, con "`c`" especificando que el contenido de esa línea es un comentario, "`p`" declarando la cantidad de vértices y aristas contenidas en el grafo y "`e`" enunciando la conexión entre dos vértices. Con lo anterior en cuenta se generaron 57 instancias sobre un tipo de dato previamente definido "GraphMinified" (1 por

cada fichero). Al finalizar el procedimiento, cada instancia de "GraphMinified" se encuentra dada por el *nombre*, *número de vértices* y *lista de aristas*.

3. Ejecución y toma de mediciones

Para la toma de muestras se itera por la lista de instancias de "GraphMinified", ejecutando por cada una la función de coloración definida en el paso **1**, recibiendo como salida la cantidad de colores a utilizar, en simultáneo se realiza la medición del tiempo requerido para hallar la solución junto con el espacio en memoria. Los datos posteriormente fueron volcados a un documento ".csv" para su posterior análisis.

4. Representación de resultados

Una vez con los resultados obtenidos sobre un fichero ".csv", se genera un dataframe con los nombres, cantidad de vértices, cantidad de aristas, tiempo necesario para encontrar solución, número de colores requeridos y tamaño en memoria. Estos datos son posteriormente ordenados por cantidad de vértices para la realización de un análisis comparativo entre los distintos escenarios.

4.3.2. Problemáticas presentadas

El principal problema presentado en la toma de muestras fue ineficiencia del modelo implementado, esto último no se encuentra ligado a la implementación o infraestructura, sino más bien debido a que la programación lineal entera escala fácilmente según la cantidad de vértices y aristas de entrada, es por ello que mientras mayor sea la cantidad de vértices y aristas de entradas, mayor será la cantidad de variables x_{ij} que son necesarias evaluar por cada instancia. Por este motivo se evaluaron 27 de las 57 instancias propuestas.

Para la obtención de datos fue necesario un preprocesamiento sobre los ficheros, dado el desconocimiento de la lectura de ficheros, fue necesaria una pequeña investigación para conocer como identificar la entrada y posteriormente ser esta traducida a los requerimientos establecidos. Sin embargo, se logró subsanar prontamente gracias al uso de la librería *Glob* y a la función "parse" que permitió reconocer fácilmente los patrones sobre cada línea de los ficheros.

5. Resultados y Análisis.

Los resultados obtenidos fueron volcados sobre un documento ".csv", posteriormente se ingresaron a un dataframe mediante `pandas` para luego generar las gráficas con `matplotlib`.

Cuadro 3: Resultados por instancia.

	nombre	**vértices**	**aristas**	**tiempo**	**colores**	**memoria**	**grado**
	nombre	vertices	aristas	tiempo	colores	memoria	grado
1	myciel3.col	11	20	3.2583082	4.0	1536.0	5
2	myciel4.col	23	71	0.066825	5.0	5376.0	11
3	queen5_5.col	25	320	0.301187	7.0	6240.0	16
4	queen6_6.col	36	580	1.5603885	11.0	12136.0	19
5	myciel5.col	47	236	1.7311918	6.0	19968.0	23
6	queen7_7.col	49	952	8.5063887	10.0	21600.0	24
7	queen8_8.col	64	1456	34.3926377	13.0	35880.0	27
8	huck.col	74	602	19.8567114	11.0	47400.0	53
9	jean.col	80	508	5.333718	10.0	55080.0	36
10	queen9_9.col	81	2112	109.1973153	16.0	56416.0	32
11	david.col	87	812	73.6978752	12.0	64768.0	82
12	myciel6.col	95	755	51.975641	7.0	76800.0	47
13	queen8_12.col	96	2736	215.3403436	15.0	78376.0	32
14	queen10_10.col	100	2940	278.5657014	16.0	84840.0	35
15	games120.col	120	1276	27.5283257	9.0	121000.0	13
16	queen11_11.col	121	3960	588.9100248	17.0	122976.0	40
17	miles1000.col	128	6432	2137.2341176	44.0	137256.0	86
18	miles1500.col	128	10396	5614.800515	76.0	137256.0	106
19	miles250.col	128	774	11.7235283	9.0	137256.0	16
20	miles500.col	128	2340	347.4175958	22.0	137256.0	38

