UNIVERSIDAD AUTÓNOMA METROPOLITANA – IZTAPALAPA
DIVISIÓN DE CIENCIAS BÁSICAS E INGENIERÍA

SOLUCIÓN A PROBLEMAS DE COLORACIÓN CLÁSICOS UTILIZANDO COLORACIÓN DE GRÁFICAS SUAVES

Tesis que presenta:
Daniel Edahi Urueta Hinojosa
Para obtener el grado de
Maestro en Ciencias y Tecnologías de la Información

Asesores: Dr. Pedro Lara Velázquez
Dr. Miguel Ángel Gutiérrez Andrade

Jurado calificador:

Presidente: Dr. Sergio Gerardo de los Cobos Silva

Secretario: Dr. Pedro Lara Velázquez

Vocal: Dra. Hérica Sánchez Larios

México, Ciudad de México, noviembre de 2017

Casa abierta al tiempo

PC y TI

UNIVERSIDAD AUTÓNOMA METROPOLITANA – IZTAPALAPA
DIVISIÓN DE CIENCIAS BÁSICAS E INGENIERÍA

SOLUCIÓN A PROBLEMAS DE COLORACIÓN CLÁSICOS UTILIZANDO COLORACIÓN DE GRÁFICAS SUAVES

Tesis que presenta:

Daniel Edahi Urueta Hinojosa

Para obtener el grado de

Maestro en Ciencias y Tecnologías de la Información

Asesores: Dr. Pedro Lara Velázquez

Dr. Miguel Ángel Gutiérrez Andrade

Jurado calificador:

Presidente: Dr. Sergio Gerardo de los Cobos Silva

Secretario: Dr. Pedro Lara Velázquez

Vocal: Dra. Hérica Sánchez Larios

Email:
crownirv@hotmail.com
crownirv@gmail.com

México, Ciudad de México, octubre de 2017

Resumen

Existe una amplia variedad de problemas de coloración, los cuales modelan problemas del mundo real tales como la asignación de radiofrecuencias, asignación de memoria en clústeres y de registro en microprocesadores, coloración de mapas, calendarización de horarios, reconocimiento de patrones, etc. Entre los problemas de coloración tenemos el problema de coloración de gráficas suaves, en el cual, dado un grafo completo con ponderaciones sobre sus aristas, se busca reducir la suma de las penalizaciones entre dos vértices pintados con el mismo color. Asimismo, existe un grupo denominado problemas clásicos de coloración, que incluye a los problemas de: coloración mínima, débil, equitativa y robusta. Adicionalmente, se sabe que para los problemas anteriormente descritos es necesario emplear técnicas heurísticas o metaheurísticas en instancias grandes, debido a que pertenecen a la clase NP-Dura.

El objetivo de este trabajo es desarrollar un modelo integrador que resuelva los problemas de coloración clásicos mediante el modelo para coloración de gráficas suaves y la técnica metaheurística GRASP, usando instancias particulares a cada problema de coloración que permiten comparar cuán eficiente resultó ser el modelo propuesto; observando así el comportamiento que tiene la coloración de gráficas suaves al resolver otros tipos de problemas clásicos de coloración.

Agradecimientos

A mis padres y abuela, por su apoyo incondicional.

A mis asesores, el Dr. Miguel Ángel Gutiérrez Andrade y el Dr. Pedro Lara Velázquez, por apoyo y consejos académicos y para la vida.

Al posgrado en Ciencias y Tecnologías de la Información por brindarme la oportunidad así como al CONACyT por la beca que me permitió estudiar la maestría.

A los que se fueron, los que llegaron y los que estuvieron hasta el final.

Índice

Introducción

En el mundo, es posible representar datos de forma descriptiva; es decir, usando adjetivos como grande, pequeño, lento, etc. O de forma numérica, pudiendo ser discretos o continuos.

Existe una clara división entre las matemáticas continuas y discretas. Principalmente debido a que las estructuras básicas y métodos son diferentes por ambos lados, ya que las matemáticas continuas usan conceptos como la continuidad y el cambio continuo mientras que los procesos en matemáticas discretas son contables.

La visión tradicional es que el tiempo y el espacio son continuos, y las leyes de la naturaleza están descritas por ecuaciones diferenciales. Existen excepciones, como en el caso de la química, mecánica estática y física cuántica que, están basados en cierto grado de discretitud. Sin embargo, los objetos discretos que los componen (moléculas, átomos, partículas elementales) viven en un mundo continuo.

Los diferentes problemas en ciencias de la computación pueden ser divididos en dos categorías: problemas de decisión y de optimización.

El problema de optimización se define como: dado un conjunto de soluciones factibles S, encontrar la mejor solución al máximo o mínimo del conjunto de acuerdo con una función objetivo (función costo/beneficio). Los problemas de optimización se dividen en: optimización continuos; estos tienen variables de decisión continuas las cuales toman sus valores dentro de un conjunto de números reales con propiedades de continuidad y los de optimización discretos, con variables de decisión discretas. A estos últimos también suele llamárseles de optimización combinatoria.

Algunos ejemplos de problemas de optimización discretas o combinatoria son: el problema de transporte, el problema de asignación, el árbol de expansión mínima, el problema de la mochila, el problema del agente viajero, el problema del Sudoku, el problema de coloración, entre otros [20].

En cuanto a los problemas continuos se han propuesto una colección de funciones de prueba escogidas para reflejar diferentes facetas que podrían resultar difíciles para un algoritmo, las cuales son: unimodalidad, continuidad, diferenciabilidad, convexidad, el número de óptimos locales y su distribución, así como de óptimos globales y su distribución. Estas funciones son: la función de la esfera, la función de Easom, la función de Griewangk, la función de Langermann, la función de Michalewicz, la función de Rastrigin y las seis jorobas del camello [20].

No todos los algoritmos que resuelven problemas como los anteriormente descritos, tienen la misma dificultad al momento de resolverse debido a que la dificultad varía de acuerdo con el comportamiento del algoritmo que lo resuelve; por ejemplo, el número de operaciones que necesita como adicciones o comparaciones [20]. Un método para analizar este comportamiento está basado en la teoría de complejidad [30].

La teoría de la complejidad computacional estudia los recursos requeridos durante el cálculo para resolver un problema. Los recursos comúnmente estudiados son el tiempo (número de pasos que ejecuta un algoritmo) y el espacio (cantidad de memoria utilizada). En esta área de trabajo, existe una clasificación de tipos de problemas que se abordan dentro de la teoría de la complejidad.

Existe un grupo de problemas denominados P ("polinomial"), que están definidos por algoritmos con funciones que se resuelven en tiempos polinomiales.

Asimismo, existe un grupo denominado NP ("polinomial, no-determinista", por sus siglas en inglés), para estos problemas, no se conocen algoritmos con tiempo polinomial que los resuelvan, y generalmente se cree que tampoco existen. También hay una subclase importante de problemas NP conocida como NP-Completa, la cual es una clase que representa el conjunto de todos los problemas x en NP para los cuales es posible reducir cualquier otro problema NP y a x en tiempo polinomial. Existe otra clase denominada NP-Duro, estos son problemas que son al menos tan difíciles como los problemas NP-Completos. Los problemas de la clase NP-Duro no tienen que estar en NP y no tienen que ser problemas de decisión. La definición precisa aquí es que un problema x es NP-Duro, si hay un problema NP-Completo y, tal que y es reducible a x en tiempo polinomial. Pero desde que cualquier problema NP-Completo puede ser reducido a cualquier otro problema Np-Completo en tiempo polinomial, todos los problemas Np-Completos pueden ser reducidos a cualquier problema Np-Duro en tiempo polinomial. Entonces, si existe una solución a un problema NP-Duro en tiempo polinomial, existe una solución a todos los problemas NP en un tiempo polinomial [1].

Para los problemas de optimización en la clase NP-Duro, a la fecha, no se conocen algoritmos que los resuelvan en tiempo polinomial. Es por esto que, en las últimas décadas, se han desarrollado algoritmos de tipo heurístico o estocástico para resolver instancias grandes de problemas que pertenecen a esta clase. Estos algoritmos, no necesariamente encuentran la mejor solución al problema, pero en general se pueden obtener "buenas soluciones" cuando se aplican, entendiendo "buena solución" en términos de cercanía al valor óptimo. La ventaja principal de este tipo de algoritmos es que son rápidos y corren en un tiempo polinomial. En computación, es fundamental encontrar algoritmos con buenos tiempos de ejecución y que proporcionen "buenas soluciones", una heurística es un algoritmo que trata de satisfacer al menos uno de estos requerimientos, sino es que ambos [20]. Una buena heurística debe de satisfacer las siguientes propiedades [9]:

- Una solución puede ser obtenida con un esfuerzo computacional razonable.
- La solución proporcionada debe tener alta probabilidad de ser óptima o cercana al óptimo.
- La probabilidad de obtener una "mala" solución debe de ser baja.

Además de la necesidad de encontrar buenas soluciones a problemas difíciles en un tiempo aceptable, existen otras razones para usar métodos heurísticos como [9]:

- El problema tiene una naturaleza tal que, ningún método para encontrar el óptimo del problema es conocido.
- Aunque exista un método polinomial para resolver el problema de manera exacta, este no puede ser implementado con el hardware disponible debido al alto costo computacional.
- El método heurísitco es más flexible que el exacto y permite incorporar condiciones difíciles de modelar.
- Se usa como parte de un procedimiento global que garantiza optimizar el problema.

Una metaheurística (el prefijo "meta" significa "más allá") es un proceso diseñado para encontrar, generar, seleccionar o guiar otras heurísticas. De manera general, se puede decir que metaheurística se refiere a una estrategia de nivel superior que guía a otras heurísticas en la búsqueda de soluciones factibles a problemas más complejos [20].

Existen estimadores que nos pueden brindar un índice de la calidad de un algoritmo metaheurístico:

- Comparación con una solución óptima: a veces se tiene el valor óptimo para un ejemplo (instancia) de tamaño reducido o bien se diseña una instancia para forzar lo que se conoce como óptimo de antemano. Se suele medir la desviación porcentual con respecto a la óptima si C_h es el costo y C_{opt} es el óptimo: $\frac{C_h - C_{opt}}{C_{opt}}\ x\ 100$.
- Comparación con una cota: la bondad de esta medida dependerá de la bondad de la cota (cercanía al óptimo) si la cota no es buena, la comparación no es de mucho interés.
- Comparación con otras heurísticas: es uno de los métodos más empleados en los problemas NP-Duros y es muy bueno especialmente para problemas bien definidos.

En general, podemos hablar de instancia como datos de entrada y la información suficiente para obtener una solución; mientras que un problema es una colección de instancias del mismo tipo. Se dice que un algoritmo resuelve un problema, si puede aplicarse a cualquier instancia y siempre garantiza una solución factible.

Uno de los problemas discretos más interesantes que existen del tipo NP-Duro es el problema de coloración de gráficas pues no hay algoritmos que en tiempo polinomial den buenas aproximaciones como ocurre con otros problemas de optimización combinatoria siendo resueltos de forma exacta únicamente cuando se tienen instancias muy pequeñas; debido a esto, el uso de metaheurísticas se vuelve necesaria a partir de ese punto; en distintos trabajos [7] [13] [26] existen indicios de cuáles son los mejores algoritmos en términos de complejidad computacional y eficiencia para encontrar soluciones a los problemas de coloración, estás son: búsqueda tabú, GRASP (Greedy Randomized Adaptive Search Procedure, en inglés), recocido simulado y algoritmos evolutivos.

En la actualidad, la coloración de grafos juega un rol muy importante en distintas aplicaciones del mundo real [4]. Las aplicaciones de coloración de grafos incluyen la asignación de frecuencias en problemas de telecomunicación, asignación de registros en un

sistema operativo, programación de tareas, problemas de asignación en vuelos, resolución de juegos recreativos como el Sudoku, entre otras [5].

El problema de coloración grafos, a veces también llamado coloración de vértices, consiste en la asignación de colores a cada vértice de tal manera que cualquier par de nodos adyacentes no tengan el mismo color. A partir del concepto de coloración de un grafo y con base en las características y restricciones del problema que se busca resolver son planteadas generalizaciones de este problema descritos a continuación.

El problema de coloración mínima, que consiste en asignar un color a cada vértice de forma que cualquier par de vértices adyacentes no tengan el mismo color. Al mínimo valor de colores necesarios para satisfacer las restricciones del problema se le conoce como número cromático. Algunos ejemplos que se pueden platear con este tipo de coloración son: la programación de exámenes, coloración de un mapa y determinación de conglomerados [31].

La coloración equitativa consiste en que los vértices de un grafo G sean coloreados con k-colores, tales que no haya vértices adyacentes que reciban el mismo color y el tamaño de las clases de color difiere en a lo más una [8]. Con este tipo de coloración se plantean problemas del tipo asignación de horarios, particionamiento y balanceo de carga, entre otros.

La coloración débil consiste en colorear el grafo con un menor número de colores al de la coloración mínima con el objetivo de obtener el menor número de incompatibilidades, es decir, una coloración menor cantidad de colores al número cromático. Este modelo se usa para las comunicaciones entre los nodos de una red y en general para las comunicaciones síncronas en sistemas asíncronos [32].

La coloración robusta contiene dos gráficas complementarias, la primera del tipo de coloración mínima, la cual es una restricción del problema y la segunda es una gráfica en las aristas complementarias ponderadas las cuales representan incompatibilidades, busca minimizar la suma de las aristas penalizadas. Este problema se distingue del problema de coloración mínima porque permite obtener soluciones donde no solo es importante encontrar soluciones válidas, sino también que sean estables [6]; por ello, es aplicado en problemas de coloración con recursos restringidos como control de tráfico aéreo, asignación de frecuencias, análisis de clústeres, entre otros.

La coloración de gráficas suaves (CGS) es una generalización del problema de coloración [3] en donde dada una gráfica completa con ponderaciones en cada una de las aristas, buscamos encontrar el valor mínimo de la dureza; es decir, la suma de las aristas pintadas con el mismo color en ambos extremos; asimismo, existen otros parámetros importantes en la coloración de gráficas suaves como: la solidez, la cual proporciona un valor promedio de las penalizaciones de la gráfica y resiliencia que permite encontrar una cantidad adecuada de colores. Investigaciones recientes [10] [11] enfocan el problema de coloración de gráficas suaves para el reconocimiento de patrones; es decir, clasificar grandes cantidades de objetos usando propiedades entre ellos y sin la necesidad de intervención humana para

hacer un entrenamiento, con lo cual se puede afirmar que el modelo es superior a otros de la inteligencia artificial que sí necesitan entrenamiento.

Sin embargo, los modelos que existen hasta el momento únicamente resuelven un problema de coloración particular y aunque se han hecho esfuerzos por crear un modelo integrador, estos son exclusivos del problema en cuestión que se busca resolver [6]. El crear un modelo integrador capaz de resolver distintos problemas de coloración permitiría resolver instancias exclusivas de cada modelo, brindando así la posibilidad de obtener soluciones tan buenas o mejores a las de los algoritmos exclusivos para cada uno de ellos.

La coloración de gráficas suaves es sumamente flexible [6] y se sabe que muchos problemas de coloración pueden mapearse con este modelo. Este trabajo es un estudio de cómo se comporta el problema CGS para resolver otros problemas de coloración de gráficas mediante el uso de instancias benchmark clásicas de cada uno de los problemas que se estudiarán.

En el Capítulo 1, se abordan las clases de problemas continuos y discretos, así como su complejidad computacional. También se hace una introducción de los problemas de coloración de gráficas.

En el Capítulo 2, se hace una descripción a detalle del problema de coloración de gráficas suaves, así como de los problemas de coloración clásicos como son: la coloración mínima, coloración equitativa, coloración de gráficas débiles y coloración robusta; definiendo cómo es que se mapean al modelo de coloración de gráficas suaves. Se describen los mejores algoritmos de solución metaheurísticos para el problema de coloración de gráficas y un método exacto para hasta 20 instancias el cual permite evaluar la propuesta en capítulos posteriores.

En el Capítulo 3, se aborda el uso de las metaheurísticas más apropiadas para resolver problemas de coloración para instancias mayores a 20 vértices, así como los algoritmos básicos de cada una de ellas.

En el Capítulo 4, se detalla la técnica metaheurística GRASP y cómo se implementa en el modelo de coloración de gráficas suaves así como los puntos de mejora que propuestos y la descripción de cada una de las funciones que componen el algoritmo.

En el Capítulo 5, se describe los orígenes y características de las instancias empleadas particulares a cada problema específico de coloración, así como sus formatos de entrada y salida.

En el Capítulo 6, se hace un análisis de resultados obtenidos del modelo propuesto para cada problema mediante la comparación con óptimos bien conocidos, otras técnicas metaheurísticas e instancias preparadas para obtener ciertos tipos de coloración. Evaluando así su calidad.

Finalmente, se presentan las conclusiones y el trabajo subsecuente.

Objetivos

Objetivo general

- Desarrollar un algoritmo que resuelva varios problemas clásicos de coloración utilizando el enfoque de coloración de gráficas suaves

Objetivos particulares

- Realizar un estudio del estado del arte
- Compilación de instancias clásicas de cada problema a estudiar
- Desarrollo de los algoritmos de solución
- Análisis de resultados y conclusiones

Metodología

Para el estudio del estado del arte se revisó la literatura correspondiente a cada problema de coloración clásico; esto es: la coloración de gráficas mínimas, la coloración de gráficas débiles, la coloración equitativa y la coloración robusta junto a las características de cada problema y las cuales a su vez, les permiten ser abarcados por el problema de coloración de gráficas suaves; asimismo, también se revisó el problema CGS y las características que posee como la dureza y la resiliencia.

Para poder obtener las instancias particulares a cada problema de coloración, se hizo una investigación a fondo para encontrar instancias adecuadas a cada uno. En el caso del problema de coloración mínima y débil, se concluyó que los problemas DIMACS son el mejor punto de partida para resolver este tipo de instancias. Para el problema de coloración robusta, las instancias fueron facilitadas por Javier Ramírez, coautor del artículo "The robust coloring problem" publicado en el European Journal of Operational Research. En cuanto a la coloración equitativa, a pesar de haber podido encontrar diversos trabajos en la literatura relacionados con este tipo de coloración, no fue posible obtener instancias de ninguno de ellos, razón por la cual se optó por desarrollar instancias lo más adecuadas al problema, estás se describen con mayor extensión en el Capítulo 5.

Debido a la complejidad que representa la solución de los problemas de coloración, una parte importante para poder crear un modelo de solución integrador es el uso de metaheurísticas, por ello, se revisó aquella literatura donde implementaran o compararan diversos tipos de metaheurísticas para solucionar problemas de coloración y así poder tomar una decisión sobre cuál implementar en el modelo con lo cual, se llegó a la conclusión de que un GRASP es la técnica más adecuada para los requerimientos específicos del trabajo.

Finalmente, con el fin de comparar los resultados del modelo desarrollado, se implementaron diversas técnicas de comparación la cuales son: usando un óptimo conocido de antemano para las coloraciones mínimas y débiles; comparando los resultados con otras técnicas metaheurísticas para el caso de la coloración robusta y observando las coloraciones resultantes en la coloración equitativa. Estos resultados son discutidos en el Capítulo 6. A partir de los resultados obtenidos en estos apartados, se elaboraron las conclusiones correspondientes.

Capítulo 1. Marco de referencia

Particionar un conjunto de objetos en clases de acuerdo con ciertas reglas, es un proceso fundamental en matemáticas debido a que permite modelar distintas aplicaciones del mundo real como la asignación de registros, programación de tareas, coloración de mapas, balanceo de cargas en servidores, etc. Un conjunto de reglas conceptualmente simples dicta si cada par de objetos están o no admitidos en la misma clase. La teoría de la coloración de grafos resuelve esta situación. Los objetos forman el conjunto de vértices $\mathrm{V}(G)$ de un grafo G, dos vértices unidos por una arista en G, pueden pertenecer a la misma clase o a clases diferentes; para distinguir las distintas clases se usan un conjunto de colores C, y la división entre clases es dada por una coloración $\varphi : \mathrm{V}(G) \to \mathrm{C}$, donde $\varphi(x) \neq \varphi(y)$ para todos los xy pertenecientes al conjunto de aristas $E(G)$ de G. Si C tiene una cardinalidad k, entonces φ es *k-coloreable*, y cuando k es finita, usualmente se asume que $C = \{1,2,3, \dots, k\}$. Para $i \in C$ el conjunto $\varphi^{-1}(i)$ es la i-ésima clase de color. Por eso, cada clase de color forma un conjunto independiente de vértices; esto es, no existen dos de ellos unidos por una arista. El mínimo cardinal k para el cual G tiene una k-coloración es el *número cromático* [3].

