BENEMÉRITA UNIVERSIDAD AUTONOMA DE PUEBLA

Facultad de Ciencias de la Computación

"UN ALGORITMO HEURÍSTICO PARA EL COLOREO DE GRAFOS"

Tesis presentada para obtener el título de licenciatura en ingeniería en ciencias de la computación

PRESENTA: LILIBETH ZUÑIGA VARGAS

ASESOR: DR. GUILLERMO DE ITA LUNA

PUEBLA, MÉXICO 04 ABRIL 2016

1 CONTENIDO

RESUMEN

El coloreo de grafos es un tema de gran importancia, debido a las numerosas situaciones de la vida que pueden ser modeladas por este.

En la actualidad no se conoce un algoritmo polinomial que resuelva el *Coloreo de grafos* de forma determinista y exacta, por lo que se considera como un problema en la clase de complejidad NP- completo.

En esta tesis, se da una breve explicación de las definiciones básicas de la teoría de grafos, y procedimientos básicos y generales utilizados en algoritmos para el coloreo de grafos, ya que son fundamentales para la comprensión de problemas típicos en la teoría de grafos.

El motivo de esta tesis es proponer un algoritmo heurístico, para calcular el número cromático de un grafo de forma eficiente, siendo esté el problema medular en el área de *Coloreo de grafos*.

En cierta forma, podemos considerar nuestra propuesta heurística, como una búsqueda local sobre vecindades de segundo orden, lo que nos muestra que es eficiente en tiempo.

Se presenta tanto el diseño, implementación y análisis del algoritmo propuesto, así como presentamos algunas instancias de prueba que muestran como ejecuta el algoritmo sobre ellas.

1.1 PLANTEAMIENTO DEL PROBLEMA

El desarrollo de la Teoría de Grafos tuvo un fuerte impulso a través de las ideas propuestas al intentar resolver la Conjetura de los Cuatro Colores, el cual ha sido uno de los grandes problemas matemáticos no resueltos durante más de 120 años.

En 1976 Wolfgang Haken y Kenneth Appel de la Universidad de Illinois, usaron un enfoque que comprende entre otras cosas, la revisión puntual de 10^{10} diferentes configuraciones, a través de un ordenador de alta velocidad.

Actualmente en la Computación, el Coloreo de Grafos es un problema de interés práctico y desafiante, aun considerando los últimos adelantos, dado que este es de los problemas más difíciles de resolver entre los problemas de la clase NP-Difícil. El problema de coloreo de nodos de un grafo, puede plantearse de la forma siguiente:

Dado un Grafo $G = (V, E)$.

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

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

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 $X(G)$. Dado un grafo G, el problema de coloración mínima, busca una función de coloración que no utilice más de $X(G)$ colores.

El número cromático es computable en tiempo polinomial cuando $X(G) \leq 2$, esto es, que el grafo es efectivamente dos coloreable. Si $X(G) \geq 3$, determinar el valor exacto para $X(G)$ se convierte en un problema NP-Completo.

Esta investigación propone un nuevo algoritmo que calcula el número cromático de un grafo arbitrario, y lo colorea.

1.2 OBJETIVOS DE LA TESIS

1.2.1 Objetivo general

- Diseñar e implementar un algoritmo heurístico que calcule el número cromático de un Grafo.

1.2.2 Objetivos específicos

- Proponer y Diseñar una heurística para calcular el número cromático de un Grafo.
- Implementar la heurística en un lenguaje portable (C#)
- Estudiar y analizar la complejidad computacional del método propuesto.

2 ESTADO DEL ARTE

La Teoría de Grafos es útil para modelar una gran cantidad de problemas y su teoría es aplicable a diferentes áreas de estudio, como: física, química, comunicación, informática, ingeniería civil, ingeniería eléctrica, genética, psicología lingüística, etc.

El comienzo de la Teoría de Grafos se atribuye a Euler (1736), quien resolvió el problema de los puentes de Königsberg, basándose en el concepto de grafos. Luego, en 1845, Kirchhoff empleó la Teoría de Grafos para analizar las redes eléctricas [1].

El Coloreo de Grafos consiste en colorear los vértices de un grafo (que en nuestro caso, asumiremos como grafo no dirigido) usando el menor número de colores, exigiendo que los nodos extremos de cada arista tengan distinto color. El Coloreo de Grafos fue uno de los problemas NP-Completos enlistados por vez primera por Karp [3].

En 1840, la conjetura de Möbins acerca del problema de los Cuatro Colores, que plantea si es suficiente usar sólo cuatro colores para pintar cualquier mapa plano, de tal manera, que si dos regiones son adyacentes, no pueden tener el mismo color, es reconocida como el problema de los cuatro colores.

Francis Guthrie abogado y botánico, observa que puede colorear un mapa complejo de los Cantones de Inglaterra con cuatro colores, en 1852 plantea el problema a su hermano Frederick Guthrie, fundador de la Physical Society, quien plantea el problema a Augustus De Morgan [2].

De Morgan, interesado en la conjetura de los cuatro colores, lo difundió entre sus colegas, una de las primeras personas con las que trato el tema fue con el matemático y físico Irlandés Sir William Rowan Hamilton, a quien le escribió una carta el 23 de octubre de 1852. Hamilton responde cuatro días después, diciendo que no compartía el interés. Decepcionado, De Morgan se pone en contacto con otros matemáticos [2].

Tras la muerte de Augustus De Morgan en 1871, el tema parece olvidado, pero Arthur Cayley de la Universidad de Cambridge continúo con la investigación.

Cayley publica una nota donde explica, que al probar el teorema de los cuatro colores, pueden imponerse condiciones más restrictivas sobre los mapas a colorear. En particular, basta con limitarse a mapas cúbicos, esto es, aquellos en los que hay exactamente tres regiones en cada punto de encuentro.

Alfred Bray Kempe, alumno de Cayley, se interesa por el problema de los cuatro colores, en 1879 obtiene su solución del teorema y lo publica en el Amer. En 1880, publica unas versiones simplificadas de su prueba, donde corrige algunas erratas de su prueba original, pero continúa errado [2].

En 1890 Percy John Heawood encuentra un caso para el que la prueba de Kempe no funciona, y prueba que se podía resolver el problema con cinco colores, usando el argumento de las cadenas de Kempe.

Ya en el siglo XX, Appel, Haken y Koch presentan una prueba sobre el problema de los cuatro colores, donde aplican un algoritmo de descarga complicado, que produjo una lista de 1,936 configuraciones de diferentes topologías en grafos que eran difíciles de colorear. Se demostró que eran reducibles con la ayuda de un ordenador programado por Koch, para buscar las extensiones requeridas del coloreado, y lo que requirió de 1,200 horas de cálculo en una IBM 360 [2].

Appel y Haken [4] completaron la demostración del teorema de los cuatro colores en 1976.

2.1 Conceptos preliminares de la teoría de Grafos

A continuación se exponen algunos conceptos que se utilizarán a lo largo de esta tesis:

Un Grafo (G) es un conjunto de vértices o nodos (V) unidos por un conjunto de líneas o flechas llamadas aristas (E) dependiendo si el grafo es dirigido o no dirigido.

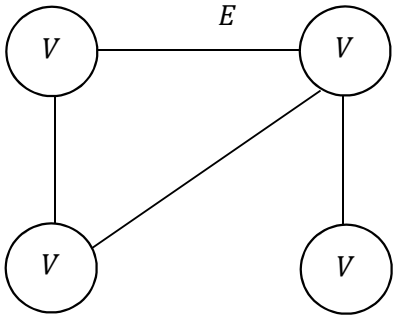


Figura 2.1: Grafo no dirigido.

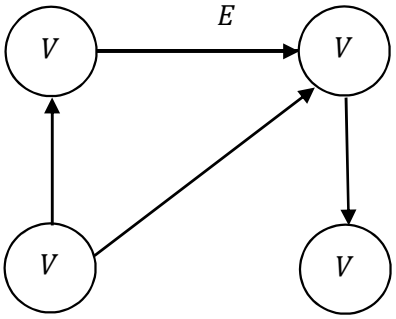


Figura 2.2: Grafo dirigido.

Un Grafo Simple G es una estructura matemática que consta de un par ordenado de conjuntos (V, E) siendo $V \neq \emptyset$. En un grafo simple, una arista es un par $\{x, y\}$ no ordenado de vértices diferentes.

Se dice que la arista $\{x, y\}$ es incidente a los vértices x e y, y que los vértices x e y son adyacentes o vecinos.

El vecindario de $x \in V$, es $N(X) = \{ y \in V : \{ x, y \} \in E \}$. Y su vecindario cerrado es: $N(x) \cup \{x\}$, que se denota por $N[x]$. Tómese en cuenta que x no está en $N(x)$.