Continuación en siguiente página

	nombre	vértices	aristas	tiempo	colores	memoria	grado
21	miles750.col	128	4226	1279.2444156	34.0	137256.0	64
22	anna.col	138	986	116.2769786	12.0	159016.0	71
23	queen12_12.col	144	5192	1322.5227147	20.0	172840.0	43
24	queen13_13.col	169	6656	2268.4142042	21.0	236640.0	48
25	mulsol.i.3.col	184	3916	7261.0171194	157.0	279720.0	157

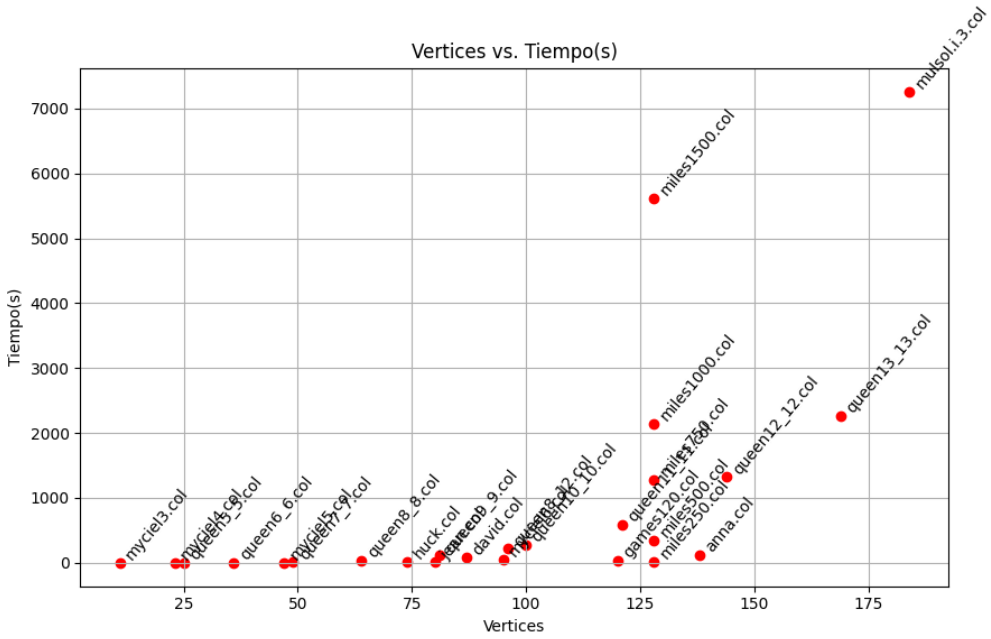

Figura 3: Tiempo requerido según cantidad de vértices.

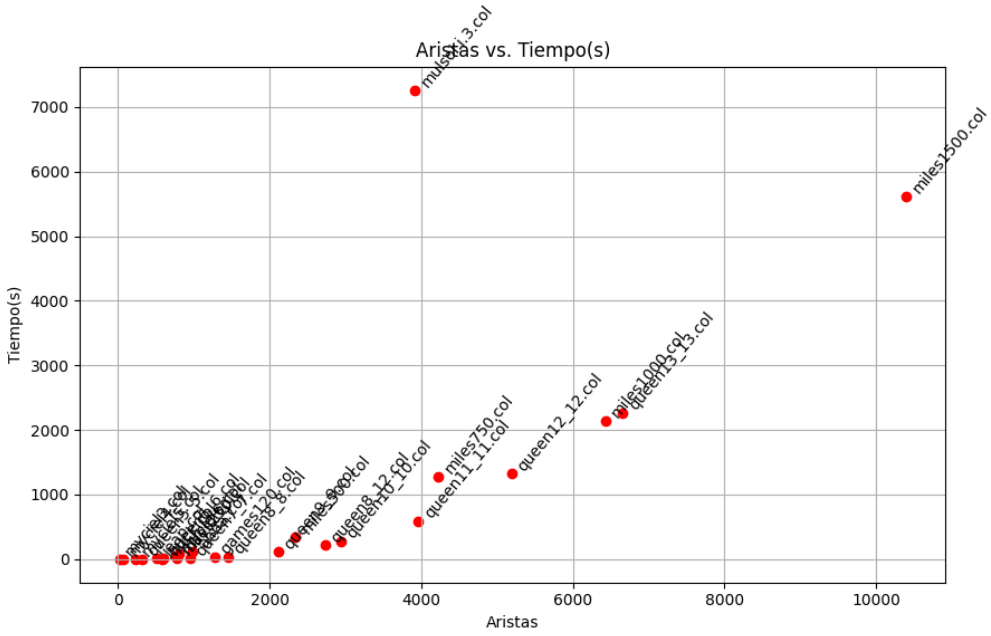

Figura 4: Tiempo requerido según cantidad de aristas.