El principal uso de los problemas de coloración es la asignación a usuarios de recursos escasos y con frecuencia estos se centran en resolver problemas de calendarización, asignación de horarios y problemas de planificación gracias a los cuales aparecieron distintas generalizaciones del problema para satisfacer una necesidad específica; este es el caso de los que se conocen como los problemas clásicos de coloración: coloración mínima, equitativa, débil y robusta. Por ejemplo, la coloración mínima surgió en un principio para la coloración de mapas; sin embargo, aunque se hicieron esfuerzos para adaptarlo a otro tipo de problemas, tales como la programación de tareas o el balanceo de carga en una computadora, este modelo resultó ser muy restrictivo en el sentido de no poder resolver otros problemas de calendarización que pueden ser modelados como problemas de coloración. Finalmente surgieron otros modelos como la coloración débil la cual busca reducir el número de incompatibilidades que proporciona la coloración mínima o la coloración equitativa que busca repartir los recursos o colores lo más equitativamente posible, y en años más recientes la coloración robusta, la cual puede modelar distintos tipos de incompatibilidades que surgen en este tipo de problemas y la coloración de gráficas suaves utilizado en la programación de eventos susceptibles a cambios como en la asignación estable de frecuencias del espectro electromagnético.

La versión suave es útil cuando grafos grandes y distribuidos deben de ser coloreados en tiempo real: el tamaño de los grafos, restricciones de tiempo, complejidad combinatoria y latencia de comunicación hacen que esta aplicación sea diseñada para trabajar con grafos que sean coloreados más propiamente dentro del cual puede contener algunos conflictos. Tales aplicaciones surgen en coordinaciones en grandes redes distribuidas; por ejemplo, los nodos pueden representar recursos que deben de ser repartidos, los colores pueden

representar slots de tiempo en horarios cíclicos y las aristas pueden representar exclusión mutua entre los recursos [33].

La coloración de gráficas suaves es considerada una generalización del problema de coloración de gráficas. En este problema tenemos que, dada una gráfica completa con ponderaciones en las aristas, el objetivo es encontrar una coloración que minimice la suma de las aristas con el mismo color en ambos extremos. Asimismo, existen otros parámetros importantes como: la solidez, que proporciona un valor promedio de las penalizaciones de la gráfica y la resiliencia, que permite encontrar una cantidad adecuada de colores [6].

Es precisamente gracias a estos parámetros que se distingue de otros tipos de coloración, como la coloración robusta, debido a que, la CGS permite modelar restricciones de recursos, además, proporciona una pauta de la cantidad más adecuada de colores a usar.

Debido a que la resolución de este problema es del tipo NP-duro, es necesario el uso de metaheurísticas en instancias mayores a 20 vértices. No obstante, para problemas con instancias pequeñas; es decir, menores o iguales a 20 vértices, existe un algoritmo de programación lineal entera que permite encontrar la solución óptima del problema desarrollado [6]; este algoritmo se ejecuta en el software licenciado General Algebraic Modeling System (GAMS, por sus siglas en inglés). Gracias a este proceso, es posible evaluar el desempeño de cualquier metaheurística comparando los resultados que esta arroja contra el óptimo encontrado por GAMS.

La coloración de gráficas suaves se ha aplicado recientemente para solucionar problemas como el reconocimiento de patrones [10] [11] el cual se puede entender como clasificar grandes cantidades de objetos físicos o abstractos con el propósito de extraer información útil que permita establecer propiedades entre agrupaciones de dichos objetos. Una manera de cumplir con esta tarea consiste en definir una distancia entre los objetos. Debido a esto, si dos objetos están muy cerca, de acuerdo con una métrica dada, es muy posible que ellos estén agrupados en la misma clase. Por otro lado, si los objetos están muy lejos entre sí, es muy probable que pertenezcan a clases distintas. Sin embargo, antes de poder usar una métrica para extraer las distancias entre los objetos, es necesario hacer una normalización de la base de datos a usar, pues algunos datos están representados con diferentes escalas. Este proceso consiste en tomar los valores máximos y mínimos de cada columna, mismos que serán los límites, los demás valores se sustituyen garantizando que todos los valores del atributo tengan una distribución entre 0 y 1 usando la fórmula:

$$x_i = \frac{x_i - x_{min}}{x_{max} - x_{min}}$$

Con la finalidad de evitar la división entre cero, si los valores máximos y mínimos son los mismos, se dará un valor de 1 a todos los valores de la columna [11].

Una de las métricas más usadas para medir esta distancia es la distancia Euclideana cuadrada [27], que se define como, dados dos vértices x, y, la distancia entre ellos está dada por:

$$d_{xy} = \sum_{i=1}^{n} (x_i - y_i)^2$$

Donde i se refiere a una dimensión específica de los elementos x, y.

La característica clave de este tipo de clasificación es que se hace únicamente usando las propiedades entre los objetos y sin la necesidad de intervención humana para hacer un entrenamiento; es decir, un aprendizaje no supervisado; por lo anterior se puede afirmar que el modelo es superior a otros de la inteligencia artificial que sí necesitan entrenamiento o supervisión para su aprendizaje. La ventaja principal se tiene cuando se trata con un problema donde no es posible entrenar al modelo dado que no se han establecido o no se conocen de antemano cómo se clasifican los objetos y es necesario encontrar una manera de clasificarlos de una forma "confiable". Lo cual a su vez puede dar nuevas pautas para la investigación entre objetos que salgan dentro de una misma clasificación y no guarden ninguna relación aparente.

La coloración de gráficas suaves ha demostrado ser un modelo de gran flexibilidad pues ha sido aplicada para resolver diversos tipos de instancias que hacen uso de las propiedades de dureza, rigidez y resiliencia así como su comportamiento para diferentes distribuciones de ponderaciones. Esta técnica ha sido aplicada para resolver varias instancias con excelentes resultados, como en los casos del reconocimiento multilenguaje donde se clasifican diversos lenguajes de acuerdo con la familia a la que pertenecen haciendo una comparación lingüística entre lenguajes ibéricos [27], el reconocimiento de literatura inglesa en el cual se clasifican un conjunto de libros escritos en diversas clases de inglés [11] y el repositorio de Aprendizaje Automático de la Universidad de Irvine California (UCI, por sus siglas en inglés) que posee instancias entre las que se encuentran: Iris, Car Evaluation, Stone Flakes y Wine [10]. De igual forma, existe una propuesta en [6] en la cual se proporciona cómo pueden ser mapeados algunos de los problemas de coloración clásicos y posteriormente resueltos mediante este modelo por lo cual puede dar la pauta necesaria para crear un modelo integrador a los tipos de coloraciones más importantes y los problemas que se plantean con ellas; además, si se resuelve un problema usando coloración de gráficas suaves, se espera no solamente que se tenga el mismo resultado o incluso un mejor resultado que los algoritmos exclusivos del problema en cuestión, sino que aporte información adicional al problema gracias a la solidez y resiliencia.

Algo a tomar en cuenta al momento de hacer el mapeo de los problemas de coloración clásicos a coloración de gráficas suaves es que las matrices de ponderación que se utilizan en los problemas de coloración mínima, equitativa y débil son matrices binarias, y las matrices del problema de coloración robusta es una mezcla de matriz binaria con valores reales. Y debido a que las matrices binarias usadas en los problemas de coloración mínima, débil y equitativa no proporcionan mucha información, éstas se ponderan en un intervalo (0,1); no obstante, cabe mencionar que, en principio, para coloración de gráficas suaves se puede considerar cualquier número real en las ponderaciones [6].

1.1. Definición del problema de coloración de gráficas suaves

Sea $G = (V, E)$ una gráfica completa no dirigida; es decir, el conjunto de vértices $V = \{1, 2, \dots, n\}$ en G y todas sus aristas posibles (i, j), donde:

$$| V | = n; | E | = n(n-1)/2.$$

Se define una penalización en cada arista (i, j), denotada por p_{ij}, tal que:

$$p_{ij} \geq 0, \ \ \forall (i,j) \in E.$$

Una función de coloración en los vértices de $G = (V, E)$ se define como:

$$C^k : \{1, 2, \dots, n\} \rightarrow \{1, 2, \dots, k\},$$

donde k es el número total de colores $1 \leq k \leq n$ que identifica $C(i)$ como el color en el vértice i. Para una coloración C^k en una gráfica, la función de dureza de la coloración C^k está dada por:

$$H(C^k) = \sum_{(i,j) \in E,\, C^k(i) = C^k(j)} p_{ij}\,.$$

El objetivo del problema de coloración de gráficas suaves, es encontrar la coloración C^k_{op} que minimiza la dureza $H(C^k_{op})$.

Solidez y resiliencia

Dada una coloración con k colores en una gráfica completa con n vértices, el número promedio m de vértices pintados con el mismo color es $m = n/k$ y el número de aristas compartidas por m vértices son las combinaciones $\mathrm{C}\,(m, 2) = m(m-1)/2$, por lo que el número promedio de penalizaciones incluidas en la función de dureza debe de ser proporcional al número promedio de vértices pintados con el mismo color multiplicado por el número de colores. Definimos la función de *solidez* de una coloración como la dureza de una gráfica dividida por el número medio de aristas que contribuyen a la dureza:

$$S\left(C^K_{op}\right) = \frac{H\left(C^K_{op}\right)}{km\dfrac{m-1}{2}} = \frac{2H\left(C^K_{op}\right)}{k\dfrac{n}{k}\left(\dfrac{n}{k}-1\right)} = \frac{2kH\left(C^K_{op}\right)}{n\,(\mathrm{n}-\mathrm{k})}.$$

La resiliencia de una coloración C^k se define como el porcentaje en que disminuye la solidez de una coloración con $k-1$ colores con respecto a la realizada con k de ellos y se puede expresar como sigue:

$$R\left(C_{op}^{K}\right)=\frac{S\left(C_{op}^{K-1}\right)-\ S\left(C_{op}^{K}\right)}{S\left(C_{op}^{K}\right)}.$$

Si resulta que, al añadir un color adicional a la coloración de la gráfica, hay un aumento significativo en su resiliencia, eso significa que hemos encontrado un número adecuado para clasificar.

Finalmente, si obtenemos la resiliencia para todas las coloraciones consecutivas desde 1 hasta n, y si pensamos en cada color como un grupo para clasificar a los vértices (objetos), los valores más grandes mostrarán las mejores opciones de número de clases que se pueden utilizar para clasificar el conjunto.

1.2 Modelo binario entero para el problema de coloración de gráficas suaves

Dado un grafo no dirigido $G=(V,E)$ con $|V|=n$ y $|E|=n(n-1)/2$, queremos obtener una coloración C_{op}^{K} usando k colores. Para resolver este problema, se define el siguiente modelo de programación binaria:

$$\min z=\sum_{(i,j)\in E} p_{ij}\,y_{ij}$$

Sujeto a:

$$\sum_{l=1}^{k} x_{il}=1\ \forall\, i\ \in\{1,\dots,n\}$$

$$x_{il}+x_{jl}-1\ \leq y_{ij}\ \forall\,(i,j)\in E\ \ y\ \ \forall\, l\in\{1,\dots,k\}$$

$$x_{il}\in\{0,1\}\ \forall\ i\ \in E\ \ y\ \ \forall\, l\in\{1,\dots,k\}$$

$$y_{ij}\in\{0,1\}\ \forall\ (i,j)\in E.$$

Donde:

$$x_{il}=\begin{cases}1 \text{ si } C(i)=l\ \forall\, i\ \in\{1,\dots,n\}\ y\ \forall\, l\in\{1,\dots,k\}\\ 0 \text{ en otro caso}\end{cases}$$

$$y_{ij}=\begin{cases}1 \text{ si para alguna } l\ \in\{1,\dots,k\} \text{ se cumple que } x_{il}=\ x_{jl}=1\\ 0 \text{ en otro caso}\end{cases}$$

El conjunto de ecuaciones anteriores garantiza que cada vértice tiene asignado solamente un color; el conjunto de desigualdades hace que la variable y_{ij} valga 1 si se colorean los vértices i y j con el mismo color y cero si los vértices i y j tienen colores diferentes. En el primer caso se activa la penalización p_{ij} en la función objetivo z. La solución al problema

con las restricciones obtiene la coloración mínima C_{op}^{K} tomando $C_{op}^{K}(i) = l$ si $x_{il} = 1$, donde $x_{il}\ \forall\ i \in \{1, \dots, n\}$ y $l \in \{1, \dots, k\}$, son los valores de las variables de decisión obtenidos en la solución óptima del problema y el valor mínimo z^* de la función objetivo es igual a $H\left(C_{op}^{K}\right)$.

El sistema tiene nk variables x_{il} y $n(n-1)/2$ variables y_{ij} por lo que el número total de variables binarias en el modelo es $nk + n(n-1)/2$. Por otro lado, el número de ecuaciones es igual a n y el número de desigualdades es igual a $kn(n-1)/2$, de modo que el número total de restricciones es igual a: $n + kn(n-1)/2$.

Capítulo 2. Problemas de coloración

2.1 Coloración de vértices

Uno de los principales problemas que se encuentran dentro de la teoría de gráficas es el problema de coloración de vértices de una gráfica. Este problema es muy conocido y consiste en asignar un color a cada vértice de forma que cualquier par de vértices adyacentes no estén pintados con el mismo color. El problema surgió como un intento por resolver la conjetura de los cuatro colores, con el teorema de Brooks en 1941, que da una cota superior para el número cromático de cualquier grafo.

La coloración de vértices de un grafo $G = (V, E)$ es un mapeo $C: V \to S$ tal que $c(v) \neq c(w)$ dondequiera que v y w sean adyacentes. Los elementos del conjunto S son llamados colores disponibles. Todo lo que interesa sobre S es su tamaño: típicamente, preguntaremos por el entero k más pequeño para el cual G tiene una k-coloración, una coloración de vértices $C: V \to \{1, \dots, k\}$. Este k es el *número cromático* de G; se denota por $\chi(G)$. Un grafo G con $\chi(G) = k$ es llamado *k-cromático*; si $\chi(G) \leq k$ [3].

Cada grafo G puede ser coloreado por $\Delta(G) + 1$ colores, donde $\Delta(G)$ es el máximo grado de G, y él describe los grafos para los cuales $\Delta(G)$ colores no son suficientes.

Como consecuencia de las definiciones anteriores podemos derivar que:

Cada grafo G con m aristas satisface

$$\chi(G) \leq \frac{1}{2} + \sqrt{2m + \frac{1}{4}}$$

Por lo tanto, si existe un vértice v coloreado de G con $k = \chi(G)$ colores. Entonces G tiene al menos una arista entre dos clases de colores; si no, podríamos haber usado el mismo color para ambas clases. Por eso, $m \geq \frac{1}{2}k(k-1)$. Resolviendo esta desigualdad para k, obtenemos la afirmación comprobada [3].

Por lo tanto, y resumiendo lo anterior, podemos afirmar que un grafo G es $k - coloreable$ si es posible asignar a cada uno de sus vértices uno de k colores de tal manera que no existan vértices adyacentes del mismo color. El valor mínimo de k que hace a G $k - coloreable$ es llamado el número cromático y se denota por $\chi(G)$.

2.2 Problema de coloración mínima

La coloración mínima consiste en asignar un color a cada vértice de forma que cualquier par de vértices adyacentes no tengan el mismo color.

Formalmente, dado un grafo $G = (V, E)$, con $|V| = n$, se define una función de coloración como una aplicación

$$C: \{1, \dots, n\} \to \{1,2, \dots\}$$

Que identifica a $C(i)$ como el color del vértice $i \in V$ de forma que dos vértices adyacentes $(i, j) \in E$ no tiene el mismo color; es decir, $C(i) \neq C(j)$

Dada una función de coloración C de un grafo G, al conjunto de vértices que tienen asignado el mismo color se le llama clase de color:

$$V_c(l) = \{i \in V \mid C(i) = l\} \quad \forall\, l \in \{1,2, \dots\}$$

Una k-coloración es una función de coloración que no utiliza más de k colores:

$$C^k: \{1, \dots, n\} \to \{1,2, \dots, k\}$$

Un grafo es k-coloreable si admite una k-coloración. Al mínimo valor k tal que G es k-coloreable se le llama el número cromático del grafo y se denota por $\chi(G)$ colores.

Problema de coloración mínima planteado con coloración de gráficas suaves

En este problema, se tiene una gráfica no dirigida n, en la cual al tenerse una arista que conecta a dos vértices, dichos vértices no pueden pintarse con el mismo color. Este modelo se puede plantear con la siguiente función objetivo:

$$\min z = \sum_{(i,j) \in E} p_{ij}$$

Del mismo modo, si deseamos emplear el modelo binario para resolver el problema, la función objetivo se plantea como sigue:

$$\min z = \sum_{(i,j) \in E} p_{ij} y_{ij}$$

Donde $A = (p_{ij})$ es la matriz de adyacencia que cumple para cada entrada uno de dos valores $p_{ij} = 0.5$ si los vértices i, j están conectados y $p_{ij} = \varepsilon$ si no lo están donde ε es un número muy pequeño; por ejemplo, $\varepsilon = 0.00001$. Si resolvemos la gráfica para todas las coloraciones consecutivas desde 1 hasta n, una coloración con el número cromático será aquella donde se obtenga por primera vez una dureza menor a 0.5.

2.3 Problema de coloración de gráficas débiles

Cuando se diseñan algoritmos distribuidos para calcular estructuras en grandes sistemas descentralizados, uno de los principales problemas es romper las simetrías. Incluso aunque varios nodos en la red pueden ejecutar el mismo algoritmo y a pesar de las posibles simetrías en la topología de la red, los diferentes nodos generalmente necesitan terminar un algoritmo con diferentes resultados; es decir, diferentes colores en el caso de un algoritmo de coloración [22].

Por ello, es necesario un algoritmo que establezca acuerdos naturales entre el número de colores de la coloración final y la medida en que el problema de coloración estándar tiene que ser resuelto (es decir, el requerimiento de rotura de simetría restante).

De manera general, la *coloración de gráficas débiles* busca colorear una gráfica con menos colores que el número cromático, con el objetivo de crear una gráfica con el menor número de incompatibilidades; es decir, una coloración mínima con menor número de colores al número cromático [6].

Formalmente, dado un grafo $G = (V, E)$, se asigna un valor $c(v) \in \{1,2,\ldots,k\}$, para cada vértice $v \in V$, tal que cada vértice no aislado es adyacente al menos a un vértice con diferente color; es decir, para cada vértice no aislado $v \in V$, hay un vértice $u \in V$ con $\{u, v\} \in E$ y $c(u) \neq c(v)$.

Problema de coloración de gráficas débiles planteado con coloración de gráficas suaves

Este es el modelo de coloración mínima cuando se tienen menos colores que el número cromático, por eso, también en este caso $p_{ij} = 0.5$ y corresponde a un vértice que está mal coloreado. Dentro del esquema de coloración de gráficas suaves, se busca minimizar la función de dureza que incluye todas las p_{ij}, cuyos extremos están pintados con el mismo color. La función de dureza da exactamente el número de incompatibilidades en la coloración de la gráfica debido a que cada penalización representa un vértice mal coloreado. En general, y usando el modelo de la coloración de gráficas suaves, cualquier solución con menos colores al número cromático es una coloración débil, por lo que resolviendo el problema de coloración mínima también es resuelto de coloración de gráficas débiles.

2.4 Problema de coloración equitativa

La coloración equitativa fue introducida por Walter Meyer en 1973 [23] para satisfacer problemas de calendarización, propone una situación donde los vértices de una gráfica representan una colección de tareas a realizar y una arista conecta a dos tareas que no se deben realizar al mismo tiempo.

La coloración de la gráfica representa el reparto de tareas en subconjuntos las cuales pueden realizarse simultáneamente; por esto, el número de colores en la coloración corresponde al número de pasos requeridos para realizar la tarea entera. A causa de consideraciones de equilibrio de carga, se desea que se realice un número igual o casi igual de tareas en cada paso y este balance es exactamente lo que se obtiene con una coloración equitativa.

Las aplicaciones de la coloración equitativa pueden encontrarse en la planificación y asignación de horarios. Considerando; por ejemplo, un problema de asignación de horarios en una universidad. Como se sabe, este problema puede ser modelado como la coloración de vértices de una gráfica G cuyos nodos corresponden a las clases, las aristas corresponden a conflictos de tiempo entre clases y los colores a las horas. Si el conjunto de salones está restringido, entonces es posible forzar a dividir los vértices en subconjuntos independientes de tamaños tan iguales como sea posible [24].

En la *coloración equitativa* los vértices de un grafo G tienen que ser coloreados con k-colores tales que no haya vértices adyacentes que reciban el mismo color y el tamaño de las clases de color difiere a lo más en un elemento.

Una coloración de vértices o simplemente coloración, de un grafo G es una asignación de colores a los vértices de G tales que no existen dos vértices adyacentes con el mismo color; el conjunto de vértices con el mismo color es llamado clase de color.

Un grafo G es k-coloreable equitativamente si el conjunto de vértices $V(G)$ pueden ser divididos en k conjuntos independientes no vacíos $V_1, V_2, \dots, V_k$ tal que $||V_i| - |V_j|| \leq 1$ para cada i y j.

El k entero más pequeño para el cual G es k-coloreable equitativamente se denomina el número cromático equitativo de G y es denotado por $\chi_e(G)$.