Se denota la cardinalidad de un conjunto A, por $|A|$. Dado un grafo $G = (V, E)$, el grado de un vértice $x \in V$, denotado por $\delta(x)$, es $|N(x)|$. El tamaño del vecindario de $x, \delta(N(x))$, es $\delta(N(x)) = \sum_{y \in N(x)} \delta(y)$. El grado máximo de G, o simplemente el grado de G es: $\Delta(G) = max\{\delta(x) : x \in V\}$, mientras que se denota el grado mínimo con $\delta mi\ (G) = min\{ \delta(x) : x \in V \}$ y con $\delta(G) = (2 \cdot |E|) / |V|$ el grado promedio del grafo.

Dado un subconjunto de vértices $S \subseteq V$, el subgrafo de G denotado por $G|S$ tiene a S como conjunto de vértices y como conjunto de aristas a $E(G|S) = \{\{u,v\} \in E: u,v \in S\}$. A $G|S$ se le llama el subgrafo de G inducido por S. Escribimos $G - S$ para denotar el grafo $G|(V - S)$. El subgrafo inducido por $N(v)$ se denota como $H(v) = G|N(v)$ que tiene a $N(v)$ como el conjunto de nodos y todas las aristas de ellos.

Un camino de un vértice v al vértice w en un grafo es una secuencia de aristas: $v_0, v_1, v_2, \ldots, v_{n-1}, v_n$. De modo que $v = v_0, v_n = w, v_k$ es adyacente a $v_k + 1\, y$ la longitud de la ruta es n. Un camino simple es un camino en el que$v_0, v_1, \ldots, v_{n-1}, v_n$, son todos vértices distintos. Un ciclo es un camino no vacío tal que los vértices primero y el último son iguales, y un ciclo simple es un ciclo en el que ningún vértice se repite, a excepción de los vértices inicial y final.

Un k −ciclo es un ciclo de longitud k, es decir, un k-ciclo tiene k aristas. Un ciclo de longitud impar se llama un ciclo impar, mientras que un ciclo de longitud par se llama un ciclo par. Un grafo G es acíclico si no tiene ciclos.

Un grafo completo de n nodos tiene $n \cdot (n-1)/2$ aristas distintas, se denota Kn el grafo completo de n nodos. Un grafo G es un grafo regular si todos los vértices tienen el mismo grado, G es k −regular si es regular, de grado k.

Un componente conectado de G es un subgráfo máximo inducido de G, es decir, un subgrafo conexo que no es un subgrafo propio de cualquier otro subgrafo conexo de G. Note que, en un componente conectado, para cada par de vértices x, y, hay un camino de x a y. Si un grafo a cíclico también es conectado, entonces se llama un árbol libre.

Una coloración de un grafo $G = (V, E)$ es una asignación de colores a sus vértices. Un coloreo es propio si los vértices adyacentes siempre tienen colores diferentes. Un k −coloreo de G es una asignación de V al conjunto $\{1, 2, \ldots, k\}$ de k-colores.

El número cromático de G denotado por $\chi(G)$ es el valor mínimo k tal que G tiene un $k - coloreo$ propio. Si $\chi(G) = k$, G se dice entonces que es k-cromático.

Determinar el valor de $\chi(G)$ es computable polinomialmente cuando $\chi(G) \leq 2$, pero cuando $\chi(G) \geq 3$, el problema es NP-completo, incluso para grafos G con grado $\Delta(G) \geq 3$.

Sea $G = (V, E)$ un grafo, G es bipartito si V se puede dividir en dos subconjuntos $U1$ y $U2$, llamados conjuntos partitos, de manera que todas las aristas de G se unen a un vértice de $U1$ con un vértice de $U2$. Dado un grafo $G = (V, E), S \subseteq V$ es un conjunto independiente en G si para cada dos vértices V_1, V_2 en $S, \{V_1, V_2\} \notin E$. Sea $I(G)$ el conjunto de todos los conjuntos independientes de G. Un conjunto independiente $S \in I(G)$ es maximal, si no es un subconjunto de cualquier conjunto independiente más grande y es máximo si tiene el tamaño más grande de todos los conjuntos independientes en $I(G)$.

Si $G = (V, E)$ es un grafo $k-$cromático, entonces es posible particionar V en k conjuntos independientes $V_1, V_2, \ldots, V_K$ llamados clases de color, pero no es posible particionar V en $k - 1$ conjuntos independientes.

2.2 APLICACIONES DEL PROBLEMA DE COLOREO DE GRAFOS

Con el fin de mostrar la relevancia del problema de Coloreo de Grafos, se presentan los siguientes problemas, donde el coloreo es central para la solución de estos problemas.

2.2.1 Asignación de horarios

Se necesita programar los exámenes en un Centro Universitario. Si dos asignaturas comparten al menos un alumno, no se pueden programar el mismo día. Teniendo en cuenta esta restricción, se puede construir el grafo $G = (V, E)$ cuyos vértices son las asignaturas.

Las aristas $\{i, j\} \epsilon E$ indican que hay al menos un estudiante matriculado en las asignaturas $\{i, j\} \epsilon V$. Este es el problema típico de planificación de recursos, donde los elementos a planificar son las asignaturas y el recurso que determina la incompatibilidad entre las asignaturas es el día en que se realizaran los correspondientes exámenes.

2.2.2 Asignación de frecuencias

Los grafos se usan para simular la topografía y las interferencias de los transmisores de frecuencias. Los vértices del grafo representan a los transmisores. Dos vértices en el grafo serán adyacentes si las transmisiones son tan altas que interfieren entre sí ocupando el mismo ancho de banda o frecuencia. Las empresas de telecomunicaciones tratan de asignar el mayor número de transmisores en las bandas de frecuencias con unas interferencias entre transmisores por debajo de un límite.

2.2.3 Aplicación en compiladores

Cuando se necesita un registro para un cálculo, pero todos los registros disponibles están siendo utilizados, se debe almacenar (vaciar) el contenido de uno de los registros utilizados en una posición de memoria para dejar libre un registro. La coloración de grafos es una técnica efectiva y sencilla para asignar registros y administrar el vaciado de los registros [9].

2.2.4 Coloración de un mapa

El problema consiste en determinar el menor número de colores que permite colorear un mapa geográfico, de tal manera que dos regiones con una frontera común tengan colores diferentes.

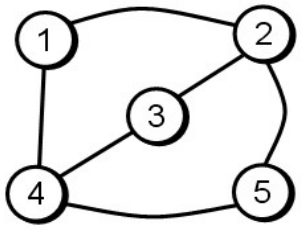
Al mapa se le puede asociar un grafo, que será un grafo planar, cuyos vértices son cada una de las regiones del mapa y habrá una arista entre dos vértices si las regiones correspondientes tienen una frontera común. Una coloración de los vértices es equivalente a una coloración de las regiones. El problema se reduce a determinar el número cromático del grafo asociado al mapa.

2.3 ESTRUCTURA DE DATOS PARA REPRESENTAR GRAFOS

Existen estructuras de datos que pueden utilizarse para representar grafos. La elección de la estructura de datos adecuada depende del tipo de operaciones que se quieran aplicar al conjunto de vértices y aristas del grafo en cuestión. Las representaciones más comunes son las matrices de adyacencia y las listas de adyacencia.

2.3.1 Matrices de adyacencia

Dado un grafo (dígrafo) $G = (V, E)$ con $v = \{1,2, \dots, n\}$, la matriz de adyacencias de G es una matriz A de booleanos con tamaño $n * n$ en la que $A[i][j]$ es cierto si y solo si la arista que una al vértice i con el vértice j está en E (Vea Figura 2.3).



M	1	2	3	4	5
1	0	1	0	1	0
2	1	0	1	0	1
3	0	1	0	1	0
4	1	0	1	0	1
5	0	1	0	1	0

Figura 2.3: Matriz de adyacencia.

Observemos lo siguiente:

- La matriz de adyacencia de un grafo permite representar los ciclos, aunque no las aristas repetidas entre dos vértices.
- Si el grafo no tiene ciclos, entonces puede calcularse el grado de un vértice sumando la fila o columna correspondiente al mismo.
- Dado que la matriz de adyacencia de un grafo es simétrica respecto de la diagonal, la información exceptuando la contenida en la diagonal, aparece dos veces.

En un grafo representado por matrices de adyacencia el tiempo que requiere para acceder a un elemento es independiente del tamaño de V y de E, por tanto, esta puede ser una representación adecuada en las aplicaciones en las que es necesario saber con mucha frecuencia si una determinada arista está presente en el grafo.

La desventaja principal de utilizar una matriz de adyacencias para representar un grafo es que la matriz requiere un espacio $\Omega(n^2)$.

Solo leer o examinar la matriz requerirá tiempo $O(n^2)$, en perjuicio de posibles algoritmos de tiempo $O(n)$ para manipular grafos con $O(n)$ aristas. Una alternativa para evitar esta desventaja es utilizar listas para representar un grafo.

2.3.2 Listas de adyacencia

Dado un grafo $G = (V, E)$ la lista de adyacencias de un vértice i de G es una lista, en un orden cualquiera, de todos los vértices adyacentes a i. Se puede representar G como un vector L en el que $L[i]$ es un puntero a la lista de adyacencias del vértice i.