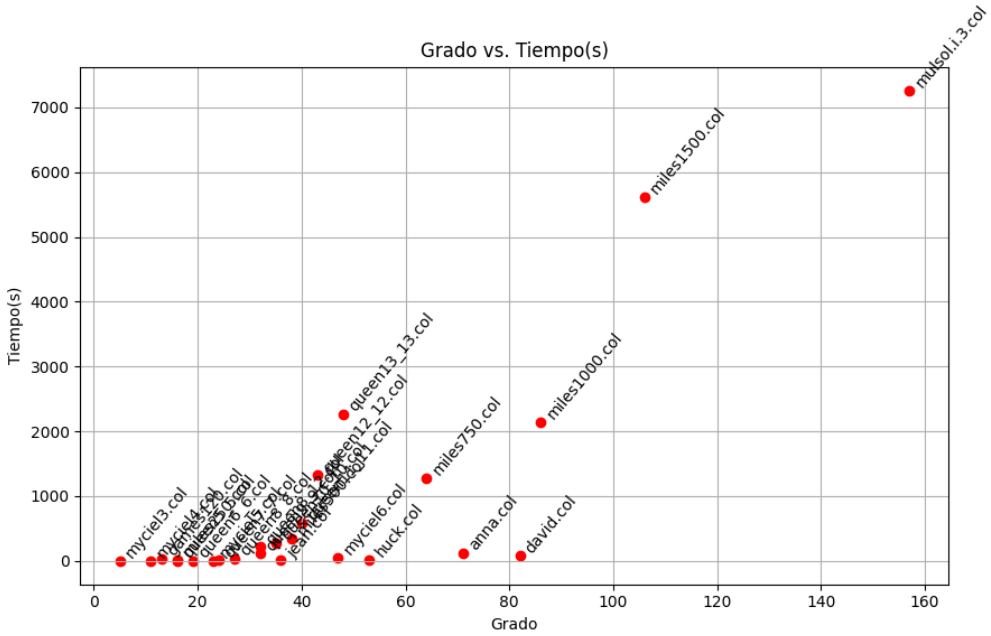

Figura 5: Tiempo requerido según grado maximal de la instancia.

5.1. Discusión.

Analizando las gráficas obtenidas, es posible evidenciar cierto comportamiento de crecimiento lineal, esto respecto al tiempo según la cantidad de aristas en la Figura 4,

exceptuando por las instancias *mulsol.i.3* y *miles1500*, las cuales escapan del crecimiento próximo esperado. Contrastando con la Figura 4, es posible ver que de igual manera ambas instancias incrementan de manera muy elevada el tiempo requerido.

La razón del escenario evidenciado previamente, se explica mediante la Figura 5, donde es posible apreciar que el grado maximal de las instancias *mulsol.i.3* y *miles1500*, es notablemente superior a las demás instancias y considerando que la cantidad de variables x_{ij}, se encuentra dada por el producto entre el número de vértices de entrada multiplicado por el grado maximal del grafo, entonces para *miles1500.col* la cantidad de variables a evaluar es de 13568 y para *mulsol.i.3.col* son 28888 variables, razón por la cual el tiempo requerido es más elevado.

Numerosos estudios y artículos científicos han corroborado la idea de que, en términos generales, a medida que aumenta el tamaño del grafo, el tiempo requerido para resolver problemas como el coloreo de grafos también aumenta. Este fenómeno se debe a la naturaleza combinatoria del problema y a la complejidad inherente de los algoritmos de optimización, como la Programación Lineal Entera, que deben explorar un número creciente de combinaciones a medida que se agregan más vértices al grafo. (Schwartz, 1976)

6. Conclusiones.

El Coloreo de Grafos, también conocido como *Graph Coloring*, asigna colores a los vértices de un grafo para evitar que vértices adyacentes tengan el mismo color. Sus aplicaciones abarcan diversas áreas como cartografía, logística, redes inalámbricas, optimización de horarios y asignación de recursos.