Problema de coloración equitativa planteado con coloración de gráficas suaves

La función objetivo se puede plantear de la siguiente forma:

$$\min z = \sum_{(i,j)\in E} p_{ij} = \sum_{(i,j)\in E} d_{ij}$$

En caso de querer resolver el problema mediante el modelo binario, la función objetivo cambia a la siguiente forma:

$$\min z = \sum_{(i.j)\in E} p_{ij} y_{ij}$$

Donde d_{ij} es la distancia Euclidiana cuadrada de los vértices i, j, calculada con la fórmula:

$$d_{ij} = \sum_{l=1}^{n} (x_{il} - x_{jl})^2$$

Donde l se refiere a una dimensión específica de los elementos x_i, x_j. Debido a que los vértices se encuentran distribuidos de manera uniforme y con una distancia de 0.4 entre ellos, se espera que las soluciones encontradas dividan a los vértices en conjuntos equitativos; sin embargo, para poder corroborar este hecho, es necesario hacer uso de la resiliencia donde, si se obtiene la resiliencia para todas las coloraciones consecutivas desde 1 hasta n, el valor más grande corresponderá a $\chi_e(G)$.

2.5 Problema de coloración robusta

En ocasiones, el recurso a minimizar con el planteamiento de encontrar el número cromático no es fundamental, en su lugar, lo que interesa es que una solución al problema sea estable en el sentido de que al añadir o cambiar aristas en la gráfica, la coloración de la misma continúe siendo válida. Este tipo de consideraciones muestran que el problema de coloración mínima es un modelo muy restrictivo para este tipo de situaciones [21].

En el problema de coloración robusta (PCR), consideramos dos gráficas, la primera del tipo de coloración mínima la cual es una restricción del problema la cual se debe de pintar válidamente, y la segunda que es un grafo con aristas complementarias a la primera; es decir, todas las aristas no incluidas en la primera gráfica. Cada arista en el grafo complementario tiene una penalización. Si ambos extremos de la arista complementaria comparten vértices con el mismo color, la penalización correspondiente se agrega a la función de rigidez.

Así, el objetivo en el problema de coloración robusta es, considerar a la función de rigidez como una función objetivo a minimizar y la validez de la coloración en la gráfica original cumpliendo el conjunto de restricciones.

El nivel de rigidez de la gráfica es inversamente proporcional al grado de robustez de la misma, por lo tanto, a mayor robustez en la coloración, se tiene un menor nivel de rigidez en la solución.

Este problema se puede reducir al problema de coloración mínima cuando pintamos la gráfica con el número cromático de colores (la mínima cantidad de colores que permite una coloración válida en la gráfica original), y todas las penalizaciones en el grafo complementario las hacemos cero.

De manera formal, dadas dos gráficas G y $\bar{G}$, y una función de peso $p : E(\bar{G}) \to \mathbb{R}$, la cual asigna una penalización a cada arista de $\bar{G}$, la rigidez de una k-coloración C^k de G,

denotada $R(C^k)$ es la suma de los pesos de las aristas de $\bar{G}$ que unen vértices de la misma clase cromática, esto es:

$$R(C^k) = \sum_{(i,j)\in E(\bar{G}), C^k(i)=C^k(j)} p_{ij}$$

El problema de coloración robusta consiste en determinar una k-coloración C^k de G que sea válida y tenga mínima rigidez; es decir, $R\left(C_R^k\right) = min_{C^k}\, R(C^k)$

Problema de coloración robusta planteado con coloración de gráficas suaves

Para convertir a este problema en un modelo de gráficas suaves, se considera la siguiente función objetivo:

$$\min z = \sum_{(i,j)\in E} p_{ij}$$

Asimismo, cuando se desea su resolución mediante el modelo binario, la función objetivo anterior se convierte en la siguiente:

$$\min z = \sum_{(i,j)\in E} p_{ij} y_{ij}$$

Donde se re-penalizan las aristas de la gráfica $G = (V, E)$ de la siguiente manera:

$$n^2 \text{ si } (i,j) \in E$$

$$p_{ij} \text{ si } (i,j) \in \bar{E}$$

Penalizando las aristas de la gráfica original con un valor igual a n^2 se garantiza que al encontrar una coloración con una dureza menor que n^2 se obtiene una coloración válida mínima. Esta gráfica es válida en el problema de coloración mínima porque se han excluido todas las aristas de la gráfica E, y dicha solución tiene un costo menor a n^2 ya que todas las penalizaciones son positivas y estrictamente menores que uno, siempre y cuando $p_{ij} < n^2$. Para la gráfica complementaria se conservan las penalizaciones originales. El número cromático es aquel número de colores donde se obtienen por primera vez una coloración menor que n^2. Para más colores que el número cromático, cualquier coloración óptima de gráficas suaves es una coloración óptima del problema de coloración robusta con esa cantidad específica de colores.

2.6 Otros posibles problemas de coloración

Como es mencionado en apartados anteriores, la coloración de grafos y sus generalizaciones son herramientas útiles en la modelación de una amplia variedad en problemas de planificación, asignación y reconocimiento de patrones. En contraste, también existen variaciones y extensiones del problema de coloración de grafos que permiten abarcar problemas más complejos de planificación como: la extensión de precoloración, multicoloración, co-coloración, partición de coloración y coloración con preferencias [17].

Multicoloración

En el problema de multicoloración, cada vértice v tiene una *demanda* $x(v)$, y nosotros tenemos que asignar un conjunto de $x(v)$ colores a cada vértices v, tal que sus vecinos reciban un conjunto de colores distintos. La multicoloración puede ser usada para modelar la planificación de trabajos con diferentes requerimientos de tiempo: el conjunto de colores asignados al vértice v corresponde a las $x(v)$ asignaciones de tiempo cuando trabajamos [17].

Extensión de precoloración

En este problema, algunos vértices del grafo a resolver tienen colores preasignados y tiene que resolverse el problema de la extensión de precoloración: extender la coloración de esos vértices al grafo entero usando el número mínimo de colores.

Con frecuencia se le suele emplear para resolver el problema de planificación aérea, donde existen periodos de mantenimiento en los cuales un avión no puede volar [17].

Co-coloración

En este problema permitimos a cada clase de color estar ya sea en un conjunto independiente o un grupo exclusivo, así que la división del conjunto de vértices V de un grafo $G = (V, E)$ en p grupos exclusivos y q conjuntos independientes es una co-coloración con $p + q$ colores. El valor más pequeño de $p + q$ para el cual tal partición existe es el número *co-cromático* de G [34]

Partición de coloración

Sea un grafo $G = (V, E)$, los vértices son particionados en k conjuntos sin unión $V = V_1 \cup \ldots \cup V_k$, y G, se dice que es particionado-coloreado si exactamente un v en cada V_i es coloreado y cada dos vértices adyacentes coloreados tienen colores diferentes. Análogo al problema estándar de coloración de vértices, se desea colorear el grafo con el número mínimo de colores que sean posibles. El problema de partición de coloración está motivado

por el problema de ruteo y asignación de longitud de onda en redes ópticas WDM (Multiplexado por división en longitudes de onda) [34].

Coloración con preferencias

En casi todas las situaciones reales hay algunas ligeras restricciones que pueden ser satisfechas o no, y cuando son violadas, se penaliza. Este problema consiste en encontrar una solución que minimice las penalizaciones totales o al menos mantenerlas bajo un límite dado.

Se comienza con un grafo $G = (V, E)$ y es introducido un subconjunto P de pares vw (vw no en E) en el cual algunos requerimientos son agregados. Así que ahora se tiene un grafo $G = (V, E; P)$ en el cual P es un conjunto de pares con preferencias representadas con líneas punteadas. Cada arista en E es llamada arista fuerte. Para cada par $vw \in P$ tenemos una penalización positiva $\omega(vw)$ la cual se activa cuando cualquier preferencia no está satisfecha.

Está descrito en [34] que este problema puede ser resuelto como el problema de coloración robusta.

Capítulo 3. Metaheurísticas

Como se ha dicho anteriormente, el problema de coloración de gráficas suaves, al ser una generalización del problema de coloración, es un problema de tipo NP-Duro a partir de instancias con 20 vértices, por lo que es necesario el uso de heurísticas y/o metaheurísticas en este punto.

En la literatura [7] [13] [20] [26] existen diversos indicios de cuál o cuáles son los mejores algoritmos en términos de complejidad computacional y eficiencia al momento de encontrar soluciones a los problemas de coloración, por lo cual la lista a la fecha de las posibles metaheurísticas a utilizar se ve reducida a la búsqueda tabú, GRASP, recocido simulado y algoritmos evolutivos.

3.1 Metaheurísticas para problemas de coloración

Búsqueda tabú

Los principios de la búsqueda tabú fueron creados e introducidos por Fred Glover en el año 1997 con su libro llamado "Tabu Search" [19]. La búsqueda tabú es una metaheurística que guía un procedimiento heurístico de búsqueda local en la búsqueda de optimalidad global. Su filosofía se basa en derivar y explotar una colección de estrategias inteligentes para la resolución de problemas, basadas en procedimientos implícitos y explícitos de aprendizaje. El marco de memoria adaptativa de la búsqueda tabú no solo explota la historia del proceso de resolución del problema, sino que también exige la creación de estructuras para hacer posible tal explotación. De esta forma, los elementos prohibidos en la búsqueda tabú reciben este estatus por la confianza en una memoria evolutiva, que permite alterar este estado en función del tiempo y las circunstancias. En este sentido es posible asumir que la búsqueda tabú está basada en determinados conceptos que unen los campos de inteligencia artificial y optimización.

El algoritmo básico de la búsqueda tabú es:

Comienza

1. Genera solución inicial
2. Para i = 1 hasta paro
3. Analiza soluciones vecinas: cambia dos clases de color a los vértices
4. Escoge la mejor solución
5. Aleatoriamente selecciona un vértice y asígnale un color aleatorio diferente
6. Fin Para

Fin

Recocido simulado

Se basa en la analogía entre la simulación de recocido de sólidos. Este algoritmo tiene la propiedad de que, en el límite, converge al conjunto de soluciones óptimas del problema de optimización combinatoria y en un tiempo polinomial produce buenas soluciones las cuales dependen del arte y la habilidad con que se definan sus diferentes componentes como son: las evaluaciones de la función objetivo de la nueva solución con respecto a la anterior, la estructura de vecindades y el programa de enfriamiento; así como también, el grado de refinamiento de la implantación en la computadora. Para encontrar las variables B, ∝, es necesario ir probando valores probables hasta encontrar los mejores; esto es, que den las mejores soluciones [20].

El algoritmo básico de recocido simulado es:

Comienza

1. Genera una coloración aleatoria inicial *i*
2. Evalúa el costo de la función f(i)
3. Coloca t = sqrt(V(G)); r = exp(d/t); B = .95; α = 1.05
4. Mientras un nuevo vecino en la solución sea aceptado en un ciclo completo
5. Para i = 1 hasta S
6. Genera una solución vecina j
7. Si Δf = f(i) - f(j) > 0, entonces i := j
8. De otra forma, genera un número aleatorio U(0,1)
9. Si exp(Δf / t) es más grande que número aleatorio, entonces i:=j
10. Ajustar el parámetro r = α *r
11. Fin Para
12. Ajustar el parámetro t = B*t
13. Fin Mientras

Fin

Donde r es la longitud de cada conjunto, t es el parámetro de temperatura, d es la extensión requerida para escapar de cualquier óptimo local y B, ∝ son constantes de decremento que se establecen con los mejores valores encontrados de forma empírica.

GRASP

GRASP (Greedy Randomized Adaptive Search Procedure, por sus siglas en inglés) es una técnica metaheurística introducida por Feo y Ressende en 1995. Es una técnica iterativa de muestreos aleatorizada en la cual cada iteración provee una solución al problema en cuestión. La solución correspondiente a todas las iteraciones de GRASP es mantenida como el resultado final.

Existen dos fases dentro de cada iteración de GRASP: la primera construye una solución inicial inteligentemente mediante una función glotona adaptativa aleatorizada; la segunda

aplica un procedimiento de búsqueda local a la solución construida en espera de encontrar una mejora [29].

A continuación, se detallan las dos fases de la técnica:

1. *Fase de construcción:* Se utiliza un algoritmo Glotón aleatorizado. Primero expliquemos qué es un algoritmo glotón. Los glotones tienen tres características: es incremental (vamos agregando valores a una cierta función a cada paso); no tiene marcha atrás (una vez tomada una decisión no se puede modificar) y siempre toma el siguiente mejor elemento de una lista de posibles soluciones. Como ejemplo de un algoritmo glotón tenemos el de "vecino más cercano" para resolver el problema del agente viajero; el cual consiste en elegir alguna ciudad al azar y visitar la ciudad que esté más cerca de la actual, de esa segunda ciudad tomamos la siguiente ciudad más cercana (sin considerar la primera, para así evitar un circuito). Continuamos este proceso hasta agotar las ciudades. Como vemos, este algoritmo glotón cumple con las tres características. Este algoritmo es extremadamente sencillo y con él podemos obtener n trayectorias para un agente viajero de n ciudades las cuales están listas para aplicarles el método de mejora.
   En un Algoritmo Glotón Aleatorizado, no necesariamente consideramos la mejor siguiente opción para incluirla al conjunto de solución, sino que se considera a todos aquellos candidatos $c(e)$, que se encuentran dentro de un porcentaje $\alpha \leq (0 \leq \alpha \leq 1)$ entre el mejor elemento, c_* y el peor de los candidatos posibles, c^*:

   $$c(e) \leq c_* + \alpha(c^* - c_*)$$

   El valor de α es una especie de "porcentaje de tolerancia" donde podemos aceptar no solo al mejor, sino también a los candidatos más cercanos a dicho valor mínimo. Valores usuales de α son 0.1 o 0.2, pero nada nos impide el uso de cualquier valor, incluyendo los extremos, $\alpha = 0$ (algoritmo glotón estándar) o $\alpha = 1$ (solución totalmente al azar) [26].

   La ventaja principal de utilizar un glotón aleatorizado respecto a tomar los extremos de α igual a cero o uno, es la diversidad que ofrece. Así el glotón aleatorizado es un equilibrio entre calidad y diversidad.

2. *Fase de mejora*. El método de mejora más utilizado es la búsqueda local, la cual consiste en explorar exhaustivamente la vecindad de una solución en busca de una mejor, si no existe dicha solución mejorada en el vecindario, nos encontramos en un óptimo local.
   La búsqueda local es muy importante dentro del GRASP ya que nos permite hacer búsquedas de mejora dentro de las regiones prometedoras del espacio de soluciones.

El algoritmo GRASP básico se describe como:

```
Comienza
1.  Para m = 1 hasta el TamañoSolución
2.      Genera una secuencia en la cual los vértices serán pintados con SeqCol(i)
3.      Para j = 1 hasta n
4.          Genera secuencia donde dos colores serán tratados con SeqCol(i)
5.          Para i = 1 hasta k
6.              Si Col(Sec(j)) = SeqCol(i) es una coloración válida, pintar el vértice
7.          Fin para i
8.          Si Col(j) = Φ, pinta el vértice de un color aleatorio
9.      Fin para j
10.     Mientras se tenga una mejora en la coloración
11.         Generar una secuencia para los vértices visitados Sec(j)
12.         Para j = 1 hasta n
13.             Genera secuencia SeqCol(i) para tratar los colores
14.             Para i = 1 hasta k
15.             Si Col(Seq(j)) = SeqCol(i), reduce número de vértices coloreados
16.             Fin Para i
17.         Fin Para j
18.     Fin Mientras
19. Fin Para m
Fin
```

Capítulo 4. Implementación de los algoritmos

Descripción de las herramientas utilizadas

Las implementaciones de los algoritmos fueron realizadas con el lenguaje de programación Python. Python es un lenguaje bajo licencia de código abierto y multiplataforma que soporta múltiples paradigmas de programación. Entre las características que se tomaron en cuenta a la hora de usarlo, para hacer el modelo propuesto se encuentran:

- *Interpretado*. Al ser un lenguaje interpretado, permite hacer pruebas y desarrollos más rápidos.
- *Legibilidad*. La sintaxis que usa Python favorece a que se tenga una mejor legibilidad y comprensión del código.
- *Librerías*. Posee una amplia gama de librerías para computación científica que facilitan tareas como el manejo y representación de datos.

Uso de la resiliencia

Un parámetro fundamental en la Coloración de Gráficas Suaves es la resiliencia porque como ya es sabido, ésta permite encontrar una cantidad adecuada de colores para clasificar (colorear); a continuación, se muestran ejemplos del uso de la resiliencia para resolver algunas instancias DIMACS de coloración mínima [12]:

Tabla 1. Resultados para la instancia DIMACS Myciel3 usando coloración de gráficas suaves.

Colores	Dureza	Solidez	Resiliencia
1	10.009	0.18198182	
2	2.0046	0.08099394	1.246857234
3	0.5029	0.03428864	1.362121682
4	0.002	0.00020779	**164.0158333**
5	0.014	0.00212121	-0.90204175
6	0.01	0.00218182	-0.02777956
7	0.008	0.00254545	-0.14285489
8	0.006	0.00290909	-0.12500128
9	0.004	0.00327273	-0.11111213
10	0.002	0.00363636	-0.09999835
11	0		

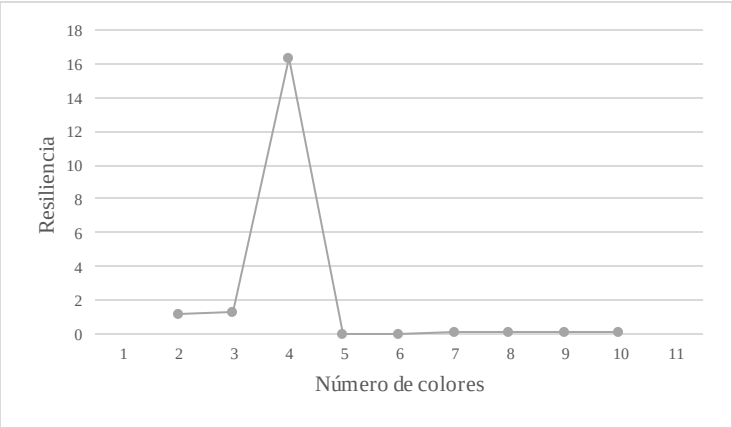

Figura 1. Representación gráfica de las resiliencias resultantes de la instancia Myciel3.

La razón principal de haber usado las instancias DIMACS Myciel3-6 es porque en éstas son conocidos los óptimos y son gráficas cuya dimensión es pequeña (11 – 95 vértices). Para el caso de la instancia Myciel3, su óptimo es en 4 colores, por ello, si se resuelve empleando el modelo CGS desde 1 hasta 11 colores, los datos obtenidos se recopilan en la Tabla 1; a pesar de que el aumento de la resiliencia se puede ver claramente, se comprueba mediante la graficación del parámetro y el resultado se muestra en la Figura 1. Si tomamos en cuenta que el óptimo conocido coincide con un aumento significativo en la resiliencia, esto quiere decir que el modelo funciona correctamente para la instancia.

La instancia Myciel4 posee 23 vértices y un número cromático de 5. Resolviéndolo usando CGS para 1 hasta 11 colores, los datos que nos proporcionará son los siguientes:

Tabla 2. Resultados para la instancia DIMACS Myciel4 usando coloración de gráficas suaves.

Colores	Dureza	Solidez	Resiliencia
1	35.5435	0.14048814	
2	9.0224	0.07471967	0.880202897
3	2.5149	0.03280304	1.277827604
4	0.5109	0.00935286	2.507273711
5	0.0084	0.0002029	**45.09590931**
6	0.0066	0.00020256	0.001678515
7	0.0054	0.00020543	-0.01397069
8	0.0044	0.00020406	0.006713712
9	0.0038	0.00021242	-0.03935599
10	0.0032	0.00021405	-0.00761504
11	0.0026		

De la Tabla 2, se pueden detectar al menos dos puntos en los que la resiliencia aumenta en proporción con respecto a los demás colores, esto es, para 4 y 5 colores. Para conseguir una mejor apreciación de los datos, las resiliencias fueron graficadas con los siguientes resultados:

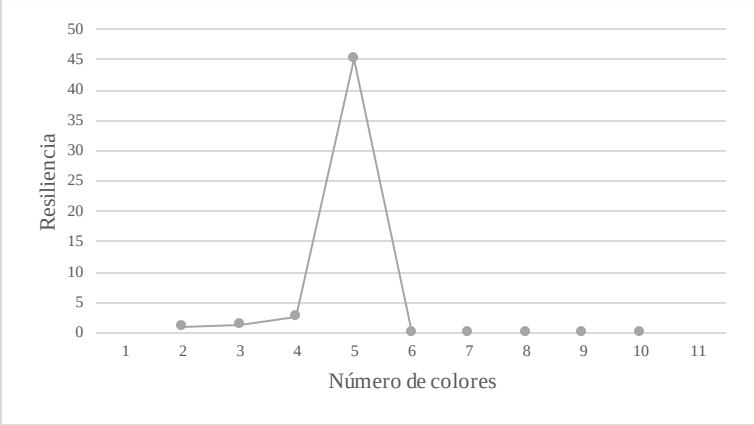

Figura 2. Representación gráfica de las resiliencias resultantes de la instancia Myciel4.