La cantidad de memoria que requiere esta representación es proporcional a la suma del número de vértices más el número de punteros (que corresponde al número de aristas). Es decir, el coste en memoria es $\Theta(n + m)$ con $n = |V|$ y $m = |E|$.

La desventaja de esta representación es que el determinar si una arista está o no en el grafo puede tomar tiempo $O(n)$ ya que el número máximo de vértices que puede haber en la lista de adyacencias de un vértice dado es n (Vea Figura 2.4).

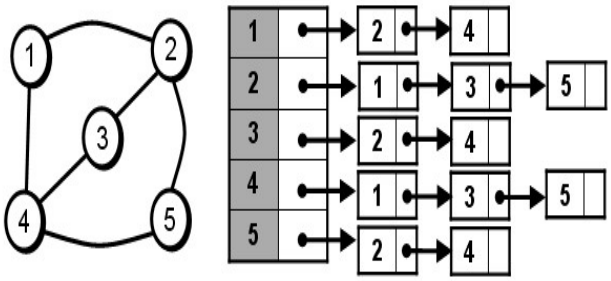


Figura 2.4: Lista de adyacencia.

3 CLASES DE COMPLEJIDAD *P* Y *NP*

Por conveniencia, la teoría de la complejidad está diseñada especialmente para aplicarse a problemas de decisión. Un problema de decisión PD se plantea como una pregunta general, la cual acepta como respuesta sólo una de dos posibilidades, la respuesta 'SI' o la respuesta 'NO'. En forma abstracta, un problema de decisión puede describirse como:

$PD :< D, S >$ Dónde: D - Dominio de valores posibles y $S \subseteq D$ - son las instancias que resuelven el problema. El planteamiento del PD es:

$$\forall x \in D: PD(x) = \begin{cases} Si & si\ x \epsilon S \\ No & en\ otro\ caso \end{cases}$$

En estos términos, un PD puede usarse como medio para reconocer un *lenguaje* S que es un subconjunto de palabras de un alfabeto D. En la práctica, el PD intenta reconocer algún conjunto específico de objetos matemáticos, tales como: fórmulas lógicas verdaderas, gráficas que tienen una cierta propiedad, etc. Pero esto requiere el codificar adecuadamente las estructuras subyacentes de los objetos para cualquier instancia del problema, de tal forma que finalmente, se pueda construir un predicado que aún sólo con respuestas 'SI' o 'NO', obtenga cualquier otra información que se desee.

El problema de decisión que abordamos en este trabajo de tesis, es el de coloración de un grafo.

Instancia: Una gráfica $G = (V, E)$, donde V es el conjunto de nodos o vértices y E es el conjunto de aristas entre los vértices. Una cota $k \in \mathbf{Z}$, con $k \leq |V|$.

Pregunta: ¿Será G k –coloreable? Es decir, ¿Existirá una función $f: V \rightarrow \{1,2,\ldots,k\}$ tal que $f(u) \neq f(v)$ siempre que $\{u, v\} \in E$?

Un algoritmo es un procedimiento general que trabaja paso a paso y en un número finito de estos, resuelve un problema. Un algoritmo resuelve un problema de decisión PD si puede aplicarse a cualquier instancia I del PD y garantiza que siempre produce una solución para esa instancia. Podemos pensar en un algoritmo como un programa de computadora, o como la descripción de la función de transición de una máquina de Turing.

En general, nos interesa encontrar el algoritmo más eficiente que resuelve un problema dado. En el sentido más amplio, la noción de eficiencia tiene que ver con los varios recursos computacionales necesarios para ejecutar el algoritmo. Aunque el recurso dominante es el del tiempo, por tal, normalmente el algoritmo más eficiente que resuelve un problema PD es aquel que se ejecuta más rápidamente de entre todos los algoritmos que resuelven el mismo problema [4].

Los requerimientos de tiempo en la ejecución de un algoritmo se expresan en términos de una sola variable: la longitud o tamaño de las instancias I del problema, que refleja la cantidad de datos de entrada y la magnitud de cada uno de estos datos.

Dada una instancia de un problema de decisión PD con m datos de entrada;

$(d1, \ldots, dm)$ Se define la longitud de la instancia como: $long(I) = \sum_{i=1}^{m} long(d_i)$

Donde $long(di)$ es la longitud de cada uno de los datos de entrada.

Por ejemplo, si se tuviera una instancia del problema del coloreo de un grafo, se requiere codificar el conjunto de nodos $V = \{u_1, \ldots, u_n\}$ y el conjunto de aristas $E = \{a_1, \ldots, a_m\}$ del grafo G de entrada. Se asume dada una instancia específica, que hay n nodos y m aristas, por lo que se requieren $m + n$ etiquetas para diferenciar los nodos y aristas. Suponiendo que $long(x)$ es el número de bytes que se requiere para codificar el dato x en el programa, se tendría que el tamaño de la instancia del problema es:

$$long(I) = \sum_{i=1}^{n} long(u_i) + \sum_{j=1}^{m} long(a_j) + \sum_{j=1}^{n+m} long(etiquetas\ j)$$

La función de complejidad de tiempo de un algoritmo, expresa los requerimientos de tiempo que un determinado modelo de computación necesita al ejecutar el algoritmo para resolver cada posible instancia del problema. Con el fin de clasificar el esfuerzo computacional que se requiere al resolver los problemas de decisión, estos problemas se han clasificado en clases de complejidad. Una misma clase de complejidad contiene a todos los problemas que requieren funciones similares de tiempo para ser resueltos. Denotando a un algoritmo determinista como AD y a un algoritmo no-determinista como AnD, podemos definir las siguientes clases de complejidad:

$$DLOG = \{PD \mid \exists\, AD \text{ que resuelve } PD \text{ usando espacio logarítmico}\}$$

$$P = \{PD \mid \exists\, AD \text{ que resuelve } PD \text{ en tiempo polinomial}\}$$

$$NP = \{PD \mid \exists\, AnD \text{ que resuelve } PD \text{ en tiempo polinomial}\}$$

$$EXP = \{PD \mid \exists\, AD \text{ que resuelve } PD \text{ en tiempo exponencial}\}$$

Las clases P y NP las podemos también identificar de la siguiente manera:

- P Es la clase de problemas que poseen algoritmos deterministas de resolución que tardan tiempo polinomial.
- NP Es la clase de problemas que tienen un algoritmo determinista de resolución que corre en tiempo exponencial, pero para los cuales también existe un algoritmo no determinista que corre en tiempo polinomial.

A los problemas de la clase P, se les reconoce como problemas tratables, puesto que para cualquiera de sus instancias, éstas se resuelven por algoritmos que corren en un tiempo acotado de cómputo. Se dice que un problema no ha sido bien resuelto hasta que es hallado un algoritmo determinista de tiempo polinomial que lo resuelve [4]. Se le asigna a un problema el término de intratable, si este se ha mostrado tan difícil que no se ha encontrado algoritmo determinista con complejidad polinomial en tiempo que lo resuelva, es decir, no se ha hallado algoritmo eficiente que lo resuelva. Entonces una primera clase de complejidad que contiene problemas intratables, es la clase NP.

Todo algoritmo cuya función de complejidad de tiempo, no pueda acotarse por una función polinomial, se dice que es un algoritmo de complejidad exponencial en tiempo, aun cuando debe notarse, que esta definición incluye ciertas funciones de complejidad de tiempo, tales como, $n * log(n)$ las cuales normalmente no son consideradas como funciones exponenciales.

El término intratable, refleja el punto de vista de que los algoritmos de tiempo exponencial no son considerados buenos algoritmos, generalmente tal clase de algoritmos refleja variaciones de búsquedas exhaustivas, mientras que algoritmos de tiempo polinomial generalmente son construidos sólo a través de explotar la estructuras internas e inherentes del problema.

Existen algoritmos de tiempo exponencial que han sido muy útiles en la práctica. La complejidad de tiempo de un algoritmo tal y como la hemos definido, es una medida para el peor de los casos, y el hecho de que un algoritmo tenga complejidad exponencial, significa que existe al menos una instancia del problema, que requiere mucho tiempo, aún y cuando muchas de las instancias del mismo problema requieran en la práctica de mucho menos tiempo que un valor exponencial para ser resueltos. Esto no ha detenido el avance de las investigaciones por buscar algoritmos de tiempo polinomial que resuelvan los mismos problemas. Y de hecho, el gran éxito de tales algoritmos exponenciales, lleva a la sospecha de que hace falta capturar alguna propiedad crucial del problema cuyo refinamiento pueda llevarnos a mejores métodos.

Un área de gran interés es investigar y determinar técnicas generales que permitan diseñar algoritmos deterministas de tiempo polinomial ya sea para los problemas intratables, o para variaciones de éstos.