Los algoritmos exactos utilizados en el coloreo de grafos, como fuerza bruta, backtracking, y programación lineal entera, son computacionalmente costosos. Estos algoritmos exploran exhaustivamente todas las combinaciones posibles de colores para los vértices del grafo con el objetivo de encontrar una solución válida. Sin embargo, debido a la complejidad combinatoria del problema, el tiempo de ejecución aumenta considerablemente a medida que el tamaño del grafo, el número de aristas y el grado maximal crecen.

Para el modelo de programación lineal, tal como fue detallado en el análisis, el método se ve severamente perjudicado en función de la cantidad de vértices, junto al grado maximal, lo cual se explica a raíz la cantidad de variables que acaba por evaluar el modelo.

Los algoritmos heurísticos, como Color Greedy, Búsqueda Local y DSatur, son métodos alternativos para abordar el problema del coloreo de grafos. A diferencia de los algoritmos exactos, que buscan la solución óptima, los algoritmos heurísticos ofrecen soluciones aproximadas que pueden no ser óptimas, pero se ejecutan más rápidamente.

Considerando el análisis del Algoritmo de Programación Lineal Entera, A medida que el grafo crece, el algoritmo de Programación Lineal Entera experimenta un aumento significativo en el tiempo de ejecución debido a la complejidad combinatoria del problema del coloreo de grafos. El número de combinaciones posibles crece exponencialmente con el número de vértices, lo que hace que el algoritmo deba explorar exhaustivamente todas las asignaciones de colores. Esto resulta en una mayor complejidad computacional.

El enfoque distribuido y paralelo es una estrategia efectiva para reducir los tiempos de solución en el problema del coloreo de grafos. Al distribuir el proceso de coloreo en múltiples nodos o procesadores, y ejecutarlo en paralelo, se logra una mayor eficiencia computacional y una búsqueda más rápida de soluciones.

Bibliografía

Aarts, E. and Lenstra, J. K. (2003). *Local Search in Combinatorial Optimization*. Princeton University Press.

Al-Gounmeein, R. (2012). On the chromatic polynomial of a cycle graph. *IJAM. International Journal of Applied Mathematics*, 25.

Bondy, J. (1969). Bounds for the chromatic number of a graph. *Journal of Combinatorial Theory*, 7(1):96–98.

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

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms*. MIT Press.

Dobre, O. A., Bajic, D., Zeadally, S., and Kolici, V. (2015). A survey on wireless communication using cognitive radio: How far are we? *Computer Communications*, 67:1–26.

Galinier, P., Hamiez, J.-P., Hao, J.-K., and Porumbel, D. (2013). *Recent Advances in Graph Vertex Coloring*, volume 38, pages 505–528.

Gross, J. L. and Yellen, J. (2004). *Handbook of Graph Theory*. CRC Press.

Jaradat, M. (2005). On the edge coloring of graph products. *International Journal of Mathematics and Mathematical Sciences*, 2005.

Muthuramakrishnan, D. and Govindasamy, J. (2021). Total coloring of certain graphs. *Advances and Applications in Discrete Mathematics*, 27:31–38.

Perron, S. and Malick, J. (2009). Graph coloring for distributed data management on mobile devices. In *International Conference on Algorithmic Applications in Management*, pages 1–12. Springer.

Roberts, F. S. (1976). *Applied Combinatorics*. Prentice-Hall Inc.

Schwartz, J. (1976). On the complexity of integer programming. *Journal of the ACM (JACM)*, 23(3):456–460.

Seijas, S. P. (2017). El problema de coloración de grafos. *Trabajo fin de Máster*, 27.

Thulasiraman, K., Arumugam, S., Brandstädt, A., and Nishizeki, T. (2016). *Handbook of graph theory, combinatorial optimization, and algorithms*.

Wilson, R. (2002). *Four Colors Suffice: How the Map Problem Was Solved*. Princeton University Press.

Wilson, R. J. (2010). *Introduction to Graph Theory*. Prentice Hall/Pearson, New York.

Wolsey, L. A. (1998). *Integer Programming*. Wiley-Interscience.