En la Figura 2, se puede distinguir que el aumento en la resiliencia para 4 colores es prácticamente insignificante cuando se le compara con el que sufre para 5 colores, por lo tanto, se puede llegar a la conclusión que, en este caso, el aumento más significativo de la resiliencia coincide con el óptimo de la gráfica.

La instancia Myciel5 posee 47 vértices y un número cromático de 6. Los datos obtenidos a partir de su solución con CGS desde 1 hasta 11 colores se detallan a continuación:

Tabla 3. Resultados para la instancia DIMACS Myciel5 usando coloración de gráficas suaves.

Colores	Dureza	Solidez	Resiliencia
1	118.1926	0.10933636	
2	31.5999	0.0597634	0.829486944
3	12.0666	0.03500948	0.707063344
4	4.0498	0.01603088	1.183877616
5	1.5395	0.00779889	1.055533544
6	0.0322	0.00020052	**37.89332735**
7	0.027	0.00020106	-0.00268576
8	0.023	0.00020076	0.001494322
9	0.02	0.00020157	-0.00401845
10	0.0176	0.00020242	-0.00419919
11	0.0156		

En la Tabla 3 se muestran tres colores a tomar en consideración: 4, 5 y 6. Sin embargo, y a pesar de que claramente el que presenta un mayor aumento está en 6, se elabora una gráfica para fines ilustrativos de comparación:

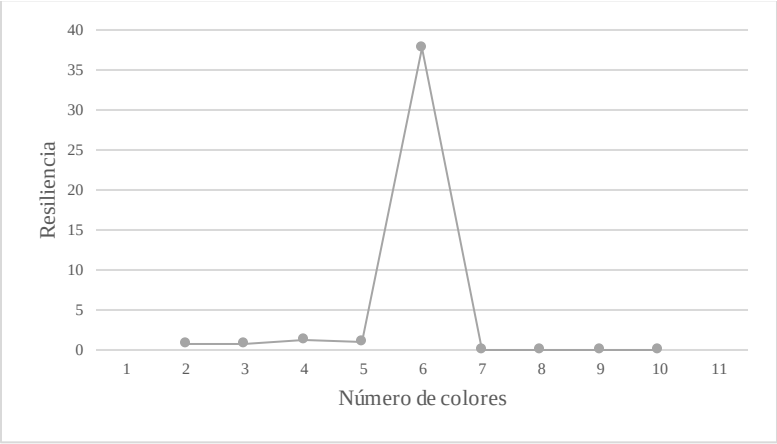

Figura 3. Representación gráfica de las resiliencias resultantes de la instancia Myciel5.

La Figura 3 comprueba que el aumento más significativo en la resiliencia dentro del modelo CGS coincide con el número cromático.

La instancia Myciel6 posee 95 vértices y un número cromático de 7. Su solución desde 1 hasta 11 colores empleando CGS arroja los siguientes resultados:

Tabla 4. Resultados para la instancia DIMACS Myciel6 usando coloración de gráficas suaves.

Colores	Dureza	Solidez	Resiliencia
1	377.58175	0.08456478	
2	105.0422	0.04755731	0.77816593
3	31.03004	0.02130209	1.23251864
4	10.02248	0.00927471	1.29679358
5	3.51783	0.00411442	1.25419485
6	1.01426	0.00143952	1.85819464
7	0.01202	2.01E-05	**70.5139401**
8	0.01038	2.01E-05	0.00173246
9	0.0091	2.00E-05	0.00226517
10	0.0081	2.01E-05	0.00064599
11	0.00728		

En la Tabla 4 podemos comprobar el comportamiento que la coloración de gráficas suaves ha tenido para resolver la instancia Myciel6, si graficamos la resiliencia:

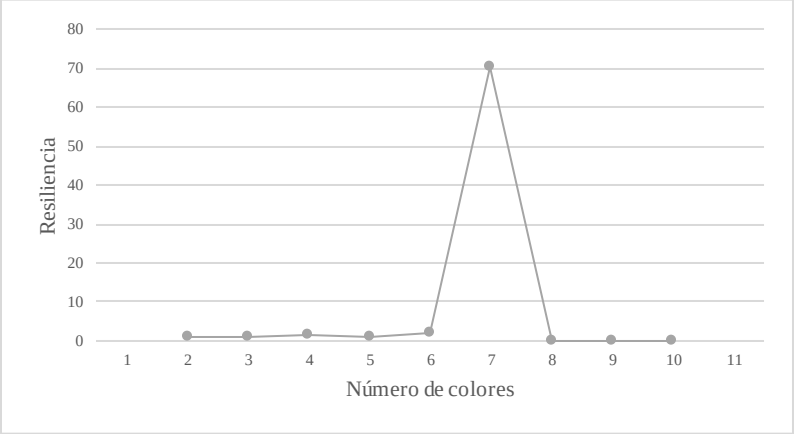

Figura 4. Representación gráfica de las resiliencias resultantes de la instancia Myciel6.

En esta gráfica nuevamente apreciamos el mismo comportamiento, que es un salto en la resiliencia en el color 7, mismo que corresponde con el número cromático de la instancia.

Otro parámetro importante a tomar en cuenta para las instancias DIMACS es la dureza, pues si es considerada además de la resiliencia, podemos observar en las Tablas 1-4 que ésta es menor a la penalización de 0.5 que se otorga cuando se convierte a gráfica completa ponderada; esto quiere decir que, todos los vértices originales de la instancia se pintaron y sólo quedan aquellos con una penalización de ε.

Con lo visto anteriormente, podemos comprobar que la resiliencia es un parámetro fundamental que permite al modelo de coloración de gráficas suaves encontrar una cantidad adecuada de colores para clasificar. A pesar de que en las instancias anteriormente vistas en todos los casos los aumentos más significativos de resiliencia coinciden con los números cromáticos correspondientes a cada gráfica; esto no quiere decir que siempre será así debido a que puede ocurrir el caso en que; por ejemplo, una instancia cuyo valor óptimo puede esperarse en cierto valor, el modelo de CGS proporcione una coloración ya sea con una menor o mayor cantidad de colores, lo cual no quiere decir que el modelo esté funcionando incorrectamente sino que simplemente encontró una solución con un número distinto de colores, lo cual puede ser un dato adicional a considerarse. Además, puede existir un caso en donde se aprecie más de un solo aumento significativo en la resiliencia con lo cual nuevamente no quiere decir que existe un error, sino que existe más de un color adecuado para poder clasificar la instancia.

4.1 Técnica metaheurística implementada: GRASP

Como los problemas de coloración están en la clase NP-Dura a partir de 20 instancias, es necesario el uso de metaheurísticas para encontrar soluciones factibles después de ese punto. La técnica metaheuristica escogida para implementarse en este trabajo es GRASP (Algoritmo Glotón Mejorado) debido a que en la literatura [7] [13] ésta es la que mejor desempeño muestra con respecto a esfuerzo computacional/calidad de las soluciones al implementarse para resolver problemas de coloración. Esta técnica puede ser fácilmente adaptada para el problema de coloración de gráficas suaves empleando el siguiente método descrito a continuación.

El grafo se pinta con una secuencia aleatoria, intentando que los colores también sean aleatorios, esperando que el color colocado genere una coloración válida. Si esto no es posible, es pintada de cualquier color. Debido al carácter aleatorio de este proceso, el número de formas en que los vértices pueden ser pintados es $n!$; en cada uno de ellos, los colores con que pueden ser pintados es de $k!$ formas distintas con lo cual se genera una gran diversidad de soluciones [7].

```
Comienza
1.  Para m = 1 hasta el TamañoSolución
2.      Genera una secuencia en la cual los vértices serán pintados con SeqCol(i)
3.      Para j = 1 hasta n
4.          Genera secuencia donde dos colores serán tratados con SeqCol(i)
5.          Para i = 1 hasta k
6.              Si Col(Sec(j)) = SeqCol(i) es una coloración válida, pintar el vértice
7.          Fin para i
8.          Si Col(j) = Φ, pinta el vértice de un color aleatorio
9.      Fin para j
10.     Mientras se tenga una mejora en la coloración
11.         Generar una secuencia para los vértices visitados Sec(j)
12.         Para j = 1 hasta n
13.             Genera secuencia SeqCol(i) para tratar los colores
14.             Para i = 1 hasta k
15.             Si Col(Seq(j)) = SeqCol(i), reduce número de vértices coloreados
16.             Fin Para i
17.         Fin Para j
18.     Fin Mientras
19. Fin Para m
Fin
```

Durante la fase de mejora (en las líneas 11 a 14), un vértice y un color son seleccionados de manera aleatoria. En la línea 15, la función objetivo se plantea de la siguiente forma:

$$\min f(S) = C_{op}^{k} = \sum_{\{i,j\} \in E, C^k(i) = C^k(j)} p_{ij}$$

La cual corresponde a la suma de las penalizaciones cuyos extremos que están coloreados igualmente.

El proceso finaliza una vez que todos los colores han sido probados en todos los vértices y no es encontrada una mejor coloración. El ciclo de 2 a 18 se ejecuta n veces; es decir, el tamaño de la solución = n y al finalizar este ciclo, la solución factible con menor rigidez será escogida.

Sin embargo, debido a que la exploración de todas las posibles soluciones puede demorar bastante en términos computacionales, se propusieron modificaciones al modelo tradicional como el uso de retroalimentación con la mejor solución existente y una búsqueda por vecindades variables la cual usa el mutado aleatorio de un vértice por un color aleatorio. El pseucódigo básico del algoritmo modificado se describe como sigue:

```
Comienza
1. Para m = 1 hasta el TamañoSolución:
2.      Genera Glotón y obtener dureza
3.      Almacenar Glotón en coloracionGloton
4.      Si dureza(coloracionGloton) es mejor que la dureza(mejorColoracion):
5.          mejorColoracion = coloracionGloton
6.      Para maxIter= 1 hasta 3500:
7.          Mutar vértice aleatorio de la mejorColoracion por color aleatorio
8.              Si mutación reduce dureza:
9.                  actualiza mejorColoracion
10.     Fin maxIter
11. Fin Para m
12. Regresar dureza(mejorColoracion), mejorColoracion
Fin
```

La fase que corresponde de las líneas 2 a 5 corresponden a la fase inicial y consiste en la generación de una coloración usando un glotón aleatorizado, misma que será almacenada y posteriormente comparada con la mejor solución encontrada hasta el momento; en la fase de mejora correspondiente a las líneas 6 a 12, se usa una búsqueda local basada en mutaciones de un vértice aleatorio por un color aleatorio. Cabe destacar que, TamañoSolución = 50, por lo que el procedimiento anteriormente descrito de las líneas 2 a

11 será repetido 50 veces y como resultado se regresará la menor dureza y su coloración correspondiente.

4.2 Puntos de mejora del modelo implementado

Con la finalidad de obtener las mejores soluciones posibles durante la fase de pruebas para cada uno de los problemas de coloración clásicos, se elaboraron distintas versiones de un GRASP, mismos que fueron comparados usando las instancias DIMACS, pertenecientes a los problemas de coloración mínima y débil. La particularidad de las instancias que se usaron para hacer las comparaciones es que en todas ellas se conoce el óptimo; por ello, la decisión de si el algoritmo encuentra o no la solución es binaria. Los resultados obtenidos se muestran a continuación:

Tabla 5. Comparación de eficacia en 20 corridas de una búsqueda local contra las primeras versiones de un GRASP.

	Myciel6	**Myciel7**	**Queen7**	**Huck**	**Jean**
Tamaño	95	191	49	74	80
Número cromático	7	8	7	11	10
Búsqueda local con 5000 iteraciones, elemento mutado aleatorio	75%	0%	0%	0%	0%
GRASP tamaño de solución a 50 y búsqueda local con 3500 iteraciones, mutado aleatorio	90%	45%	15%	90%	90%
GRASP tamaño de solución a 50 y búsqueda local con 3500 iteraciones, generando secuencia para visitar todos los vértices	60%	0%	0%	85%	85%

En la Tabla 5 se comparan las dos primeras versiones de un GRASP contra una búsqueda local, estos resultados se obtuvieron corriendo de manera aleatoria los tres algoritmos 20 veces por cada instancia (también escogida aleatoriamente) a resolver; en la Tabla 5 se muestra el porcentaje de veces donde los algoritmos encontraron el óptimo de las instancias.

Es interesante notar que a pesar de que GRASP es una técnica superior en cuanto a búsqueda local, la forma en cómo se hacen las fases de mejora y construcción afectan mucho al desempeño que tiene porque en el caso del GRASP donde se genera una secuencia para visitar los vértices, los resultados fueron peores que los encontrados por una búsqueda local. Tomando en cuenta esto, las siguientes versiones del GRASP se enfocaron

en probar distintos métodos de mejora para las distintas fases, los resultados se ven a continuación:

Tabla 6. Comparación de eficacia en 20 corridas de versiones mejoradas de un GRASP.

	Myciel6	**Myciel7**	**Queen7**	**Huck**	**Jean**
Tamaño	95	191	49	74	80
Óptimo	7	8	7	11	10
GRASP tamaño de solución a 50, elitismo entre 500 glotones y b. local con 3500 iteraciones, mutado aleatorio	100%	85%	70%	100%	100%
GRASP tamaño de solución a 50, elitismo entre 500 glotones y b. local con máx. iter. en 500, mutado aleatorio	100%	65%	55%	100%	100%
GRASP tamaño de solución a 50, elitismo entre 500 glotones, b. loc. 3500, 2 tipos mut., retroalimentación con coloración de menor dureza	100%	100%	90%	100%	100%

En la Tabla 6 se ve que haciendo un elitismo entre más de 100 glotones se mejora la eficacia del GRASP, asimismo la fase de mejora con mutado aleatorio y una cantidad definida de iteraciones se pone por encima de las demás. Para comprobar matemáticamente si realmente existen diferencias entre usar una versión u otra se hizo un análisis de varianza comparando a las versiones descritas en las Tablas 5 y 6.

ANOVA unidireccional: Eficacia vs. Programas

Tabla 7. Pruebas de hipótesis para el análisis de varianza.

Método	
Hipótesis nula	Todas las medias son iguales
Hipótesis alterna	Por lo menos una media es diferente
Nivel de significancia	$\alpha = 0.05$

Tomando en consideración las pruebas de hipótesis de la Tabla 7, se propuso igualdad de varianzas para el análisis con los siguientes datos:

Tabla 8. Información de los factores (programas) usados en el análisis ANOVA.

Información del factor		
Factor	Niveles	Valores
Programas	5	1, 2, 3, 4, 5, 6

La Tabla ANOVA para los experimentos tomando en cuenta la información de los programas de la Tabla 8, y usando una cantidad de muestras de igual tamaño para cada tratamiento queda como sigue:

Tabla 9. Análisis estático descriptivo de las Tablas 5 y 6.

Análisis de varianza							
Fuente	GL	SC. Sec	Contribución	SC Ajust	MC Ajust	Valor F	Valor p
Programas	5	24857	56.03%	24857	4971.3	6.12	0.001
Error	24	19510	43.97%	19510	812.9		
Total	29	44367	100.00%				

Si con la información descrita de las Tablas 7 – 9, se hacen comparaciones utilizando el método de Tukey con una confianza de 95% los resultados obtenidos son los siguientes:

Tabla 10. Comparaciones en parejas de Tukey de las Tablas 5 y 6.

Programas	**N**	**Media**	**Desv. Est.**	**IC de 95%**
1	5	15.0	33.5	(-11.3, 41.3)
2	5	66.0	34.5	(39.7, 92.3)
3	5	46.0	43.2	(19.7, 72.3)
4	5	91.00	13.42	(64.68, 117.32)
5	5	84.00	22.19	(57.68, 110.32)
6	5	98.00	4.47	(71.68, 124.32)

Mientras que la agrupación de la información de la Tabla 10, se muestra a continuación:

Tabla 11. Agrupación de información usando el método de Tukey.

Programas	**N**	**Media**	**Desv. Est.**
6	5	98.00	A
5	5	91.00	A

4	5	84.00	A
3	5	66.0	A B
2	5	46.0	A B
1	5	15.0	B

De la Tabla 11, si las medias no comparten una letra, esto significa que son significativamente diferentes. Estos datos también pueden ser representados gráficamente para una mejor comprensión:

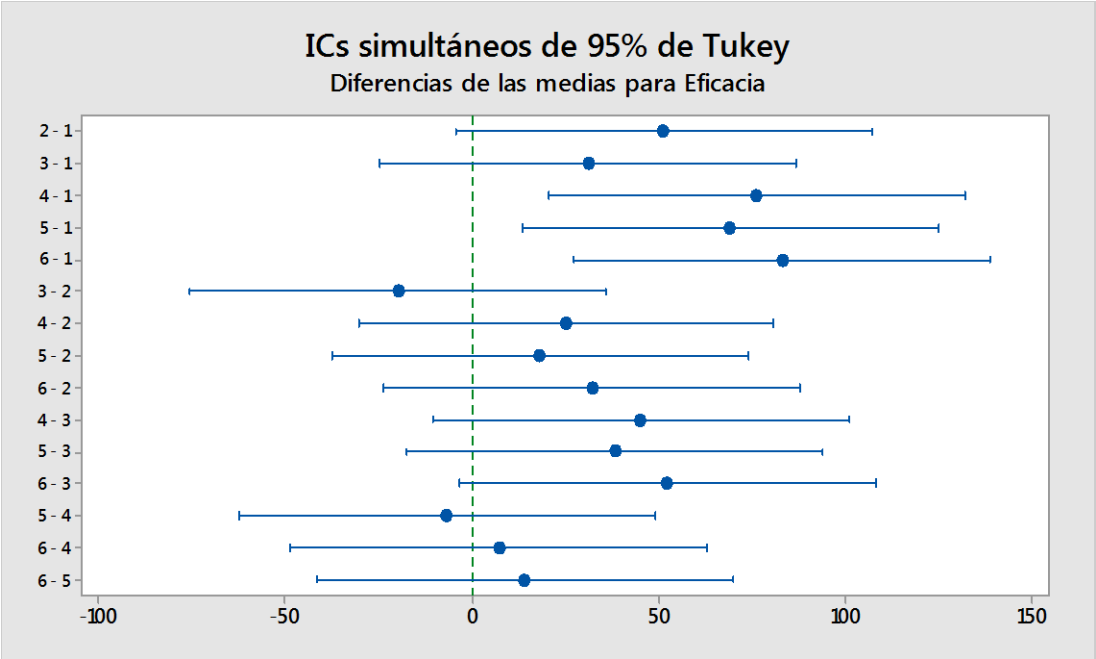

Figura 5. Representación gráfica de los intervalos de confianza del análisis ANOVA usando el método de Tukey.

La Figura 5 muestra de una forma más dinámica entre qué parejas de programas existe una diferencia significativa, donde, si en alguno de los intervalos no contiene el cero, significa que las medias correspondientes son significativamente diferentes. Por lo que a partir de este momento, podemos hacer válida la hipótesis alterna donde, por lo menos una media es diferente.

Asimismo, y con la finalidad de corroborar y ampliar la información del ANOVA, se hizo un diagrama de cajas con las medias obtenidas:

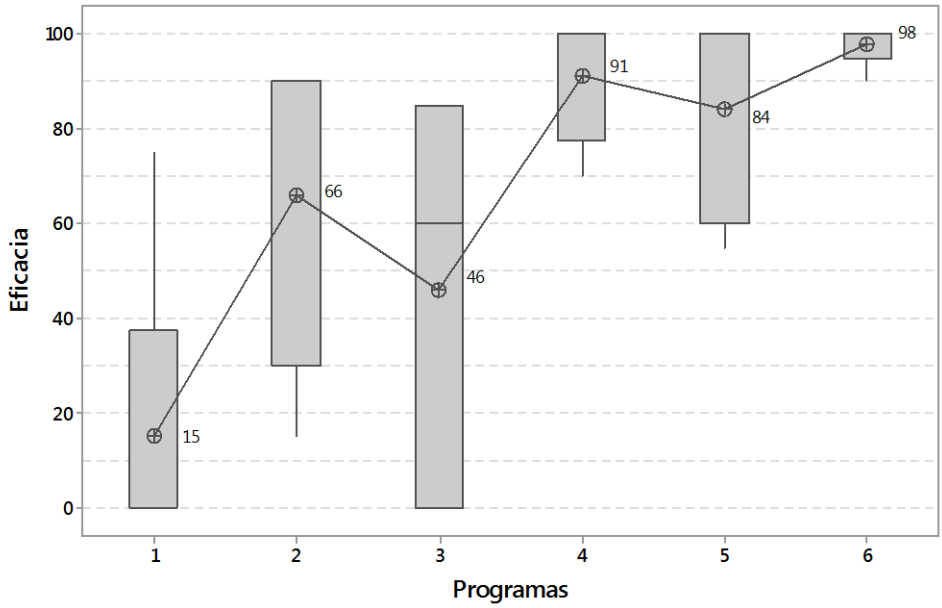

Figura 6. Diagrama de caja de las medias del análisis ANOVA.