La habilidad de algoritmos no deterministas, de poder revisar un número exponencial de posibilidades en tiempo polinomial, genera la sospecha, de que este tipo de algoritmos son estrictamente más poderosos que los algoritmos deterministas de complejidad polinomial. Sin embargo, contra todos los esfuerzos realizados, no se ha podido demostrar esta especulación. La pregunta de si $¿P = NP?,$ es uno de los problemas fundamentales de estudio en la ciencia de la computación [4].

3.1 COMPLEJIDAD DEL PROBLEMA DE COLOREO DE GRAFOS

Se dice que una función $G(n)$ es de orden superior a otra función $F(n)$, si existe algún entero n_0 tal que $\forall_n > n_0$, se cumple que $G(n) > k * F(n)$ para un número k positivo. En la tabla 3.1 se comparan valores de distintas expresiones en función de n. Puede observarse que las expresiones polinomiales crecen a un ritmo mucho menor que las funciones exponenciales, o la función factorial; Se dice así mismo que una función $f(n)$ es de $O(g(n))$ si el cociente $f(n)/g(n)$ puede acotarse por un valor constante para todo n arbitrariamente grande.

En la práctica, se considera que un algoritmo es eficiente cuando es capaz de encontrar una solución exacta del problema y cuya complejidad en tiempo está acotada superiormente por una función polinomial sobre el argumento n – longitud de la entrada. En ocasiones, esto no es posible; en estos casos se considera entonces que un algoritmo de resolución es aceptable si es capaz de dar una aproximación de la solución óptima en un intervalo de tiempo polinomial, o bien, si en la mayoría de los casos es capaz de encontrar la solución óptima en un tiempo polinomial de ejecución.

Función	$n = 2$	$n = 8$	$n = 128$	$n = 1024$
n	2	2^3	2^7	2^{10}
$n \log_2 n$	2	$3 \cdot 2^3$	$7 \cdot 2^7$	$10 \cdot 2^{10}$
n^2	2^2	2^6	2^{14}	2^{20}
n^3	2^3	2^9	2^{21}	2^{30}
2^n	2^2	2^8	2^{128}	2^{1024}
$n!$	2	$4.9 \cdot 2^{13}$	$4.5 \cdot 2^{714}$	2^{8769}

Tabla 3.1: Orden de crecimiento de algunas funciones básicas.

Sea un grafo G de orden n y con número cromático $x(G) = k$ tal que quiere obtenerse un coloreo óptimo de G. Un proceso de búsqueda exhaustiva de un coloreo óptimo de G requiere, en el mejor de los casos, y suponiendo el número cromático desconocido, un número de iteraciones igual a:

$$Niter = \sum_{i=2}^{k-1} i^n$$

En este caso, el algoritmo comenzaría explorando todos los posibles $2-$coloreados del grafo y comprobando en cada caso si el coloreo cumple las restricciones (dos vértices adyacentes deben tener asignado distintos colores); A continuación, si no se hubiera encontrado ningún 2- coloreado correcto, el algoritmo repetirá la operación para tres colores y así sucesivamente.

En lo que respecta al problema del coloreo de un grafo, cuando un grafo es $2-$coloreable, tal propiedad puede ser reconocida en un tiempo polinomial, esto significa que determinar si un grafo es o no 2-coloreable, es un problema de la clase de complejidad P. De hecho, una forma de revisar que un grafo G es $2-$coloreable es revisando que G sea un grafo bipartito, y esto puede hacerse al verificar que todo ciclo que aparece en G sea de longitud par.

Al incrementar el número de colores de 2 a3, el problema del $3-$coloreo de un grafo es conocido como un problema que se encuentra en la clase de complejidad NP. Y en general, el problema del $k-coloreo$ con $k\geq 3$ es un problema de la clase NP. Esto significa, que aún y con todo el esfuerzo que se ha dedicado al diseño de algoritmos que resuelven el problema del coloreo de grafos, estas propuestas algorítmicas no son eficientes, considerando como entrada grafos generales donde se requieran más de dos colores para ser coloreados.

4 TÉCNICAS DE RECORRIDOS EN GRAFOS

Un recorrido de un grafo es una manera sistemática de explorar sus vértices de manera completa, siguiendo la misma estructura del grafo.

Los recorridos dan lugar a esquemas para el tratamiento de grafos. En un recorrido puede aplicarse una determinada operación en cada visita a los elementos (nodo o arista) del grafo.

4.1 RECORRIDO A LO PROFUNDO

El recorrido a lo profundo de un grafo, comienza en el vértice inicial (vértice con índice 1), el cual se marca como vértice activo, hasta que todos los vértices hayan sido visitados, en cada paso se avanza al vecino con el menor índice siempre que se pueda, pasando a ser este el vértice activo.

Cuando todos los vecinos al vértice activo hayan sido visitados, se retrocede al vértice x desde el que se alcanzó el último vértice activo, y se prosigue siendo ahora x el vértice activo. A continuación se muestra el pseudocódigo de la Búsqueda primero en profundidad DFS (Depth-First Search):

$dfs\ (v)$
Entrada: $v \in V$
Salida: T_G
1: $Vista\ v$
2: $V_{T_G} = V_{T_G} \cup \{v\}$
3: ***para*** $w \in N(v)$ ***hacer***
4: ***si*** $w\ no\ Visitado$ ***entonces***
5: $E_{T_G} = E_{T_G} \cup \{(v, w)\}$
6: $dfs\ (w)$
7: ***si no***
8: $E'_{T_G} = E'_{T_G} \cup \{(v, w)\}$
9: fin si
10: fin para
11: devolver

Este procedimiento tiene una complejidad de tiempo de $O(m+n)$ donde n y m son el número de nodos y el número de aristas de entrada del grafo G.

Tomemos como ejemplo el siguiente grafo de la figura 4.1 para mostrar cómo es el recorrido en profundidad:

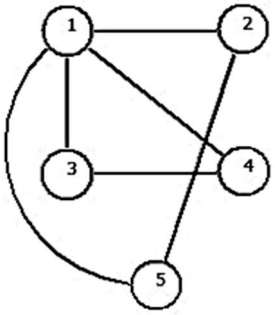


Figura 4.1: Grafo G.

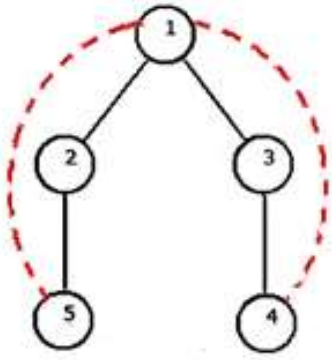


Figura 4.2: Recorrido a profundidad de grafo G.

4.2 RECORRIDO EN AMPLITUD

En este algoritmo también se utiliza la estrategia de marcar los nodos como visitados para detectar la culminación del recorrido, pero los nodos se recorren de una manera ligeramente distinta.

Se selecciona de cualquier nodo como punto de partida (usualmente el primer nodo), se marcan todos los nodos del grafo como *no visitados*. El nodo inicial se marca como visitado, y luego se visitan todos los nodos adyacentes a este, al finalizar este proceso se busca visitar nodos más lejanos visitando los nodos adyacentes a los nodos adyacentes del nodo inicial.

Este algoritmo puede crear menos ambientes recursivos que el anterior porque visita más nodos en un mismo ambiente, pero esto depende de cómo este construido el grafo. El algoritmo se conoce como el algoritmo de BFS (Breadth- first Search).

```
bfs( v )
Entrada: v ∈ V
Salida: T_G
 1: QUEUE C = ∅
 2: C.Inserta( v )
 3: mientras not( C.Vacia() ) hacer
 4:    v = C.saca()
 5:    V_TG = V_TG ∪ {v}
 6:    Visita v
 7:    para w ∈ N(v) hacer
 8:       si w no Visitado entonces
 9:          E_TG = E_TG ∪ {(v, w)}
10:          C.Inserta( w )
11:       si no
12:          E'_TG = E'_TG ∪ {(v, w)}
13:       fin si
14:    fin para
15: fin mientras
16: devolver
```

Este algoritmo tiene exactamente el mismo orden en tiempo de ejecución del algoritmo de recorrido en profundidad y también se puede obtener el conjunto de aristas de cubrimiento mínimo del grafo (Vea Figura 4.2.1).

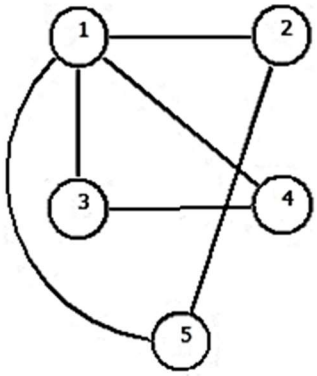


Figura 4.2.1: Grafo G.

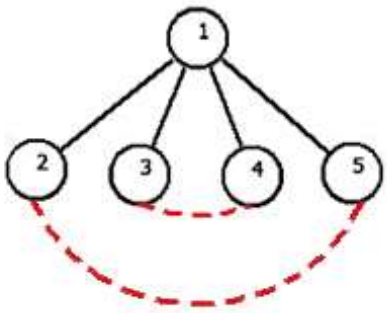


Figura 4.2.2: Recorrido en amplitud de un grafo G.

Una diferencia entre la DFS y la BFS es que esta última necesita de una estructura auxiliar, que por lo general es una cola, para el almacenamiento de las aristas que se van a visitar durante el recorrido.

4.3 ÁRBOL GENERADOR Y CICLOS BÁSICOS

El resultado de la búsqueda a lo profundo genera un árbol que denotaremos con T_G este se forma por los nodos y las aristas de árbol cuando se aplica el algoritmo dfs sobre el grafo G.

Dado un grafo no dirigido conexo $G = (V, E)$, aplicando una búsqueda en profundidad para recorrer G se produce T_G donde $V(T_G) = V(G)$.

Las aristas de T_G se llaman aristas de árbol, mientras que las aristas $E(G) - E(T_G)$ se llamaran aristas de retroceso.

Sea T el árbol generador de un grafo conexo $G = (V, E)$ cualquier arista $\{v, w\}$ que pertenece a $E(G - T)$ la cual se agregue a T formará un ciclo único denotado como Cv, w donde $v\ y\ w$ se conectan por un camino único que denotaremos como $Pv, w.$ en $T.$ De esta forma $Pv, w + \{v, w\}$ forman el ciclo Cv, w el cual llamaremos ciclo básico de $v\ a\ w$, con respecto al árbol generador T. Entonces, sea $e \in E(G) - E(TG)$ una arista de retroceso, la unión de la ruta en T_G entre los extremos finales de e con la misma arista e forma un ciclo simple, un ciclo de este tipo se llama ciclo básico (o fundamental) de G con respecto a TG (Figura 4.3). Cada arista de retroceso tiene el máximo trayecto contenido en el ciclo básico del que forma parte

Llamamos nodo final de un ciclo, al nodo donde se encontró una arista de retroceso durante la búsqueda en profundidad.

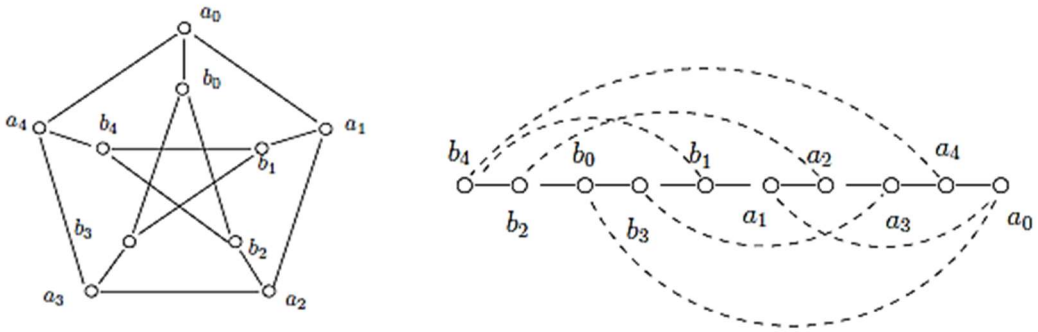


Figura 4.3: Grafo de Petersen con su respectivo árbol generador.

A partir de los ciclos básicos del árbol T_G podemos generar cualquier ciclo que aparece en G, de hecho, cualquier ciclo que se pueda formar en G es un ciclo que se forma al realizar uniones simétricas entre ciclos básicos.

Las aristas de retroceso que sean encontradas durante la búsqueda a lo profundo darán como resultado un ciclo básico (Figura 4.4).

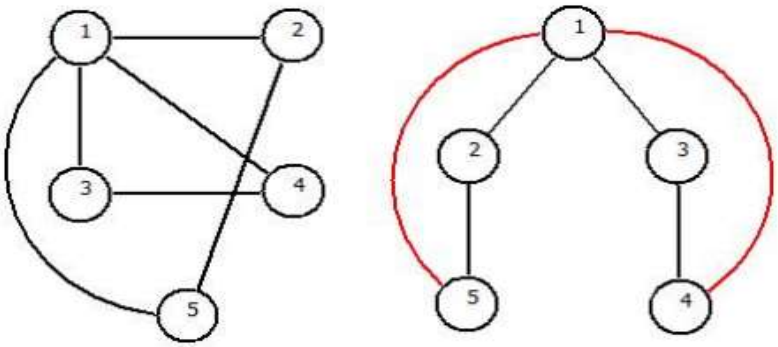


Figura 4.4: Ciclo Básico.

4.4 CICLOS INTERSECTADOS Y COMPUESTOS

Sea $C = \{C1, C2, \ldots, Ck\}$ el conjunto de ciclos básicos encontrados durante la búsqueda en profundidad en G, dado un par de ciclos básicos $Ci\ y\ Cj$ del grafo G, si Ci y Cj comparten al menos una aristas se dice que son *ciclos intersectados*, de los contrario se les denomina *ciclos independientes*.

Si dos ciclos comparten una sola arista, se les denomina *ciclos adyacentes;* Se dice que un ciclo básico Ci es independiente de Cj, si y sólo si $Ci\ y\ Cj\ \in\ C$ son disjuntos para cada $j <> i$.

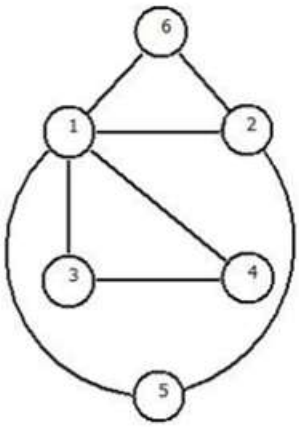


Figura 4.5: Ciclos intersectados e independientes.

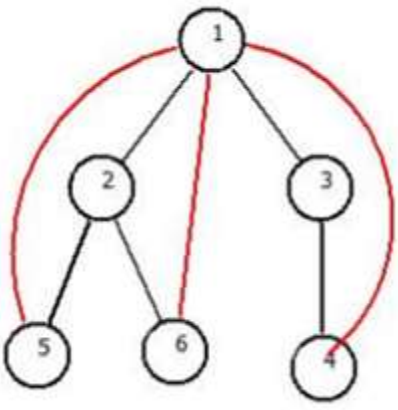


Figura 4.6: Búsqueda en profundidad.

Las aristas de retroceso son marcadas en color rojo, por consiguiente existe 3 ciclos:

C_1={v_1, v_2, v_5, v_1}

C_2={v_1, v_2, v_6, v_1}

C_3={v_1, v_3, v_4, v_1}

El ciclo $C_1\ y\ C_3$, así como $C_2\ y\ C_3$ son independientes, pero el ciclo $C_1\ y\ C_2$ son ciclos intersectados.

5 DISEÑO DEL ALGORITMO

En este capítulo se examinaran las condiciones y los procedimientos del coloreo de grafos con $x(G) = 2$ y $x(G) = 3$, utilizando ejemplos para mostrar la importancia del análisis de los ciclos dentro del coloreo de un grafo.

De igual forma se propondrán los algoritmos heurísticos, su análisis y algunos ejemplos gráficos.

5.1 PROCEDIMIENTOS BÁSICOS PARA EL COLOREO DE GRAFOS

Como ya se mencionó anteriormente, con la búsqueda en profundidad se obtendrán los ciclos básicos que conforman el grafo, y en base al conjunto de ciclos básicos, se propone una heurística para el coloreo dependiendo de cuando se tienen ciclos impares intersectados, o cuando ya no se tiene este tipo de ciclos.

Primero, presentamos el caso del *k*-coloreo que se resuelve eficientemente, esto es, cuando existe un 2-coloreo y cuando no hay ciclos intersectados con ciclos impares.

5.2 PROCEDIMIENTO PARA UN DOS COLOREO

El coloreo de un grafo se puede calcular en tiempo polinomial cuando $x(G) = 2$.

Cualquier grafo con un ciclo de longitud impar es claramente no 2-coloreable (Vea Figura 5.1).

Para colorear un grafo con dos colores, dado un color asignado a un vértice v, debe colorearse el resto del grafo asignando el otro color a cada vértice adyacente a v. Este proceso es equivalente a alternar los dos colores para los nodos en cada nivel de un grafo dfs, y durante el retorno de las llamadas recursivas, verificar si no hay otra arista que conecte dos nodos del mismo color, ya que en este caso, tal arista es evidente de un ciclo de longitud impar (Vea Figura 5.1).

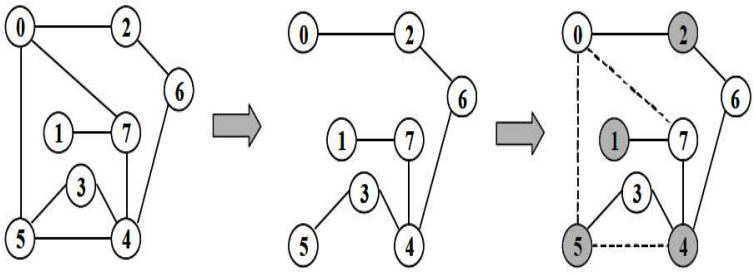


Figura 5.1: Grafo que no es dos coloreable.