El diagrama de caja de la Figura 6 muestra una tendencia a que el algoritmo 6 genere soluciones con mejor eficacia al resolver instancias.

Con este análisis es posible llegar a la conclusión de que el GRASP más eficaz es el 6 puesto que es aquel cuyo valor de media es más grande; por ello, para pruebas posteriores esta versión es la que se usará.

Cabe destacar que durante las pruebas para solucionar las instancias DIMACS si bien son conocidos los óptimos a los que se deben llegar, mismos que coinciden con un aumento significativo en la resiliencia en el modelo de coloración de gráficas suaves, el resolverlo por este último aporta además un dato adicional importante y es la solución del problema de coloración débil a estas mismas instancias.

A continuación, se describen las mejoras realizadas al GRASP que resultó ser más eficiente en el ANOVA, esto con la finalidad de obtener mejores resultados.

Mejora 1: Elitismo entre glotones: Como se ha comentado en apartados anteriores, en la fase de construcción se hizo un agregado importante al generar 500 soluciones iniciales con un glotón aleatorizado de las cuales únicamente guardaremos aquella cuya coloración sea la menor. Esta mejora, además de que nos da una notablemente mejor solución inicial, también supuso una ventaja en algunas instancias DIMACS, debido a la variedad de soluciones encontradas por los glotones y las propiedades de algunas instancias como su baja densidad, por ello, éstas pueden ser resueltas en esta fase sin necesidad de pasar por la fase de mejora; tal es el caso de las instancias Mycielski-3-5, Huckleberry Finn y Les Miserables.

Mejora 2: Dos tipos de mutación sin repeticiones: En el caso del método de mejora usando búsqueda por vecindades variables, originalmente se contaba con un tipo mutación la cual consiste en mutar un vértice aleatorio por un color aleatorio; sin embargo, a pesar de haber probado otros métodos para cambiar coloraciones como el del *intercambio*, que consiste en cambiar un vértice aleatorio por otro vértice también aleatorio pero distinto y la combinación entre ellos, fue descubierto que sólo dejaban a la solución en un óptimo local; por ello, se optó por usar dos tipos de mutación, las cuales se usan dependiendo de una probabilidad. El tipo de mutación se selecciona de acuerdo con una probabilidad de selección generada entre 0 y 1; así, la mutación 1 se escoge si la probabilidad es menor a .70, de lo contrario se usa la mutación 2 que consiste en la mutación de dos vértices aleatorios sin repeticiones por colores aleatorios distintos a los originales.

De esta forma, el pseudocódigo del algoritmo GRASP implementado para pruebas posteriores queda de la siguiente forma:

```
Comienza
1.  Para m = 1 hasta el TamañoSolución:
2.      Para i = 1 hasta 500:
3.          Genera Glotón y obtener dureza
4.          Almacenar la mejor coloración generada en coloracionGloton
5.      Fin para i
6.      Si dureza(coloracionGloton) es mejor que la dureza(mejorColoracion):
7.          mejorColoracion = coloracionGloton
8.      Para maxIter= 1 hasta 3500:
9.          Obtener la probabilidad p U(0,1)
10.         Si p es menor a .70:
11.             Mutar vértice aleatorio de mejorColoracion por un color aleatorio
12.         En caso contrario:
13.             Mutar dos vértices de mejorColoración por colores aleatorios
14.             Si mutación reduce dureza:
15.                 actualiza mejorColoracion
16.     Fin maxIter
17. Fin Para m
18. Regresar dureza(mejorColoracion), mejorColoracion
Fin
```

4.3 Descripción de las funciones para el modelo propuesto

Algoritmo Coloración de Gráficas Suaves

```
1.  Función coincidencias (color, coloraciónEvaluada)
2.      Para cada elemento en coloraciónEvaluada:
3.          Si el elemento = color
4.              Guarda posición en coincidencia
5.      Fin para cada elemento
6.      Regresa coincidencia
```

```
1.  Función penalizaciones (coincidencia, matriz)
2.      Establece una variable llamada suma = 0
2.      Para cada elemento en coincidencia
3.          Recupera el valor que le corresponde en la matriz
4.              suma += valor recuperado
5.      Fin para cada elemento
6.      Regresa suma
```

```
1.  Función dureza (coloraciónEvaluada, matriz)
2.      Establece una variable llamada dureza = 0
3.      Para cada color en la lista de colores
4.          variableAuxiliar = coincidencias (color, coloraciónEvaluada)
5.          dureza += penalizaciones (variableAuxiliar, matriz)
6.      Fin para cada color
7.      regresa dureza
```

Comentarios del algoritmo de Coloración de Gráficas Suaves

La función que se puede considerar principal es la de *dureza*, esta función recibe una coloración a evaluar en forma de lista y el grafo en forma de matriz; ésta extrae todos los colores de la lista de uno a uno y, para cada color, llama a la función *coincidencias* quien se encarga de encontrar conflictos entre el color que se evalúa con sus vecinos; en caso de encontrar un conflicto (un vecino que esté pintado con el mismo color) guarda la posición de este; al finalizar la evaluación de la función *coincidencias*, ésta regresa una lista con la posición de las penalizaciones que tiene el color que se evaluó. Posteriormente, la función *dureza* llama a la función *penalizaciones* y le pasa como parámetros la lista de posiciones en conjunto con la matriz, esta función se encarga de recuperar los valores de penalización

correspondientes a cada color almacenándolos en un acumulador llamado *dureza*. El ciclo termina cuando la función *dureza* ejecuta a las funciones de *coincidencias y penalizaciones* tantas veces como colores disponibles existan.

Algoritmo glotón

```
1.  Función pruebaColor (vértice, posibleColor)
2.      Recupera los vecinos del vértice recibido:
3.          Recupera los colores de sus vecinos
4.          Si el posibleColor ya lo tiene un vecino, entonces
5.              Regresa Falso
6.          En otro caso, se puede pintar de ese color, entonces
7.              Regresa Verdadero
```

```
1.  Función coloreado (vértice)
2.      Genera una lista ordenada aleatoriamente de posiblesColores
3.      Toma un color de la lista de posiblesColores
4.          Si la pruebaColor (vértice, color) regresa Verdadero,
5.              regresa color
6.      Fin para toma un color
7.      Si se terminaron los colores y no es posible pintar con ninguno,
8.          regresa colorAleatorio
```

```
1.  Función glotón (grafo)
2.      Para cada vértice tomado aleatoriamente del grafo
3.          colorVértice = coloreado (vértice)
4.          Guarda en un diccionario la entrada del vértice y su color asignado
5.      Fin para cada vértice
```

Comentarios del algoritmo glotón

La función principal es la del *glotón*, la cual toma el grafo y para cada uno de los vértices tomado aleatoriamente y sin repeticiones, guarda en un diccionario la entrada de color que le envió la función *coloreado*. Esta segunda función toma el vértice y de una lista generada en orden aleatorio con todos los posibles colores, prueba cada uno de ellos mediante la función *pruebaColor* la cual recibe el vértice y un posible color que se le puede asignar. Esta función recupera los nodos vecinos y sus colores asignados, si el color que recibió no lo tiene ningún vecino regresa que se puede pintar de ese color y es guardado en un diccionario, en caso contrario recibe otro color hasta terminar con la lista. En caso de que

no poderse pintar con ningún color disponible, la función *coloreado* lo pinta aleatoriamente y ese color es el que se guardará en el diccionario.

4.4 Descripción de las funciones para cargar instancias

Función para crear el grafo

```
1.  Función grafo (puntos, puntoMáximo)
2.      Para i = 1 hasta puntoMáximo
3.          Para un punto en puntos
4.              Si el punto en x es igual a i
5.                  inserta en vecinos el punto y
6.          Fin para cada elemento
7.          inserta en grafo el vértice i y sus vecinos
8.      Fin para i
9.      regresa el grafo
```

Comentarios de la función para crear el grafo

La función *grafo* es usada por todas las demás funciones para cargar instancias, esta se encarga de recibir una lista con los puntos extremos de un grafo y el punto máximo del mismo. Esta función toma a todos los vértices y por cada uno, busca a todos sus vecinos. Una vez que hace esto, el resultado lo guarda en forma de diccionario. Cuando termina con todos los vértices y vecinos, regresa un grafo completo.

Función para cargar instancias DIMACS

```
1.  Función cargarDIMACS ()
2.      Lee el archivo de entrada
3.      Lee una línea por una a una aquellas que inicien con la letra "e"
4.          Divide a la línea en elementos separados por un espacio
5.              Guarda a los puntos en una lista puntos
6.      Fin para lee una línea
7.      De la lista puntos obtén el puntoMáximo y el puntoMínimo
8.      Crea el grafo llamando a la función grafo (puntos, puntoMáximo)
```

```
9.      Para cada vértice en grafo
10.         Si el vértice existe en puntos
11.             inserta en auxiliar una penalización de 0.5
12.         En caso contrario
13.             inserta en auxiliar una penalización de 0.000001
14.         Si se ha llegado al límite de la matriz en x
15.             inserta en matriz el auxiliar
16.             vacía lo que tenga auxiliar
17.     Fin para cada vértice
18.     regresa la matriz, longitud de la matriz y grafo
```

Comentarios de la función para cargar instancias DIMACS

Esta función se encarga de tomar una instancia DIMACS con el formato estándar y lee sólo aquellas líneas que contengan información correspondiente al grafo; es decir, los puntos extremos. Estos puntos los guarda en una lista de la cual saca los valores máximos y mínimos. Con los datos obtenidos anteriormente, crea el grafo llamando a la función *grafo*. Con el grafo completo creado, se procede a crear la matriz de distancias tomando cada uno de los vértices comparándolo con la lista de *puntos*. Si el vértice existe en la lista de *puntos* se almacena en una lista auxiliar una penalización de 0.5, en caso contrario ésta será de 0.000001. Cuando se llega al límite de la matriz se inserta la lista auxiliar en la matriz. Una vez que se han tomado todos los vértices, la matriz está creada. Finalmente, la función regresará la matriz, longitud de la matriz y el grafo completo.

Función para cargar instancias robustas

```
1.  Función cargarRobusta ()
2.      Lee el archivo de entrada
3.      Extrae el tamaño n de la matriz por el nombre del archivo
4.      Lee una línea a una
5.          Divide a la línea en elementos separados por un espacio
6.              Por cada elemento de la línea
7.                  Si elemento != 0
8.                      Agrega coordenadas del elemento a puntos
9.                  Si elemento = 1
10.                     inserta en auxiliar una penalización de n^2
11.                 En caso contrario
12.                     inserta en auxiliar la penalización original
13.                 Si se ha llegado al límite de la matriz en x
```

```
14.                                   inserta en matriz el auxiliar
15.                                   vacía lo que tenga auxiliar
16.                           Fin para cada elemento
17.          Fin para lee una línea a una
18.          Crea el grafo llamando a la función grafo (puntos, n)
19.          regresa la matriz, longitud de la matriz y grafo
```

Comentarios de la función para cargar instancias robustas

Esta función se encarga de leer las instancias robustas y extrae mediante el nombre del archivo el tamaño n de la matriz. El archivo lo lee línea a línea y separa los datos tomando como referencia los espacios en blanco entre ellos, debido a que el formato de entrada es una matriz cuadrada, esta únicamente se re-penaliza como se ha visto en el Capítulo 2. Cuando en la entrada se lee un valor distinto a cero, el vértice es agregado a la lista *puntos*. Si el valor leído es exactamente 1, entonces es insertada una penalización de n^2 en una lista auxiliar; en caso contrario, se inserta la penalización original. Cuando la lista auxiliar tiene un número de elementos igual a n, ésta se inserta en una matriz y su contenido es borrado. El ciclo termina cuando se han leído todas las líneas de la instancia. Después, se llama a la función *grafo* pasando la lista *puntos* y la dimensión n. Finalmente regresa la matriz, la longitud de la matriz y el grafo.

Función para cargar instancias generadas

```
1.   Función cargarArchivo ()
2.          Lee el archivo de entrada
3.          Lee una línea a una
4.                 Divide a la línea en elementos separados por un espacio
5.                        Guarda a los puntos en una lista puntos
6.          Fin para cada lee una línea a una
7.          De la lista puntos obtén el puntoMáximo y el puntoMínimo
8.          Crea el grafo llamando a la función grafo (puntos, puntoMáximo)
9.          Para cada punto x, y en puntos
10.                crea una lista vacía llamada auxiliar
11.                Para cada punto x2, y2 en puntos
12.                       d = ( ((x2-x)**2)  + ((y2-y)**2) )**.5
13.                       inserta en auxiliar la distancia d
14.                Fin para cada punto x2, y2
15.                inserta en matriz el auxiliar
16.         Fin para cada punto x, y
```

15. regresa la matriz, longitud de la matriz y grafo

Comentarios de la función para cargar instancias generadas

Esta función lee un archivo línea por línea y separa los vértices tomando como referencia los espacios en blanco entre ellos, estos puntos los guarda en una lista de *puntos* de la cual se obtienen los valores máximos y mínimos. Con los datos obtenidos anteriormente, crea el grafo llamando a la función *grafo*. Con el grafo completo creado, se procede a crear la matriz de distancias usando la distancia euclidiana cuadrada; esto es, toma la coordenada de un punto y procede a sacar las distancias de ese punto a todos los demás, guardando el resultado en una variable auxiliar del tipo lista y posteriormente insertando la lista en una matriz cuando se han sacado todas las distancias de un solo punto hacia los otros. El proceso termina una vez se han obtenido las distancias del último punto o punto máximo hacia todos los demás. Finalmente, la función regresa la matriz, la longitud de la matriz y el grafo.

Capítulo 5. Instancias

Para hacer una validación del modelo propuesto, es necesario utilizar instancias de prueba o, dicho de otro modo, pruebas de las cuales es conocido el óptimo y así verificar si este es encontrado o, en su defecto, qué tanto se acerca a él; sin embargo, aunque en la literatura existen ejemplos muy extendidos sobre instancias para la coloración de grafos en general como es el caso de DIMACS, estas no abarcan por completo a los problemas de coloración clásicos mencionados, por lo cual se hace evidente la importancia de implementar instancias exclusivas de cada problema a resolver y encontrar una forma evaluar la calidad de los resultados que arroje el nuevo modelo.

Instancias para coloración mínima y débil

Para el caso de la coloración mínima y débil se emplearon los repositorios DIMACS [12] de las cuales se seleccionaron aquellas instancias que una heurística de búsqueda local no pudo resolver o cuya solución fue sumamente complicada.

El reto de implementación DIMACS a las cuales pertenecen estas instancias, establece preguntas del desempeño de un algoritmo realista determinístico, donde el peor caso de análisis es sumamente pesimista y los modelos probabilísticos son demasiado irreales. La experimentación también trae preguntas algorítmicas cercanas a los problemas originales que motivan el trabajo teórico. Esto también prueba muchas suposiciones sobre los métodos de implementación y estructura de datos.

Asimismo, provee una oportunidad para desarrollar y probar instancias de problemas, generadores de instancias, otros métodos de prueba y comparación del desempeño de los algoritmos. Además, es un paso en la transferencia de la tecnología, brindando implementaciones de algoritmos de punta para que otros adapten.

Muchos artículos de investigación han implementado el formato de grafos DIMACS. El primer reto usó redes (grafos dirigidos con capacidades de aristas y vértices) y grafos no dirigidos (para coincidencias). Este formato fue propuesto por David Johnson de los laboratorios Bell AT&T en 1993 en un artículo llamado “Flujo de redes y coincidencias”.

El segundo reto usa grafos no dirigidos y está descrito en un artículo llamado “Problemas NP-Duros: clan máximo, coloración de grafos y satisfatibilidad” publicado por Michael Trick en 1996.

El propósito del reto DIMACS es minimizar el esfuerzo que requiere probar y comparar algoritmos y heurísticas proveyendo un banco de pruebas común de instancias y herramientas de análisis. Para facilitar este esfuerzo, un formato estándar debe de ser escogido para dirigir los problemas. A continuación, se describirá el formato que las instancias poseen. Éste es un formato flexible y apropiado para muchos tipos de grafos y problemas de redes. Un archivo de entrada contiene toda la información necesaria sobre las

necesidades de la gráfica para definir cualquier problema de clan o un problema de coloración. Cierta información que está incluida puede no ser relevante para un problema; por ejemplo, el peso de los nodos no es necesario para problemas de coloración y, por lo tanto, la información puede ser ignorada.

En este formato, los vértices están numerados del 1 hasta el *n* y también existen *m* aristas en la gráfica. Para usarlos, se asume que los archivos están bien estructurados y son consistentes internamente: los valores identificadores del nodo son válidos, los nodos definidos son únicos, existen exactamente *m* aristas definidas y así.

- *Comentarios*. Una línea de comentario brinda información legible para los humanos sobre el archivo y la cual es ignorada por los programas. Las líneas de comentarios pueden aparecer en cualquier lugar dentro del archivo. Cada comentario comienza con una letra minúscula **c**

  c Este es un ejemplo de una línea de comentario

- *Línea de problema*. Hay una línea de problema por archivo de entrada. La línea del problema debe de aparecer justo antes de cualquier línea descriptora de nodos. Para instancias de redes, la línea del problema tiene el siguiente formato:

  p FORMATO NODOS ARISTAS

  La letra minúscula **p** significa que esta es una línea del problema. El campo FORMATO es para la consistencia con el reto anterior y debe de contener la palabra "edge". El campo NODOS debe de contener un valor entero especificando a *n*, el número de vértices de la gráfica. El campo ARISTAS contiene un valor entero especificando *m*, el número de aristas en la gráfica.
- *Descriptores de nodo*. Para este reto, un descriptor de nodo es requerido solamente para problemas de cliqué ponderados. Estas líneas darán el peso asignado a un nodo en un cliqué. Existe una línea de descriptor de nodo para cada nodo con el siguiente formato. En caso de que existan nodos sin un descriptor se tomará por definición el valor de 1.

  n ID VALOR

  La letra minúscula **n** significa que es una línea descriptora de nodo. El campo ID da el número de identificación del nodo, un entero entre 1 y *n*. El VALOR proporciona el valor objetivo a poseer por el nodo en el cliqué. Este valor debe de ser entero y puede ser positivo, negativo o cero.
- *Descriptores de aristas*. Hay una línea descriptora de arista por cada arista en la gráfica, la cual tiene el siguiente formato. Cada arista (*v*, *w*) aparece exactamente una vez en el archivo de entrada y no es repetida como (*w*, *v*).

e W V

La letra minúscula **e** significa que ésta es una línea descriptora de arista. Para una arista (*w*, *v*), el campo W y V especifican los puntos extremos.

- *Descriptores opcionales*. Según la información requerida, pueden existir piezas adicionales de información acerca de la gráfica. Esta información por lo general define parámetros usados para generar la gráfica o bien, definen información específica del generador. La siguiente lista puede ser agregada de acuerdo con el interés de los generadores del problema:
    - Descriptores geométricos. Un método común para generar o mostrar gráficas es tener los nodos embebidos en algún espacio y que las aristas estén incluidas de acuerdo con una función de distancia entre nodos usando alguna métrica. La información del nodo puede ser definida por un descriptor de dimensión y un vértice embebido.

      d DIM MÉTRICA

      DIM es un entero que da el número de dimensiones en el espacio, mientras la MÉTRICA es una cadena de texto que representa la métrica para el espacio, y puede tomar un número de formas.
    - Descriptores de parámetro. Los descriptores de parámetro son usados para dar información adicional sobre cómo fue generada la gráfica. Estas se encuentran incluidas únicamente para ayudar a aquellos códigos específicamente diseñados para atacar problemas especialmente estructurados. La forma general del descriptor del parámetro es:

      x PARÁMETRO VALOR

      La letra minúscula **x** significa que este es un parámetro de descriptor de línea. El campo PARÁMETRO es una cadena de texto que da el nombre del parámetro, mientras que el campo VALOR es un valor numérico que le da el valor correspondiente.

A continuación, se describen los tipos de instancias DIMACS usadas en este trabajo:

- REG.- Son problemas basados en asignación de registros para variables en códigos reales.
- MYC.- Grafos basados en la tranformación de Mycielski. Estos grafos son difíciles de resolver porque son libres de triángulos (clique número 2) pero el número de coloración incrementa en el tamaño del problema.