5.3 Condiciones necesarias para el 3-Coloreo

Existen topologías de grafos que han sido identificadas por ser 2-coloreables o 3-coloreables

1. Si T_G tiene ciclos impares, pero todos ellos son independientes de otros ciclos impares entonces T_G es 3 –coloreable. Podemos colorear T_G por niveles, pero usando el tercer color para pintar un nodo final de cada ciclo impar, (Figura 5.2) muestra un grafo con dos ciclos impares que no comparten ninguna arista (independientes), se puede colorear con 3 colores como se muestra, asignando el tercer color al nodo final del ciclo.
2.

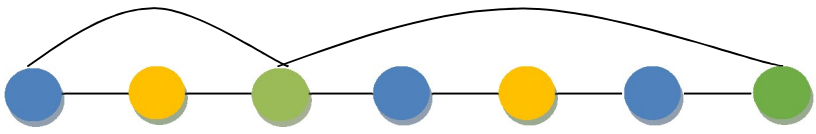


Figura 5.2: Grafo con dos ciclos impares.

2. Si un ciclo impar en G se cruza solo con ciclos pares o a lo mucho hay dos ciclos intersectados entonces T_G es 3-coloreable. Como el caso anterior, T_G es dos coloreado por niveles, y el tercer color se utiliza para colorear el nodo final que es común a cada par de ciclos intersectados impares (Figura 5.3). Podemos ver un ejemplo en que se tiene un ciclo impar intersectado con un ciclo par de tal forma que se colorea con verde el nodo final común y el resto es dos coloreado.

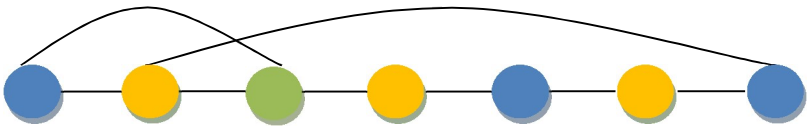


Figura 5.3: Ciclo impar intersectado con un ciclo par.

3. Si todo el conjunto de los ciclos impares intersectados de G puede ser dispuesto como ciclos planos embebidos uno dentro de otro entonces G es 3 –coloreable. Coloreamos cualquier conjunto de ciclos planos embebidos desde el ciclo más interno al ciclo más externo (Figura 5.4). Cuando empezamos a colorear un nuevo ciclo, se utiliza un color diferente al de los nodos vecinos (se tendrá un máximo de dos nodos vecinos), ya que consideramos que sólo los nodos en los ciclos internos han sido ya coloreados. En este caso, podemos considerar a T_G como un ejemplo de un grafo serial paralelo, el cual es conocido por ser coloreable en tiempo polinomial [4].

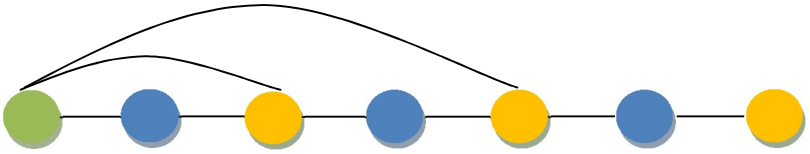


Figura 5.4: Ejemplo de ciclos impares embebidos.

5.4 GRAFOS QUE NO SON 3-COLOREABLES

Si un ciclo par en G se cruza con dos ciclos impares intersectados de forma que compartan un nodo adyacente a sus extremos, entonces el grafo no es 3 coloreable (Figura 5.5).

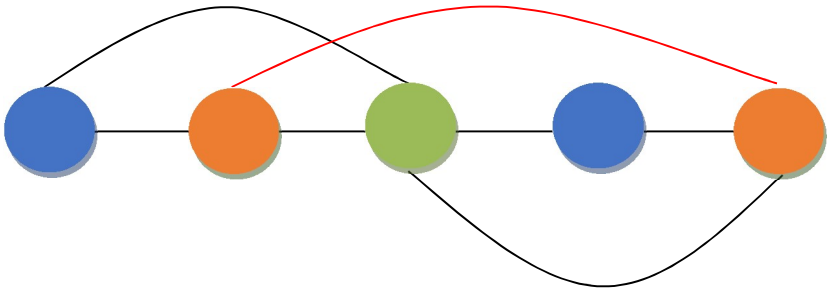


Figura 5.5: Grafo que no es 3 coloreable.

En caso de que un grafo no sea regular, primero obtenemos el árbol que nos devuelve el recorrido en profundidad, después coloreamos los vértices que se encuentran en el último nivel del árbol (los últimos descendientes), y el padre de cada sub-árbol se colorea con el color distinto a los de sus hijos. Este procedimiento es un coloreo que sigue el orden: post-orden, es decir, primero se colorean los hijos y después el padre.

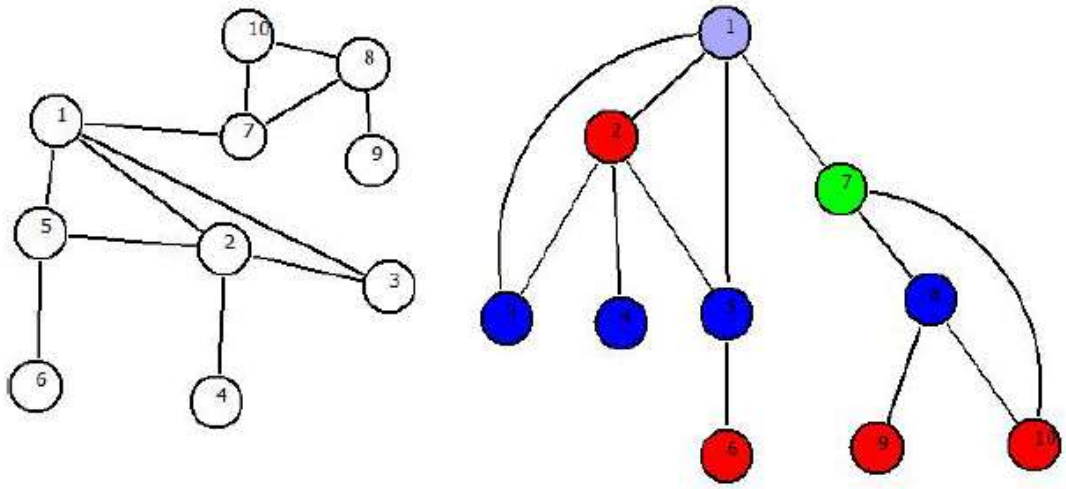


Figura 5.6: Grafo 4 coloreable (no óptimo).

5.5 PROPUESTA HEURÍSTICA PARA UN 3 COLOREO

A continuación se mostrará y analizará la heurística que se está proponiendo para realizar un tres coloreo.

Algoritmo 3-Coloreo

1.- Dado un grafo conexo $G = (V, E)$, sea $n = |V|$, $m = |E|$ el número de nodos y aristas, respectivamente. Se realiza el recorrido en profundidad sobre G. Durante el recorrido es calculado para cada nodo, el número de ciclos en el que el vértice esta contenido.

2.- Se inicializa C_3 que es un conjunto de nodos, los cuales formarán la clase de un mismo color (al inicio el tercer color).

3.-Seleccionamos un $x \in V$, que aparezca en el número máximo de ciclos.

Se forma el conjunto de vecinos de $x \in V$, denotado por $N(x)$.

4.- Inicializamos los conjuntos:

$N_1 = N(x)$;

$N_t = N(N_1) - N_1$;

$C_3 = C_3 \; U\{x\}$; /* Adicionar nodo a la clase de color */

5.- Eliminamos al nodo x del grafo G y se añade éste al conjunto C_3.

6.- Mientras el grafo G no sea bipartito, esto significa que aún existen nodos para C_3, por lo tanto, no se pasaría a un $2-$coloreo:

Si en $G/N_t U N_1$ aún hay ciclos impares, entonces:

Selecciona un $x \epsilon N_t$ que aparece en el mayor número de ciclos impares

y ahora $C_3 = C_3 \; U \{x\}$, eliminándolo de G.

Si ya no hay ciclos modificamos

$$N_1 = N_1 U N_t$$

$$N_t = N(N_1) - N_1$$

Y se regresa a verificar si G es bipartito G (paso 6).

En otro caso, se regresa C_3, que representa el conjunto que será coloreado por el tercer color.

Al ser G bipartito, se sabe que entonces puede ser coloreado por los dos primeros colores.

Pseudocódigo del algoritmo 3-Coloreo

Entrada: G (grafo no dirigido)

Salida: C_3 (conjunto de nodos del tercer color)

$G = dfs(G);$ /*Ordena ciclos de G vía búsqueda a lo profundo*/

$C_3 = 0;$ /*Inicializa el tercer color*/

Select $x \in V(G)$ /*Selecciona un x que pertenece a mas ciclos*/

/*Forma vecinos de x y vecinos de los vecinos de x*/

$N_1 = N(x);$

$N_t = N(N_1) - N_1;$

$C_3 = C_3 U \{x\};$

$G = G - \{x\};$ /*1er. Elemento para tercer color*/

while ($G \neq$bipartito) do /*hay nodos para C3*/

{ if ($G/N_t \mathrm{U}\, N_1$ has odd cycles) then

/*Selecciona un $x \epsilon N_t$ que aparece en el mayor número de ciclos impares*/

$C_3 = C_3\, U\, \{x\};$

$G = G - \{x\};$ /*Siguiente elemento para el tercer color*/

else

$N_1 = N_1 U N_t;$

$N_t = N(N_1) - N_1;$ /*Siguiente nivel de vecinos de x*/

end if

} end while

Return C_3

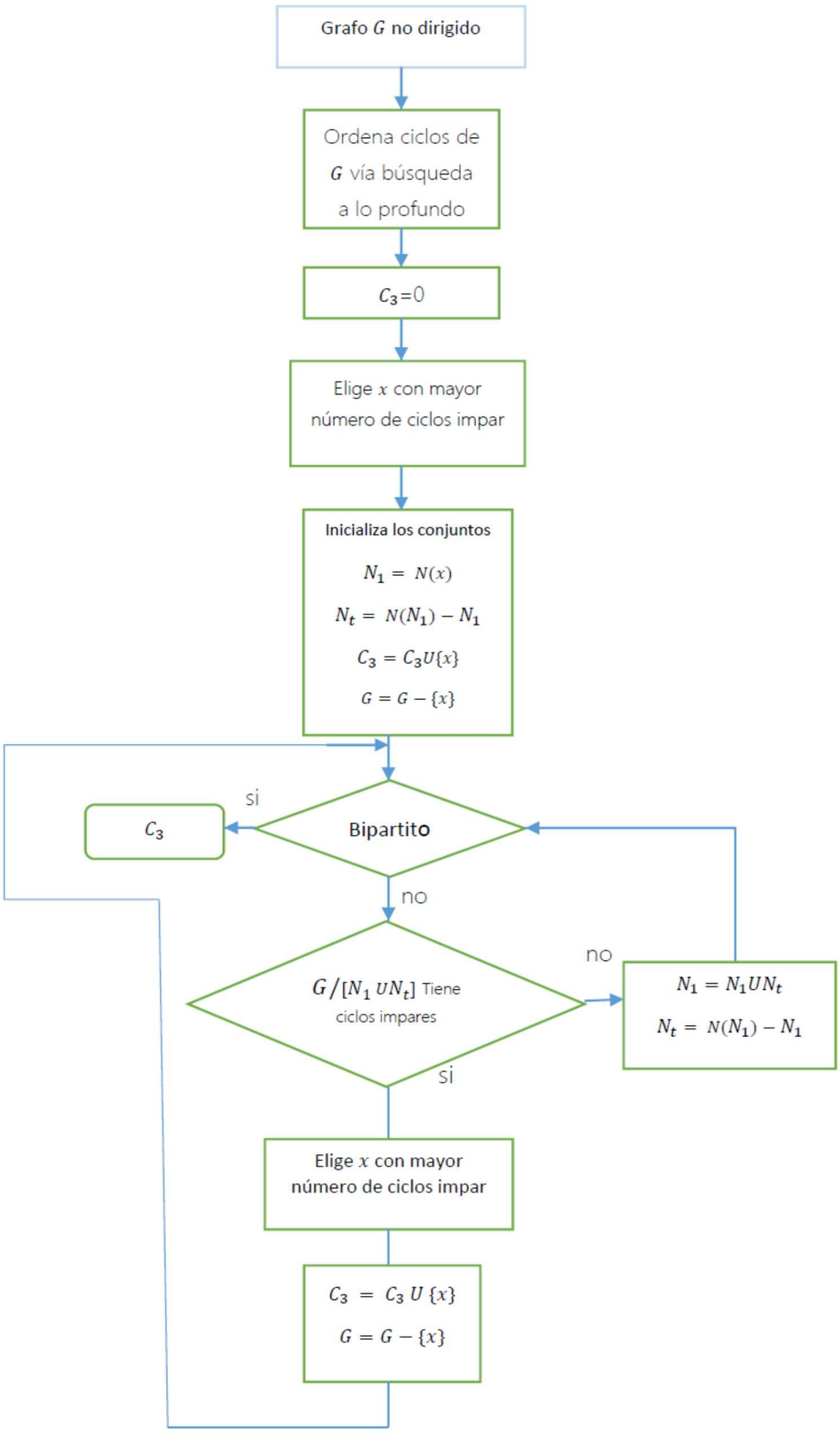


Figura 5.7: Representación gráfica del algoritmo.

En cierta forma, podemos considerar nuestra propuesta algorítmica, como una *búsqueda local* por el nodo x a ser coloreado, sobre vecindades de segundo orden. Esto es, una vez que se decidió colorear al nodo x, se salta sobre sus nodos vecinos $N(x)$, y se busca seleccionar entre los vecinos de los vecinos $N(N(x))$, que nodo y aparece en un número máximo de ciclos intersectados y de longitud impar, y además que aparte de no estar coloreado, no sea parte de $N(x)$.

El seleccionar el nodo y que aparece en un número máximo de ciclos intersectados y de longitud impar, inmediatamente después de que se encuentra, nos determina una heurística ávida, que es caracterizable por ser una propuesta eficiente, como veremos en capítulos posteriores, pero por no garantizar que siempre encontrará el coloreo óptimo.

Nótese que las vecindades $N(x)$ y $N(N(x))$ se van actualizando conforme se selecciona el nodo x, y de acuerdo al resultado de la búsqueda en profundidad sobre $G - \{x\}$.

5.6 Ejemplos de aplicación de la heurística

En esta sección mostraremos de manera gráfica la operación de nuestra heurística. Consideraremos grafos que si son 3 –coloreables, con la finalidad de mostrar todos los pasos que realiza nuestra propuesta.

Ejemplo 1

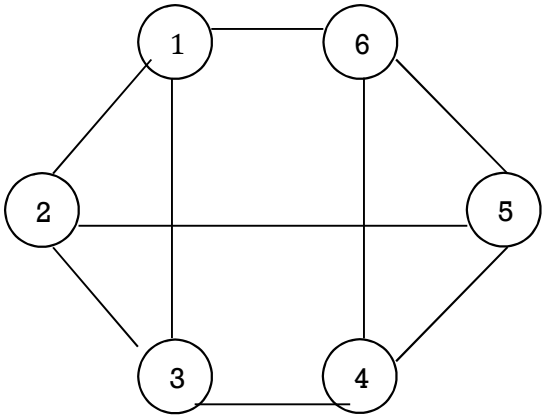


Figura 5.8: Grafo G.

$G = G - \{2\};$ (Figura 5.9)

Ciclo 1:$\{1,3,4,5,6\}$
Ciclo 2: $\{4,5,6\}$

x=4;

$C3 = \{2,4\}$

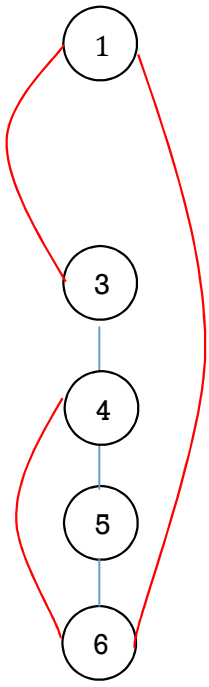


Figura 5.9: G-{2}.

$G = G - \{4\};$ (Figura 5.10)

El subgrafo ya es Bipartito.

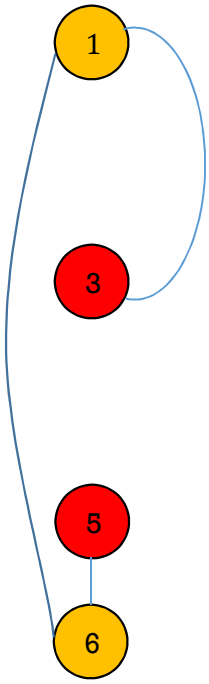


Figura 5.10:$G - \{4\}$.

EJEMPLO 2:

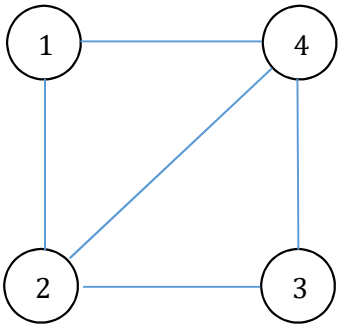


Figura 5.11: Ejemplo 3.