- SGB.- Grafos de Donald Knuth's Stanford GraphBase. Estos pueden ser divididos en:
    - *Grafos de libros*: Dado un trabajo de literatura, creamos un grafo donde cada vértice representa una letra o símbolo. Dos vértices están conectados por una arista si los caracteres se encuentran uno al lado del otro en el texto.
    - *Grafos de juegos*: En este grafo, cada vértice representa un equipo en la liga de fútbol americano colegial de EUA. Si dos equipos compitieron entre sí durante la temporada, son unidos por una arista. Este grafo está basado en datos reales de la liga de 1990.
    - *Grafos de distancias*: Los vértices en este grafo representan un conjunto de ciudades de Estados Unidos y dos vértices están unidos por una arista si la distancia entre ambas ciudades es menor o igual que una cierta cota. Aunque este ejemplo no tiene una aplicación directa, está basado en datos reales de distancias y se pueden generar conglomerados con distintos umbrales de distancia α.
    - *Grafos de reinas*: Dado por un tablero de nxn, un grafo de reinas es una gráfica de n^2 vértices, donde cada vértice corresponde a un cuadro del tablero. Dos vértices están conectados por una arista si los cuadrados correspondientes están en la misma fila, columna o diagonal. Este problema tiene una interpretación natural: dado un tablero de ajedrez, ¿Es posible colocar n conjuntos de n reinas en el tablero tal que, no existan dos reinas en la misma fila, columna o diagonal? La respuesta es afirmativa, si y sólo si la gráfica tiene una coloración con una cantidad de colores igual a n. En todos los casos, el clan máximo en la gráfica no es más de n y el valor de coloración no es menor que n.

Instancias para coloración robusta

Para la coloración robusta se retomaron comparativas de trabajos existentes en la literatura [26] [31], en los cuales instancias exclusivas del problema son resueltas con algoritmos pensados especialmente para la solución del problema de coloración robusta. Asimismo, dado que las instancias están divididas en dos tamaños, aquellas que pueden ser resueltas por el modelo binario, o menores a 20 vértices, y aquellas en las que son necesarias técnicas heurísticas para encontrar buenas soluciones, es decir, mayores a 20 vértices. Esto genera una pauta para que distintas metaheurísticas sean comparadas con respecto a su efectividad para encontrar la solución del problema; por ejemplo, si una metaheurística encuentra o se acerca lo suficiente al óptimo encontrado por el modelo binario significa que tendrá un buen comportamiento en la solución de problemas con más vértices.

Instancias para coloración equitativa

Puesto que el problema de coloración equitativa no posee una base de datos con instancias, éstas son propuestas y generadas de tal manera que se pueda demostrar mediante la resiliencia y la coloración resultante que el modelo propuesto las resuelve coloreándolas equitativamente.

Las instancias generadas se describen como sigue:

- *Instancias cuadradas*.- Son instancias de 16, 36, 64 y 100 vértices donde se colocan vértices en un plano uniformemente separados por una distancia de 0.4. Debido a que todos los puntos poseen la misma separación y a que son instancias cuadradas, se espera que se formen grupos lo más homogéneamente distribuidos con la cantidad de colores disponibles y a su vez, posea un óptimo en 4 colores.
- *Instancias triangulares*.- Son instancias de 18, 54 y 162 vértices donde fueron generadas figuras muy parecidas a triángulos hechas, a su vez, con otros triángulos. Los puntos con las que son formadas las figuras se encuentra muy cercanos entre sí. De este tipo de instancias se espera que el modelo distinga en la medida posible los triángulos por los que están compuestas y los pinte con distintos colores de la manera más uniformemente posible usando la cantidad de colores disponibles en ese momento.

En ambos casos descritos previamente, las ponderaciones se obtienen mediante la distancia entre dos puntos usando la fórmula de la distancia Euclidiana cuadrada descrita anteriormente. Esto es, se toma un vértice y se obtiene mediante la fórmula su distancia correspondiente a todos los demás vértices. Para el siguiente vértice se obtienen las distancias de la misma manera.

El formato para definir una instancia de este tipo es como sigue:

- Coordenada de un vértice

  X Y

  Donde X, Y especifican las coordenadas de un vértice en el espacio las cuales pueden establecerse enteras o de punto flotante

Por ejemplo, para definir dos vértices separados entre sí por 0.4 unidades en y, se define de la siguiente forma:

0.0 0.0

0.0 0.4

Finalmente, para usarlo en el modelo, este contenido es guardado en un archivo .txt con un nombre cualquiera y aunque no es necesario colocar un nombre para determinar la cantidad de vértices, es deseable que se haga para evitar confundirla con otras instancias que se puedan generar posteriormente.

5.1 Formato de salida

Para todas las instancias mencionadas existe un formato estándar de salida que el modelo proporciona y el cual es descrito a continuación.

La salida del programa divide a los resultados por coloraciones desde 1 hasta *k* colores. Para cada color se muestra el valor numérico correspondiente a la mejor solución encontrada; es decir, la menor dureza encontrada, también se muestran los valores de su solidez y resiliencia. Finalmente, se muestra la coloración correspondiente en forma de lista (vector) donde un número representa un color cualquiera, pero diferente a los demás, lo cual permite que esta coloración pueda ser fácilmente traducida en términos gráficos. Esta lista posee una longitud igual a la cantidad de vértices en el grafo.

Capítulo 6. Análisis de resultados

6.1 Coloración mínima

En la Tabla 12 se describen distintas instancias DIMACS, de las cuales se muestra el número de vértices, los colores necesarios para obtener el número cromático o, en su defecto, una coloración válida; asimismo se muestra el número de colores que necesitó el algoritmo GRASP propuesto para encontrar una solución factible y los colores extra requeridos para encontrarla.

Tabla 12. Desempeño del modelo propuesto para resolver instancias DIMACS de coloración mínima utilizando coloración de gráficas suaves.

Instancia	Número de vértices	Número cromático	Solución factible con GRASP CGS	Colores extra para encontrar una coloración válida
Mycielski-3	11	4	4	0
Mycielski-4	23	5	5	0
Mycielski-5	47	6	6	0
Mycielski-6	95	7	7	0
Mycielski-7	191	8	8	0
College Football	120	9	9	0
David Copperfield	87	11	11	0
Huckleberry Finn	74	11	11	0
Les Miserables	80	10	10	0
Anna Karenina	138	11	11	0
Queen Graph 5x5	25	5	5	0
Queen Graph 6x6	36	7	9	2
Queen Graph 7x7	49	7	9	2
Queen Graph 8x8	64	9	11	2
Queen Graph 9x9	81	10	11	1
Queen Graph 10x10	100	≥10	13	≤3
Queen Graph 11x11	121	11	14	3
Queen Graph 15x15	256	≥15	21	≤6
Miles Graph 250	128	8	8	0
Miles Graph 500	128	20	20	0
Miles Graph 750	128	31	31	0
Miles Graph 1000	128	42	42	0
Miles Graph 1500	128	73	73	0
Mulsol.i.1	197	49	49	0
Mulsol.i.2	188	31	31	0
Mulsol.i.3	184	31	31	0
Mulsol.i.4	185	31	31	0
Mulsol.i.5	186	31	32	1

Es destacable que el algoritmo resultó ser lo suficientemente bueno para resolver estas instancias mediante la coloración de gráficas suaves tan bien como algoritmos

especializados; y a pesar de que en el caso de los grafos que modelan el problema de las reinas necesitó hasta 6 colores extra para el caso de la instancia más grande, estas instancias destacan por su alta densidad y el pequeño porcentaje entre cantidad de colores con respecto al número de vértices (entre 7 y 10% de número de colores respecto al número de vértices) [27].

6.2 Coloración robusta

En cuanto al problema de coloración robusta, de [26] se retomó una tabla comparativa que emplea las mismas instancias propuestas anteriormente en este trabajo [31]; los resultados correspondientes se muestran a continuación, donde n es el número de vértices de la instancia y k el número de colores empleados por la metaheurística. Adicionalmente, se cuenta con una columna más la cual se discutirá posteriormente.

Tabla 13. Calidad de las soluciones del GRASP propuesto para la coloración robusta.

Instancia	n	k	Recocido simulado	Modelo binario	Búsqueda dispersa	GRASP_50 (Este trabajo)	Menor valor encontrado por CGS
al(10)	10	4	4.2386*	4.2386*	4.2386*	4.2386*	4
		5	1.8716*	1.8716*	1.8716*	1.8716*	
al(11)	11	4	3.4450*	3.4450*	3.4450*	---	5
		5	2.0227*	2.0227*	2.0227*	2.4762	
al(15)	15	5	5.6376*	5.6376*	5.6376*	5.6376*	4
		6	3.5592*	3.5592*	3.5592*	3.5592*	
al(20)	20	7	4.9557	---	4.9557	4.9557	6
		8	3.2285	---	3.2285	3.2285	
al(25)	25	9	5.5344	---	5.2584	5.2670	6
		10	3.7923	---	3.7368	3.7368	

De la Tabla 13, es importante mencionar que se evaluó la metaheurística propuesta contra otras e incluso con el modelo binario el cual da exactamente el óptimo -caso en el que se marca con un asterisco-. Este es un buen punto de comparación porque proporciona un parámetro adecuado para determinar la calidad de las soluciones del modelo en los problemas de coloración robusta; posterior a esto, se resolvieron instancias más grandes y los resultados se muestran enseguida:

Tabla 14. Comparación del GRASP propuesto contra diversas metaheurísticas para coloración robusta.

Instancia	n	k	Recocido simulado	Búsqueda TABÚ	Búsqueda dispersa	GRASP_100	GRASP_50 (Este trabajo)	Menor valor encontrado por CGS
al(50)	50	17	8.4552	9.8259	8.2587	8.9531	9.1834	10
		18	6.8232	7.4966	6.7164	7.1464	7.2610	
al(60)	60	20	8.8676	9.8331	8.8676	9.9687	10.1501	12
		21	7.7431	8.2181	7.2380	8.1430	8.6879	
al(70)	70	24	9.7195	11.1307	9.2634	11.2388	11.1096	13
		25	8.5979	9.5478	7.7048	9.2145	9.9406	
al(80)	80	27	10.7772	11.1946	9.9835	11.7512	11.6381	14
		28	9.0361	10.5845	8.5961	10.2631	10.0451	
al(90)	90	30	11.1629	12.2832	10.8911	13.4919	14.1151	15
		31	11.0050	11.3699	9.5008	11.5060	12.3635	

De la anterior Tabla, podemos concluir que, los resultados tienen valores muy cercanos a los mejores encontrados por un algoritmo exclusivo; además, debido a la naturaleza de la coloración de gráficas suaves, es posible obtener información adicional que antes no se tenía, es decir, el menor número encontrado que nos proporciona una cantidad adecuada de colores para clasificar y en la cual, si se hace una comparación con las aproximaciones iniciales, distan en algunos casos por el doble o más.

6.3 Coloración equitativa

De acuerdo con lo descrito en el apartado de instancias para la coloración equitativa, las instancias propuestas fueron resueltas tomando en consideración la distancia Euclidiana cuadrada para obtener la distancia entre los vértices, así como el GRASP para la coloración de gráficas suaves, los resultados se muestran a continuación:

Tabla 15. Resultados de la instancia equitativa cuadrada de 16 vértices usando coloración de gráficas suaves.

Instancia	Colores	Dureza	Solidez	Resiliencia
eq(16)	1	271.9744	2.26645333	0
	2	57.344	1.024	1.21333333
	3	15.4112	0.44455385	1.303433
	4	2.4576	0.1024	3.34134615
	5	1.8432	0.10472727	-0.02222222
	6	1.3312	0.09984	0.04895105
	7	0.8704	0.08462222	0.17983193
	8	0.4096	0.0512	0.65277778

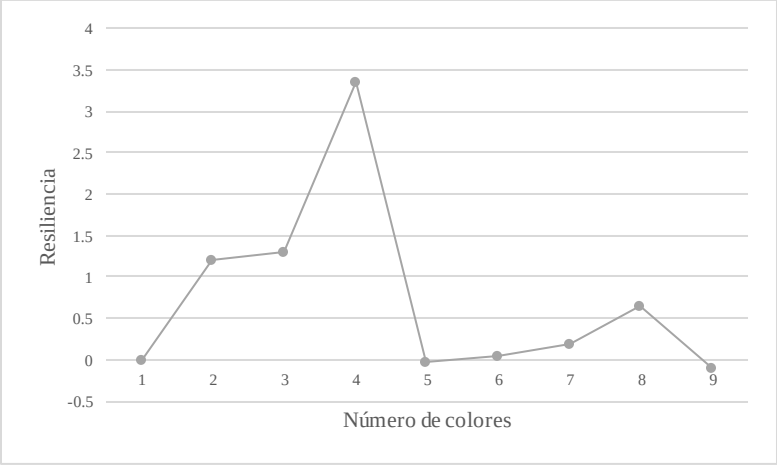

Figura 7. Representación gráfica de las resiliencias de la instancia equitativa con 16 vértices.

La solución de la primera instancia cuadrada con 16 vértices se muestra en la Tabla 15, aquí se detallan los valores de dureza, solidez y resiliencia para 1 hasta 9 colores. De esta Tabla, si se grafican los resultados de resiliencia obtendremos la Figura 7, donde es posible apreciar que la resiliencia da dos saltos importantes, en 4 y 8 colores. Debido a la característica del formato de salida que tiene el programa, podemos conocer visualmente cómo es la coloración y los resultados se muestran a continuación:

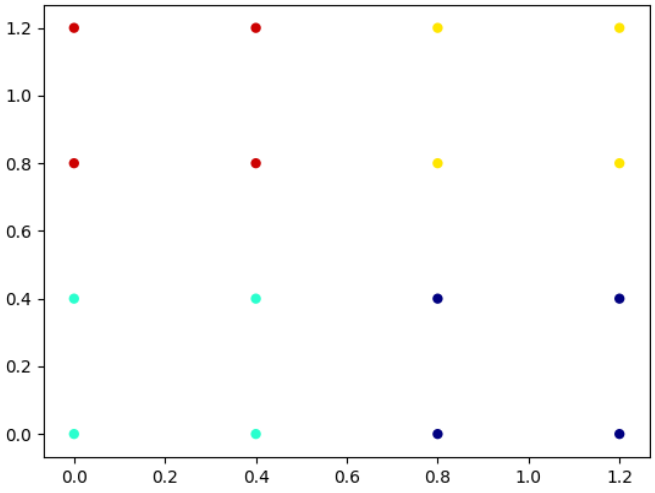

Figura 8. Representación gráfica de la instancia equitativa cuadrada con 16 vértices y coloración con 4 colores.

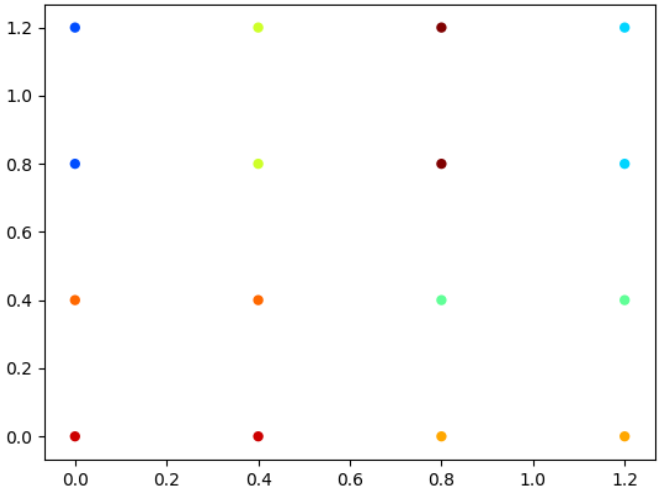

Figura 9. Representación gráfica de la instancia equitativa cuadrada con 16 vértices y coloración con 8 colores.

Si se observa la Figura 8 se notará que la coloración equitativa se logró agrupando a los vértices con 4 colores, lo cual era el resultado buscado; asimismo, un resultado adicional que corrobora que el modelo funciona correctamente para resolver el problema de la coloración equitativa se observa en la Figura 9 donde con 8 colores el modelo formó conjuntos de 2 vértices.

De manera similar, en la tabla siguiente se muestran los resultados obtenidos de dureza, solidez y resiliencia para la instancia cuadrada equitativa de 36 vértices desde 1 hasta 13 colores.

Tabla 16. Resultados de la instancia equitativa cuadrada de 36 vértices usando coloración de gráficas suaves.

Instancia	Colores	Dureza	Solidez	Resiliencia
eq(36)	1	7599.5136	12.06272	0
	2	1659.8016	5.42418824	1.22387562
	3	458.4448	2.31537778	1.34267958
	4	95.8464	0.6656	2.47863248
	5	62.3104	0.55833692	0.19211175
	6	32.256	0.3584	0.5578597
	7	23.1936	0.31102529	0.15231788
	8	14.336	0.22755556	0.36681034
	9	5.5296	0.1024	1.22222222

	10	4.9152	0.10502564	-0.025
	11	4.3008	0.10513067	-0.000999
	12	3.6864	0.1024	0.02666667
	13	3.2256	0.10128696	0.01098901

Debido a que en la instancia anterior la gráfica perteneciente a la resiliencia aportó información importante para comprobar qué tan bien se hizo la coloración, para esta instancia se procedió de la misma forma y el resultado puede observarse en la Figura 10.

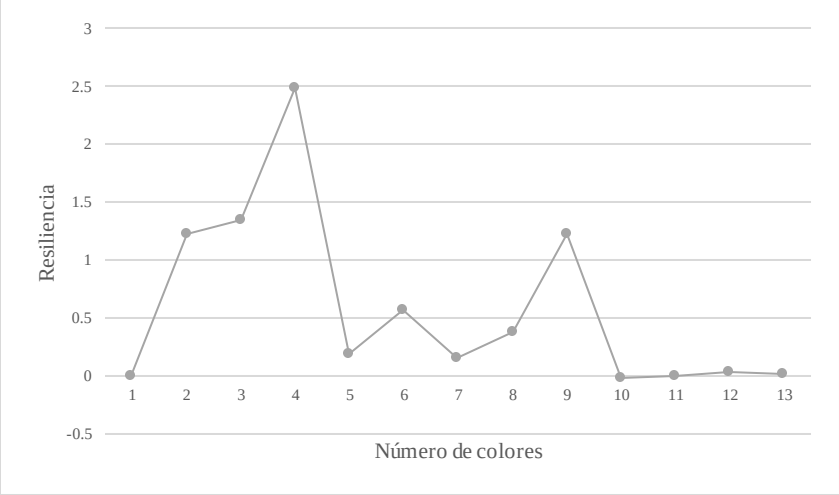

Figura 10. Representación gráfica de las resiliencias de la instancia equitativa con 36 vértices.

En este caso, dos valores en la resiliencia destacan de los demás, el 4 y 9, por lo que estos valores serán representados visualmente para sacar conclusiones.

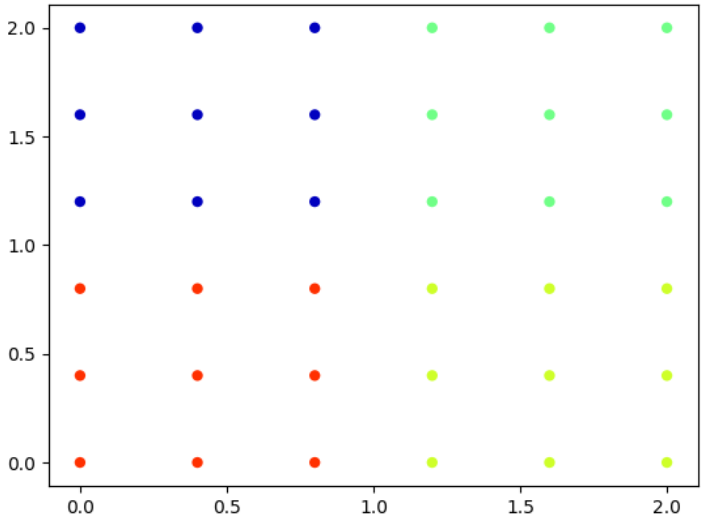

Figura 11. Representación gráfica de la instancia equitativa cuadrada con 36 vértices y coloración con 4 colores.

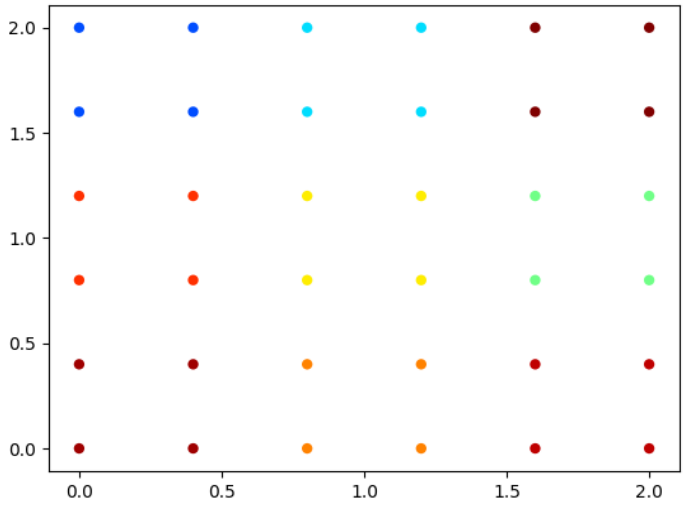

Figura 12. Representación gráfica de la instancia equitativa cuadrada con 36 vértices y coloración con 9 colores.

En la Figura 11 se puede apreciar que la coloración de la instancia es similar a la anterior, se elaboran grupos equitativamente distribuidos con 4 colores. Cuando el modelo resuelve la instancia con 9 colores, se puede apreciar en la Figura 12 que existe una característica

similar, la cual consiste en que se forman conjuntos con la misma cantidad de vértices en todos los casos.

Siguiendo con la forma de resolución, a continuación se presenta en la Tabla 17 los resultados que el modelo arrojó para la instancia equitativa con 64 vértices para 1 hasta 18 colores.

Tabla 17. Resultados de la instancia equitativa cuadrada de 64 vértices usando coloración de gráficas suaves.

Instancia	Colores	Dureza	Solidez	Resiliencia
eq(64)	1	78171.3408	38.7754667	0
	2	17275.2896	17.4146065	1.22660597
	3	4791.3472	7.36375082	1.3649098
	4	1087.8976	2.26645333	2.24901939
	5	670.3104	1.77518644	0.27674101
	6	416.7168	1.34714483	0.31773986
	7	257.4336	0.98795789	0.36356502
	8	162.2016	0.72411429	0.36436736
	9	104.7552	0.53568	0.35176651
	10	75.264	0.43555556	0.22987755
	11	54.3232	0.35233208	0.2362075
	12	44.6464	0.32196923	0.09430356
	13	35.7376	0.28467451	0.13100829
	14	27.136	0.23744	0.1989324
	15	18.6368	0.17828571	0.33179487
	16	9.8304	0.1024	0.74107143
	17	9.216	0.10417021	-0.01699346
	18	8.6016	0.10518261	-0.00962513

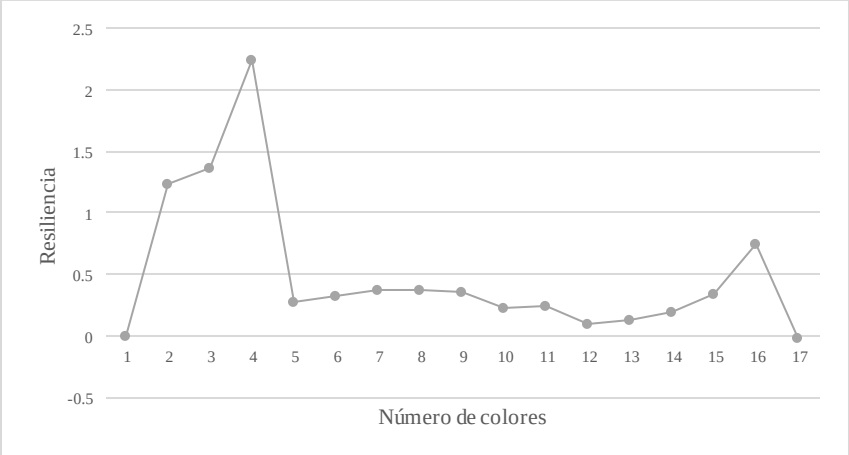

Figura 13. Representación gráfica de las resiliencias de la instancia equitativa con 64 vértices.

A partir del análisis de los datos de la Tabla 17 y la gráfica de la resiliencia, comienza a notarse un patrón en estas gráficas y es que el comportamiento de las mismas muestra un nuevo escalón en aquellos colores por los que la cantidad de los vértices dividido entre el color da exactamente 4. Si se analizan las coloraciones proporcionadas por el algoritmo tendremos como resultado las siguientes figuras:

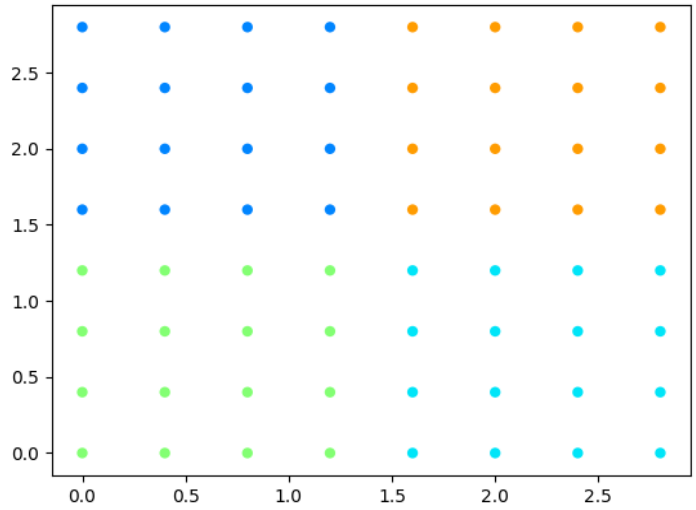

Figura 14. Representación gráfica de la instancia equitativa cuadrada con 64 vértices y coloración con 4 colores.

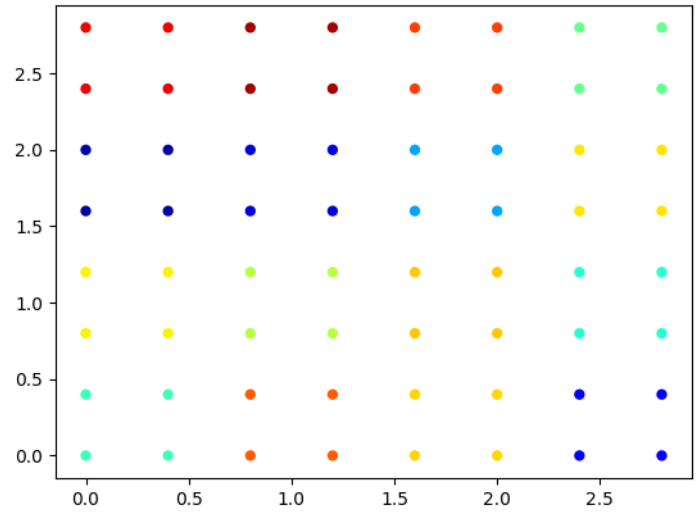

Figura 15. Representación gráfica de la instancia equitativa cuadrada con 64 vértices y coloración con 16 colores.

En la Figura 14 el comportamiento sigue siendo prácticamente igual a todas las instancias vistas, esto es, agrupar al grafo en grupos de 4 a causa de que son los colores que tiene disponibles. En la Figura 15 se comprueba el segundo escalón que retoma la instancia, la cual corresponde a una agrupación de 4 vértices. Gracias a esto, es posible suponer que, para la instancia restante de 100 vértices, existirán dos escalones importantes en la resiliencia, el primero con 4 colores y el segundo en 25. Para comprobar esto, la instancia fue resuelta y la siguiente Tabla muestra los resultados:

Tabla 18. Resultados de la instancia equitativa cuadrada de 100 vértices usando coloración de gráficas suaves.

Instancia	Colores	Dureza	Solidez	Resiliencia
eq(100)	1	472243.2	95.4026667	0
	2	104921.6	42.8251429	1.22772559
	3	28922.368	17.8901245	1.39378674
	4	6860.8	5.71733333	2.12910294
	5	4278.5792	4.50376758	0.26945568
	6	2616.6272	3.34037515	0.34828197
	7	1611.6224	2.42609824	0.37685074
	8	973.0048	1.69218226	0.43370977
	9	647.168	1.28011253	0.32190118
	10	499.1488	1.10921956	0.15406596
	11	367.9744	0.90959964	0.2194591
	12	277.2992	0.75627055	0.20274371
	13	220.416	0.65871448	0.14810068
	14	172.8	0.56260465	0.17083014
	15	141.056	0.49784471	0.13008061
	16	111.6672	0.42539886	0.170301
	17	93.7984	0.38423441	0.10713368
	18	77.568	0.34054244	0.1283011
	19	68.5568	0.32162449	0.05881998
	20	59.4432	0.297216	0.08212375
	21	50.5856	0.2689361	0.10515471
	22	41.7792	0.23567754	0.14111893
	23	32.9728	0.19698036	0.19645194
	24	27.4432	0.17332547	0.13647671
	25	17.408	0.11605333	0.49349845
	26	14.7456	0.10361773	0.12001425

	27	14.1312	0.10453216	-0.00874788

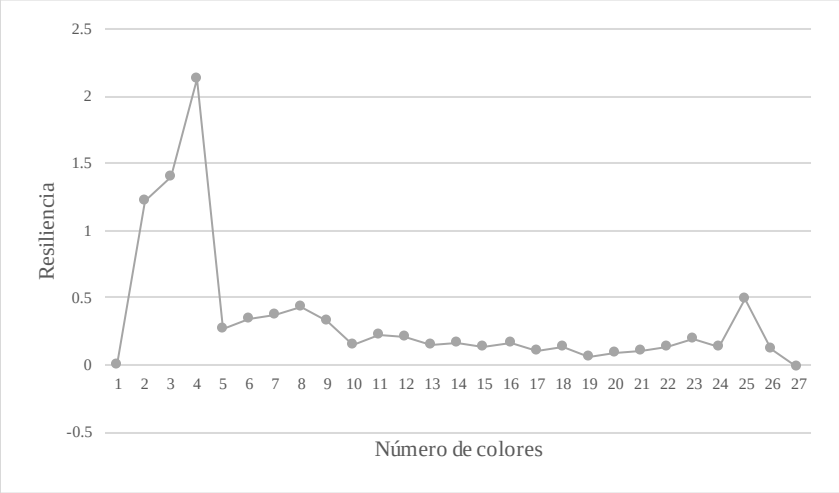

Figura 16. Representación gráfica de las resiliencias de la instancia equitativa con 100 vértices.

Analizando los datos de la Tabla 18 y la Figura 16 comprobamos las suposiciones previas, los escalones importantes coinciden en 4 y en la cantidad de vértices que tiene la instancia dividida entre 4; es decir, 25. Intuitivamente para las coloraciones obtenidas en las resiliencias ya mencionadas, el resultado será la agrupación en grupos de 4 colores con 25 vértices cada uno para el primer caso y grupos de 25 colores con 4 vértices para el segundo:

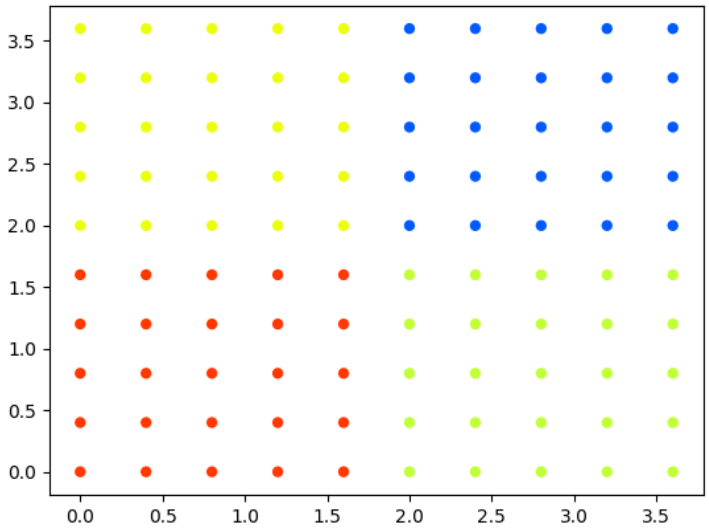

Figura 17. Representación gráfica de la instancia equitativa cuadrada con 100 vértices y coloración con 4 colores.

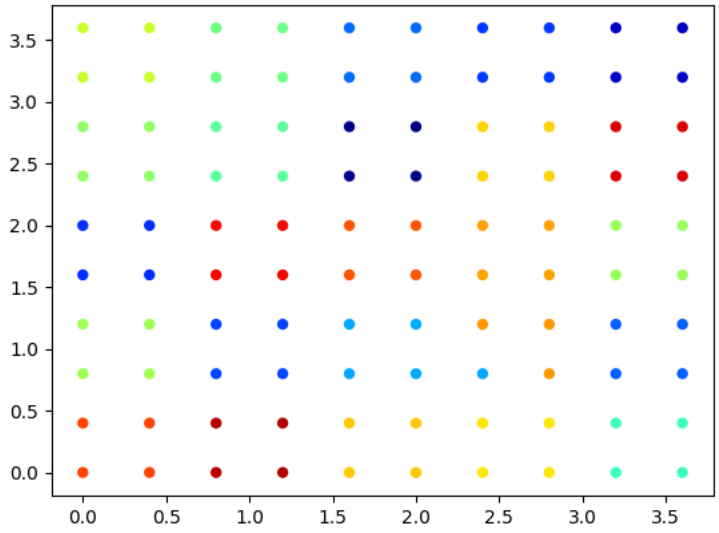

Figura 18. Representación gráfica de la instancia equitativa cuadrada con 100 vértices y coloración con 25 colores.

En las Figuras 17 y 18 se comprueba el comportamiento del modelo para colorear equitativamente las instancias; sin embargo, es notable destacar que a medida que crecen los vértices en la gráfica y debido a que estos se encuentran muy cercanos entre sí, el modelo parece tener ciertas dificultades pues en el caso de la Figura 18, existe un grupo que posee 5 vértices del mismo color, dejando a otro sólo con 3.

Como ha sido mencionado anteriormente, las instancias equitativas triangulares fueron diseñadas y construidas usando figuras parecidas a triángulos compuestas, a la vez, de otros triángulos. Así, el resultado esperado es el de obtener un salto en la resiliencia igual al número de triángulos por los que cada instancia se conforma; por ejemplo: en 3 colores para la instancia con 18 vértices; en 3 y 9 colores para la instancia con 54 vértices y; finalmente, en 3, 9 y 27 colores para la instancia con 162 vértices.

Los resultados de resolver la instancia equitativa triangular con 18 vértices son mostrados a continuación:

Tabla 19. Resultados de la instancia equitativa triangular de 18 vértices usando coloración de gráficas suaves.

Instancia	Colores	Dureza	Solidez	Resiliencia
eqTri(18)	1	1095.4496	7.15980131	0

	2	265.8432	3.69226667	0.9391344
	3	20.0768	0.44615111	7.27582085
	4	12.9216	0.41020952	0.08761763
	5	6.5856	0.2814359	0.45755935
	6	3.632	0.20177778	0.39478143
	7	1.3728	0.09706667	1.07875458
	8	0.8832	0.07850667	0.23641304
	9	0.576	0.064	0.22666667
	10	0.4672	0.06488889	-0.0136986

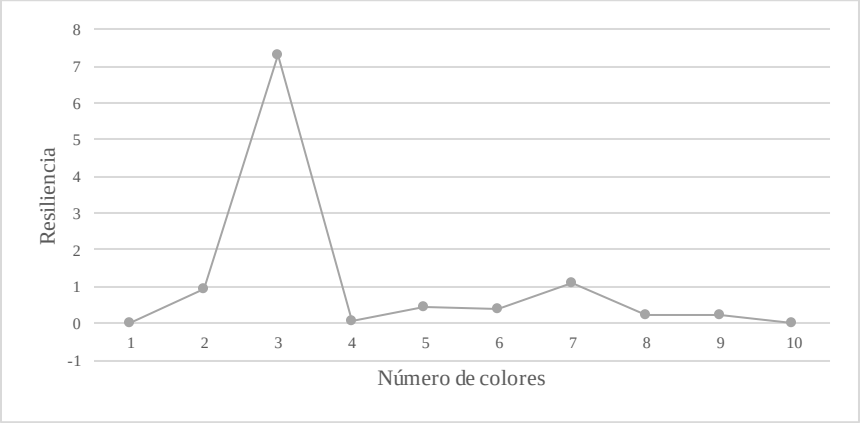

Figura 19. Representación gráfica de las resiliencias de la instancia equitativa triangular con 18 vértices.

En este caso, analizando la Tabla 19 y la Figura 19, se cumple el propósito de obtener un salto en 3 colores; no obstante, se aprecia un nuevo escalón para 7 colores. Analizando las coloraciones sólo para estos dos casos mencionados obtenemos lo siguiente:

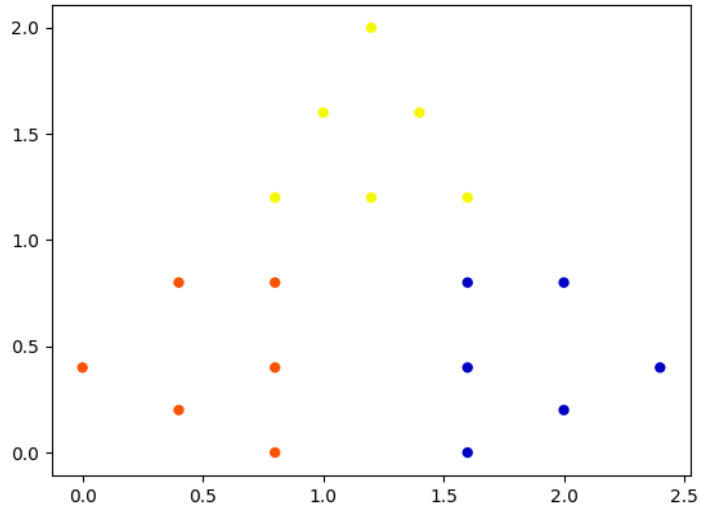

Figura 20. Representación gráfica de la instancia equitativa triangular con 18 vértices y coloración con 3 colores.

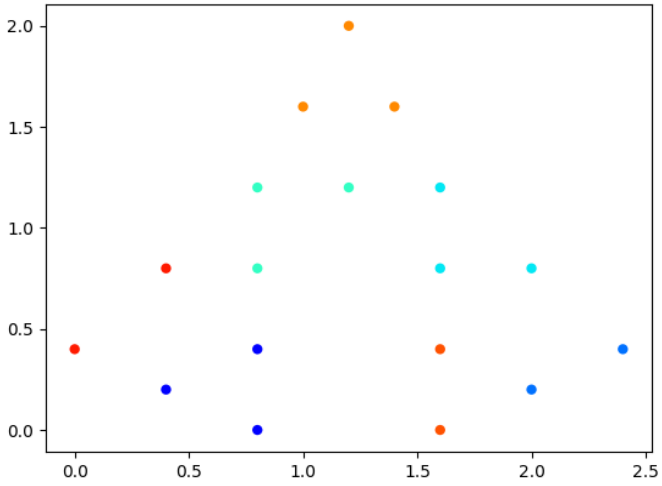

Figura 21. Representación gráfica de la instancia equitativa triangular con 18 vértices y coloración con 7 colores.

En el caso de la Figura 20 podemos apreciar claramente la coloración buscada; es decir, las figuras parecidas a triángulos fueron pintadas cada una con un color distinto y a pesar de que en la Figura 21 aparentemente no existe una secuencia para su coloración, el modelo está obteniendo que la instancia también puede pintarse con 7 colores.

La solución a la instancia equitativa triangular de 54 vértices usando el modelo de coloración de gráficas suaves devuelve los datos mostrados enseguida:

Tabla 20. Resultados de la instancia equitativa triangular de 54 vértices usando coloración de gráficas suaves.

Instancia	Colores	Dureza	Solidez	Resiliencia
eqTri(54)	1	1529859.73	1069.08437	0
	2	102511.54	146.027835	6.32109991
	3	19556.4132	42.6065647	2.42735529
	4	9240.8884	27.3804101	0.55609666
	5	4697.2122	17.7521247	0.54237369
	6	2229.8672	10.3234593	0.71959072
	7	1331.5298	7.34492403	0.40552294
	8	755.1062	4.86380805	0.51011799
	9	309.6576	2.29376	1.12045203
	10	243.4404	2.04916162	0.1193651
	11	196.388	1.86069595	0.10128773

Figura 22. Representación gráfica de las resiliencias de la instancia equitativa triangular con 54 vértices.

Esta instancia arroja resultados particulares a causa de que se tienen saltos en tres ocasiones: 2, 3 y 9 colores. Para tener una idea más clara del porqué ocurre esto, se usaron las coloraciones correspondientes a cada uno de los colores mencionados:

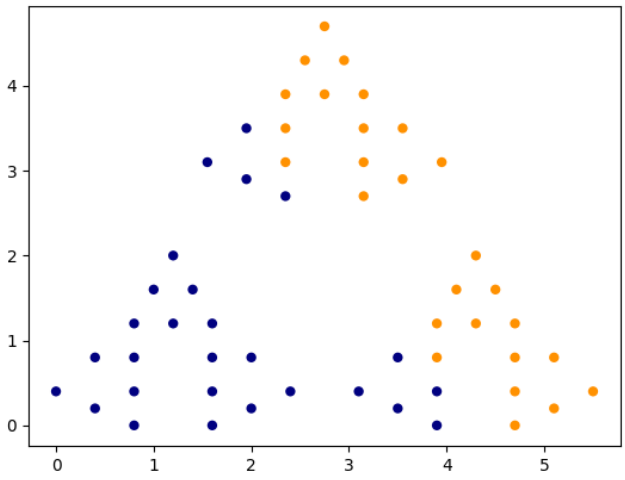

Figura 23. Representación gráfica de la instancia equitativa triangular con 54 vértices y coloración con 2 colores.

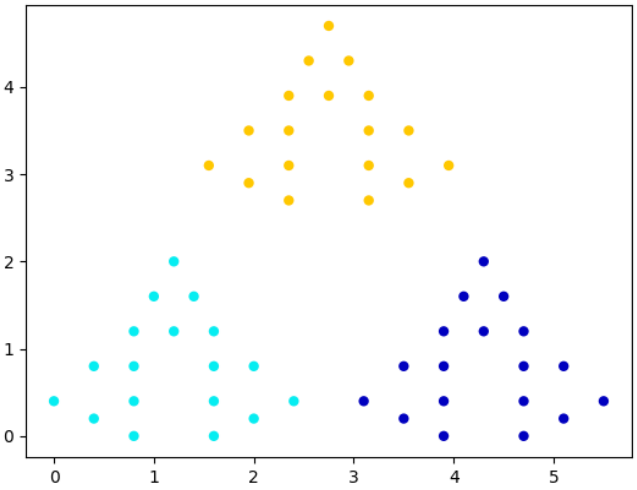

Figura 24. Representación gráfica de la instancia equitativa triangular con 54 vértices y coloración con 3 colores.

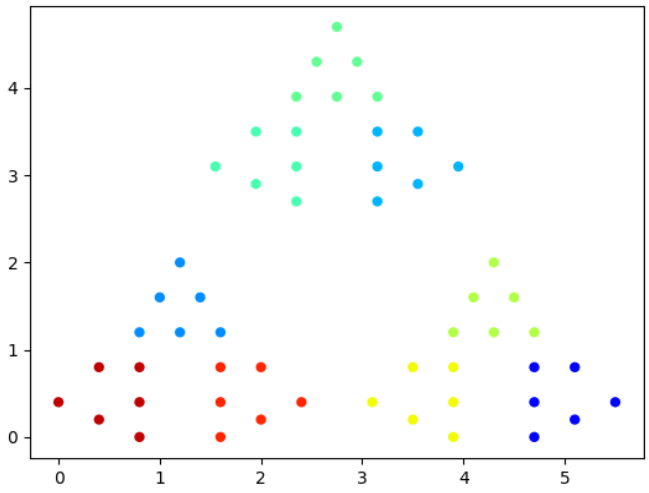

Figura 25. Representación gráfica de la instancia equitativa triangular con 54 vértices y coloración con 9 colores.

Si es observada la Figura 23, la coloración está hecha de tal forma que la instancia es partida prácticamente a la mitad en dos grupos; cada uno con 27 vértices. Esto no se trata de una equivocación, sino que el modelo indica que con una cantidad menor de recursos a los esperados y para esta instancia en particular, es posible hacer un mejor manejo de ellos. Para la Figura 24, el resultado que era esperado fue cumplido, cada uno de los triángulos de los que se compone la figura fueron pintados con un color diferente. No obstante, el modelo sí es capaz de identificar los triángulos más pequeños por los que está formada la instancia como se había previsto desde un inicio y esto se aprecia en la Figura 25 donde cada uno de ellos fue pintado con un color. Por ello se justifica la existencia de un nuevo escalón en la resiliencia para 9 colores; tomando en cuenta esto se espera que, para la instancia triangular siguiente se aprecien al menos 3 aumentos significativos en la resiliencia: para 3, 9 y 27 colores. Los resultados para la instancia equitativa triangular con 162 vértices son los siguientes:

Tabla 21. Resultados de la instancia equitativa triangular de 162 vértices usando coloración de gráficas suaves.

Instancia	Colores	Dureza	Solidez	Resiliencia
eqTri(162)	1	50051633.9	3838.02116	0
	2	12860487.6	1984.64315	0.93385958
	3	890727.548	207.483705	8.56529643
	4	571551.633	178.637798	0.16147707
	5	311125.7	122.326689	0.4603338

	6	144733.759	68.7244818	0.77995796
	7	82463.5689	45.9772985	0.49474815
	8	44343.2236	28.4388158	0.61670932
	9	9859.0464	7.15980131	2.97201188
	10	8779.2708	7.13066179	0.00408651
	11	7763.7762	6.98238396	0.02123599
	12	6805.9258	6.72190202	0.03875122
	13	5862.51115	6.31474397	0.06447737
	14	4995.60974	5.83404541	0.08239541
	15	4089.63313	5.15196917	0.13239137
	16	3457.5648	4.67791618	0.10133849
	17	2952.1718	4.2730456	0.09474989
	18	2487.63649	3.8389452	0.11307804
	19	2058.3237	3.37634035	0.13701369
	20	1686.70433	2.93288876	0.1511996
	21	1317.62178	2.42273507	0.21056932
	22	1097.56438	2.1293136	0.13780097
	23	898.589875	1.83564856	0.1599789
	24	723.544713	1.55350448	0.18161781
	25	542.22435	1.22155616	0.27174217
	26	406.423675	0.95924252	0.27345916
	27	208.2688	0.51424395	0.86534526
	28	173.6608	0.44799174	0.14788711
	29	166.3808	0.44788297	0.00024287

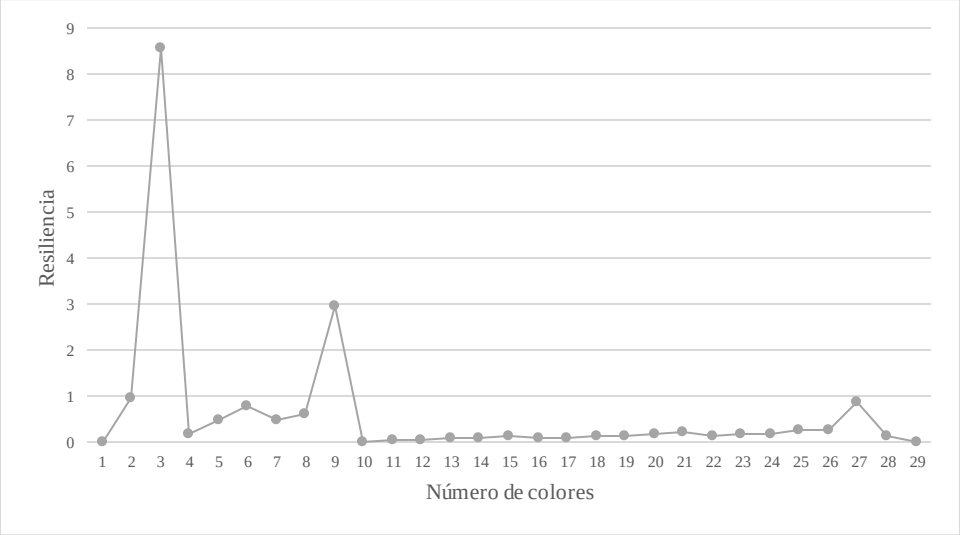

Figura 26. Representación gráfica de las resiliencias de la instancia equitativa triangular con 162 vértices.

En la Tabla 21 y Figura 26 se comprueba los aumentos supuestos en la resiliencia, esto significa que, para las coloraciones correspondientes a estos podrá notarse que el modelo separa y pinta los triángulos desde los más grandes a los más pequeños contenidos en la instancia:

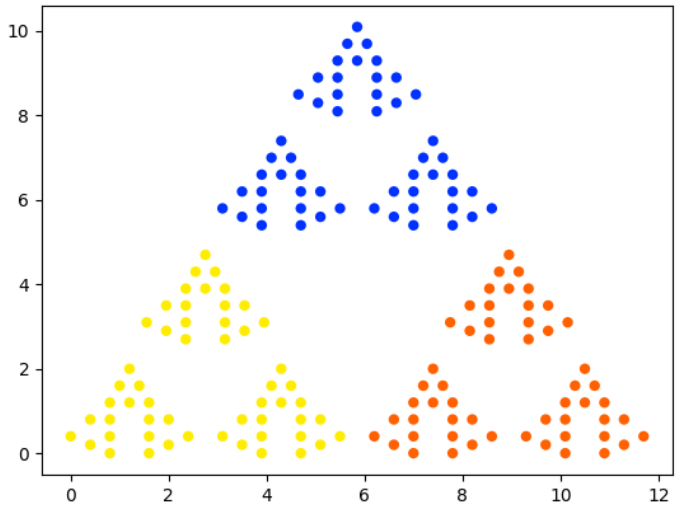

Figura 27. Representación gráfica de la instancia equitativa triangular con 162 vértices y coloración con 3 colores.

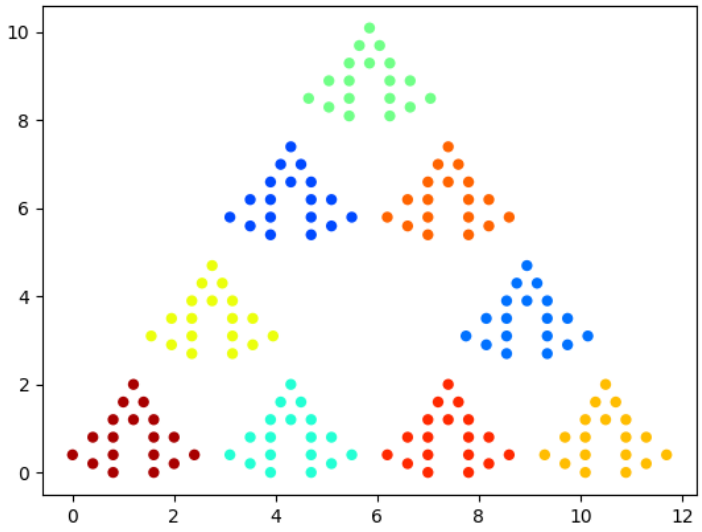

Figura 28. Representación gráfica de la instancia equitativa triangular con 162 vértices y coloración con 9 colores.

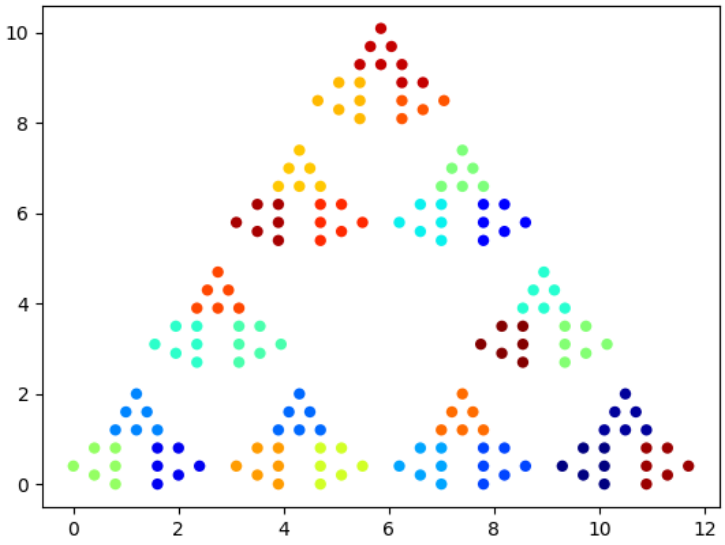

Figura 29. Representación gráfica de la instancia equitativa triangular con 162 vértices y coloración con 27 colores.

Las Figuras 27-29 demuestran que el modelo de coloración de gráficas suaves puede hacer una coloración equitativa muy efectiva y en algunos casos aporta resultados no previstos, pero a considerarse como en la instancia triangular equitativa con 54 vértices donde el trabajo puede ser eficientemente elaborado con sólo 2 recursos.

Conclusiones

La coloración de gráficas suaves es el punto de partida para crear un modelo integrador a otros problemas de coloración clásicos, permitiendo así resolver instancias hasta el momento exclusivas para un problema en particular, puesto que, al momento de evaluar el modelo propuesto con las ya mencionadas instancias de cada problema de coloración, este mismo resultó ser tan bueno como los modelos dedicados a resolver uno específico.

Asimismo, el algoritmo GRASP propuesto permite tener la ventaja de ser relativamente sencillo de codificar, con lo cual es posible centrarse en la tarea de probar distintos tipos de instancias y su correspondiente análisis.

No obstante, el uso de este modelo también supone otras ventajas como el aporte de nuevos datos de interés no conocidos anteriormente por modelos exclusivos; por ejemplo:

En el caso de la coloración mínima, debido a que en la CGS es necesario resolver la instancia desde 1 hasta n colores para obtener el número cromático, no es necesario hacer modificaciones adicionales al modelo para que una vez se haya encontrado el número cromático también se tenga la solución de esa misma instancia para el problema de coloración de gráficas débiles.

Para la coloración robusta, tomando en cuenta los datos proporcionados por la resiliencia, el modelo permite identificar una cantidad adecuada de colores con los que se puede resolver este tipo de instancias, lo que se traduce en una mejor asignación de recursos.

Finalmente, para la coloración equitativa se obtiene que, gracias a la falta de entrenamiento previo del modelo, se considera en algunos casos un número menor de colores a los que se pensaron como adecuados en un inicio, lo cual significa que en determinadas instancias, es posible hacer un trabajo eficiente con una cantidad menor de recursos.

Referencias

[1] Christos H. Papadimitriou, Kenneth Steigliths.: Combinatorial Optimization. Algorithms and Complexity. Dover Publications, Inc. 1998.

[2] N. Musliu y M. Schwengerer, “Algorithm Selection for the Graph Coloring Problem”, en Learning and Intelligent Optimization: 7th International Conference, LION 7, Catania, Italy, January 7-11, 2013, Revised Selected Papers, G. Nicosia y P. Pardalos, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 389–403.

[3] Diestel, R. (2000) Graph Theory. Springer-Verlag, New York.

[4] M. de Longueville, A Course in Topological Combinatorics. New York, NY: Springer New York, 2013.

[5] J. Nešetřil y G. Woeginger, “Graph colorings”, Theoretical Computer Science, vol. 349, núm. 1, p. 1, dic. 2005.

[6] P. Lara Velázquez, M. Á. Gutiérrez Andrade, S. G. De los Cobos Silva, y E. A. Rincón García, “Coloración de gráficas suaves”, Revista de Matemática: Teoría y Aplicaciones, vol. 22, núm. 2, p. 311, ago. 2015.

[7] M. Á. Gutiérrez Andrade, P. Lara Velázquez, R. López Bracho, y J. Ramírez Rodríguez, “Heurísticas para el Problema de Coloración Robusta”, Revista de Matemática: Teoría y Aplicaciones, vol. 18, núm. 1, p. 137, mar. 2011.

[8] H. Fan, H. A. Kierstead, G. Liu, T. Molla, J.-L. Wu, y X. Zhang, “A note on relaxed equitable coloring of graphs”, Information Processing Letters, vol. 111, núm. 21–22, pp. 1062–1066, nov. 2011.

[9] R. Martí y G. Reinelt, The linear ordering problem: exact and heuristic methods in combinatorial optimization. Heidelberg [Germany]; New York: Springer, 2011.

[10] J. Flores Cruz, P. Lara Velázquez, M. Á. Gutiérrez Andrade, S. G. De los Cobos Silva, y E. A. Rincón García, “Un sistema clasificador utilizando coloración de gráficas suaves”, Revista de Matemática: Teoría y Aplicaciones, núm. 1, feb. 2016.

[11] Pedro Lara-Velázquez, Sergio de-los-Cobos-Silva, Miguel A. Gutiérrez-Andrade, Eric A. Rincón-García, Antonin Ponsich, Roman Mora-Gutiérrez, "Pattern recognition using soft graph coloring.", procc. XVIII SIGEF CONGRESS; GIRONA(Spain), pp.1-11, 6-8 July 2015.

[12] “Graph Coloring Instances”. [En línea]. Disponible en: http://mat.gsia.cmu.edu/COLOR/instances.html. [Consultado: 10-abr-2016].

[13] F. Wang y Z. Xu, “Metaheuristics for robust graph coloring”, Journal of Heuristics, vol. 19, núm. 4, pp. 529–548, ago. 2013.

[14] Ramírez Rodríguez, Javier; Gutiérrez Andrade, Miguel Ángel; Lara Velázquez, Pedro; López Bracho, Rafael; 2005. Un Algoritmo Evolutivo para Resolver el Problema de Coloración Robusta. Revista de Matemática: Teoría y Aplicaciones 12: 111-120.

[15] M. de Longueville, A Course in Topological Combinatorics. New York, NY: Springer New York, 2013.

[16] Garey, D. S. Johnson, Computers and intractability. A guide to thetheory of NP-completeness, Freeman, 1979.

[17] Marx, Dániel. Graph colouring problems and their applications in scheduling. Periodica Polytechnica Electrical Engineering, 2004.

[18] J. Yáñez y J. Ramírez, “The robust coloring problem”, European Journal of Operational Research, vol. 148, núm. 3, pp. 546–558, ago. 2003.

[19] F. Glover y M. Laguna, “Tabu Search”, en Handbook of Combinatorial Optimization, P. M. Pardalos, D.-Z. Du, y R. L. Graham, Eds. New York, NY: Springer New York, 2013, pp. 3261–3362.
[20] Sergio de-los-Cobos-Silva, John Goddard Close, Miguel Á. Gutiérrez-Andrade, Alma Edith Martínez Licona, Búsqueda y exploración estocástica. México, D.F.: Universidad Autónoma Metropolitana, Unidad Iztapalapa, 2010.
[21] M. Aboytes, Algoritmo de búsqueda tabú para una variante del problema de coloración, Tesis Doctoral, Universidad Autónoma de México, Ciudad de México, México, 2011.
[22] F. Kuhn, “Weak graph colorings: distributed algorithms and applications”, 2009, p. 138.
[23] W. Meyer, “Equitable Coloring”, The American Mathematical Monthly, vol. 80, núm. 8, p. 920, oct. 1973.
[24] Hanna Furmańczyk, Equitable coloring of graph products, Opuscula Math. 26, no. 1 (2006).
[25] Pedro Lara-Velázquez, Sergio de-los-Cobos-Silva, Miguel A. Gutiérrez-Andrade, Eric A. Rincón-García, Antonin Ponsich, Roman Mora-Gutiérrez, "Comparative Philology Among Iberian Languages Using Soft Graph Coloring", procc. XVIII SIGEF CONGRESS; GIRONA(Spain), pp.39-48, July 2015.
[26] Lara, P. (2005) “Un algoritmo evolutivo para resolver el problema de coloración robusta”, Tesis Doctoral, Universidad Nacional Autónoma de México, Ciudad de México, México.
[27] Pedro Lara-Velázquez, Sergio de-los-Cobos-Silva, Miguel A. Gutiérrez-Andrade, Eric A. Rincón-García, Antonin Ponsich, Roman Mora-Gutiérrez, "Comparative Philology Among Iberian Languages Using Soft Graph Coloring", procc. XVIII SIGEF CONGRESS; GIRONA(Spain), pp.39-48, July 2015.
[29] T. A. Feo, MGC Resende, "Greedy Randomized Adaptative Search Procedures", Journal of Global Optimization, 1995.
[30] N. Musliu y M. Schwengerer, “Algorithm Selection for the Graph Coloring Problem”, en Learning and Intelligent Optimization: 7th International Conference, LION 7, Catania, Italy, January 7-11, 2013, Revised Selected Papers, G. Nicosia y P. Pardalos, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 389–403.
[31] Ramírez, J. (2001) Extensiones del problema de coloración de grafos. Tesis Doctoral, Universidad Complutense de Madrid, Madrid, España.
[32] C. Lenzen y B. Patt-Shamir, “Fast Routing Table Construction Using Small Messages”, ArXiv e-prints, oct. 2012.
[33] A. Framework, S. Fitzpatrick, y L. Meertens, Soft, Real-Time, Distributed Graph Coloring using Decentralized, Synchronous, Stochastic, Iterative-Repair, Anytime Algorithms. 2001.
[34] A. Hertz, “Chromatic Scheduling D. de Werra”, 2013.

Casa abierta al tiempo
UNIVERSIDAD AUTÓNOMA METROPOLITANA

ACTA DE EXAMEN DE GRADO

No. 00065
Matrícula: 2153805524

SOLUCIÓN A PROBLEMAS DE COLORACIÓN CLÁSICOS UTILIZANDO COLORACIÓN DE GRÁFICAS SUAVES

DANIEL EDAHI URUETA HINOJOSA
ALUMNO

REVISÓ

LIC. JULIO CESAR DE LARA ISASSI
DIRECTOR DE SISTEMAS ESCOLARES

En la Ciudad de México, se presentaron a las 14:00 horas del día 27 del mes de octubre del año 2017 en la Unidad Iztapalapa de la Universidad Autónoma Metropolitana, los suscritos miembros del jurado:

DR. SERGIO GERARDO DE LOS COBOS SILVA
DRA. HÉRICA SÁNCHEZ LARIOS
DR. PEDRO LARA VELAZQUEZ

Bajo la Presidencia del primero y con carácter de Secretario el último, se reunieron para proceder al Examen de Grado cuya denominación aparece al margen, para la obtención del grado de:

MAESTRO EN CIENCIAS (CIENCIAS Y TECNOLOGIAS DE LA INFORMACION)

DE: DANIEL EDAHI URUETA HINOJOSA

y de acuerdo con el artículo 78 fracción III del Reglamento de Estudios Superiores de la Universidad Autónoma Metropolitana, los miembros del jurado resolvieron:

APROBAR

Acto continuo, el presidente del jurado comunicó al interesado el resultado de la evaluación y, en caso aprobatorio, le fue tomada la protesta.

DIRECTOR DE LA DIVISIÓN DE CBI

DR. JOSE GILBERTO CORDOBA HERRERA

PRESIDENTE

DR. SERGIO GERARDO DE LOS COBOS SILVA

VOCAL

DRA. HÉRICA SÁNCHEZ LARIOS

SECRETARIO

DR. PEDRO LARA VELAZQUEZ