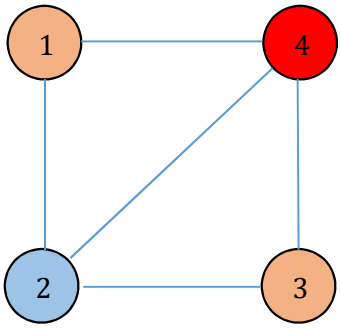


Figura 5.12: Ejemplo 3.

G=dfs(G);

Ciclo 1 : {*2,3,4*};

Ciclo 2 :{*1,2,3,4*};

x=2;

N_1 = {*1,3,4*};

N_t ={};

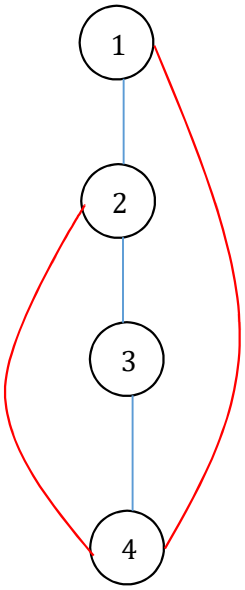


Figura 5.13: dfs(G).

El subgrafo ya es Bipartito.

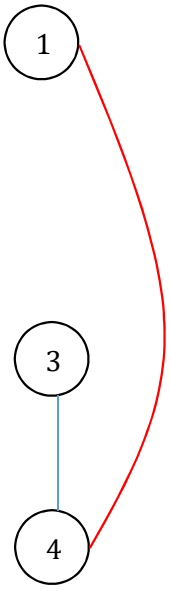


Figura 5.14: dfs(G) − {2}.

5.7 Segunda propuesta heurística para el Coloreo de grafos

En esta sección se mostrará una segunda versión de nuestra propuesta, con esto se busca no limitarse sólo al caso de un 3 coloreo, por lo que la llamaremos una *heurística para un k-coloreo*.

Algoritmo:

1.- Mientras ($G \neq \emptyset$)

2.- Dado un grafo conexo $G = (V, E)$, sea $n = |V|$, $m = |E|$ el número de nodos y aristas, respectivamente. Se realiza un recorrido en profundidad sobre G; Durante el recorrido es calculado para cada nodo, el número de ciclos en el que el vértice esta contenido.

3.- Se inicializa C_i que es un conjunto de nodos, los cuales formaran la clase del color i.

4.-Seleccionamos un $x \in V$, que aparezca en un número máximo de ciclos.

Se forma el conjunto de vecinos de $x \in V$, denotado por $N(x)$.

5.- Inicializamos los conjuntos:

$N_1 = N(x)$;

$N_t = N(N_1) - N_1$;

$C_i = C_i U\{x\}$; /* Adicionar nodo a la clase de color *i*/

6.- Eliminamos al nodo x del grafo G, y se añade al conjuntoC_i.

7.- Mientras que el grafo G/N_tUN_1no sea bipartito y $Nt \neq \emptyset$, significa que aún existen nodos para C_i , por lo tanto, no se pasaría a un 2-coloreo:

Si en G/N_tUN_1 aún hay ciclos impares, entonces:

7 a) Se selecciona un $x \epsilon N_t$ que aparece en el número máximo de ciclos impares, y $C_i = C_i\, U\, \{x\}$, eliminando x de $V(G)$.

En otro caso (ya no hay ciclos impares) entonces

$$N_1 = N_1 U N_t$$

$N_t = N(N_1) - N_1$ - se pasa al siguiente nivel de los vecinos

Y se regresa a verificar si G/N_tUN_1 es bipartito (paso 7).

8.- Si G no es bipartito y $G \neq \emptyset$ regresa al paso 1.

En otro caso, se regresa C_i, que representa un conjuntos de nodos, cada uno de ellos tendrán un mismo color i.

Al ser G bipartito, se sabe que entonces puede ser coloreado mediante dos colores.

5.8 EJEMPLOS DE LA HEURÍSTICA DEL COLOREO DE GRAFOS

A continuación, se mostrarán unos ejemplos de cómo trabaja nuestra propuesta algorítmica, y además, se trabajará con instancias de grafos con un número cromático mayor a tres.

Ejemplo 3

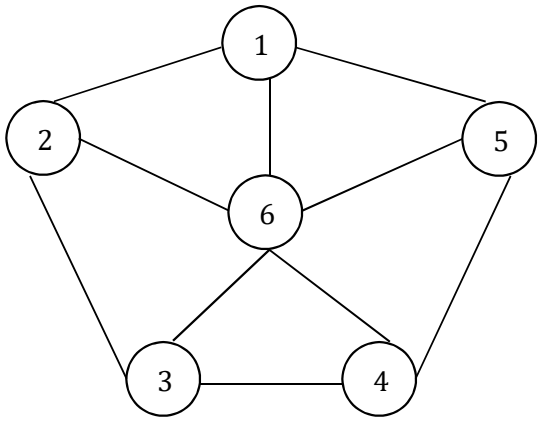


Figura 5.15: Grafo G.

$\boldsymbol{G = dfs(G);}$ (Figura 5.16)

C_i =0

Ciclo 1: {1,2,3,4,5,6}

Ciclo 2: {1,2,3,4,5}

Ciclo 3: {4,5,6}

Ciclo 4: {3,4,5,6}

Ciclo 5: {2,3,4,5,6}

x=4;

$N1 = \{3,5,6\}$
$Nt = \{2,1\}$
$C1 = \{4\}$

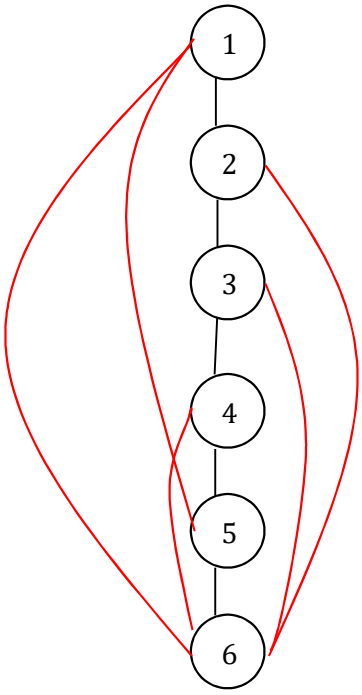


Figura 5.16: $G = dfs(G)$.

$\boldsymbol{G = G - \{4\};}$ (Figura 5.17)

Ciclo 1: {1,2,3,6,5}

Ciclo 2: {2,3,6}

Ciclo 3: {1,2,3,6}

$N1 = \{3,5,6\}$
$Nt = \{2,1\}$

x=2;

$C1 = \{4,2\}$

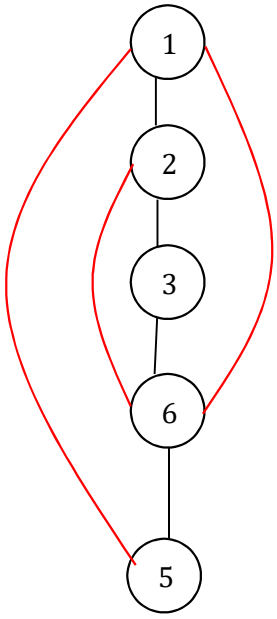


Figura 5.17: $G = dfs(G)$.

$\boldsymbol{G = G - \{2\}}$; (Figura 5.18)

Ciclo 1: {1,5,6}

$N1 = \{1,5,6,3\}$
$Nt = \emptyset$

$C2 = \emptyset$

$X = 6$

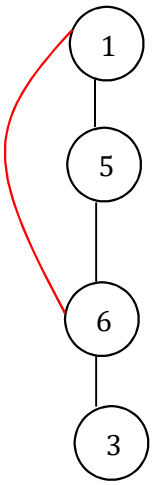


Figura 5.18: $G = dfs(G)$.

$\boldsymbol{G = G - \{6\}}$; (Figura 5.19)

$N1 = \{1,5,3\}$
$Nt = \emptyset$

$C2 = \{6\}$

El subgrafo ya es Bipartito.

Figura 5.19: $G = dfs(G)$.

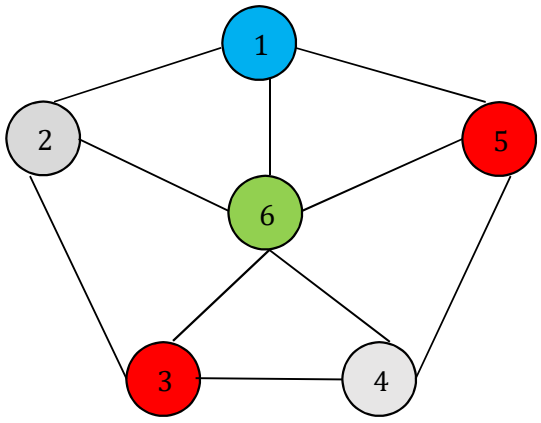


Figura 5.20: Grafo G coloreado.

Ejemplo 4

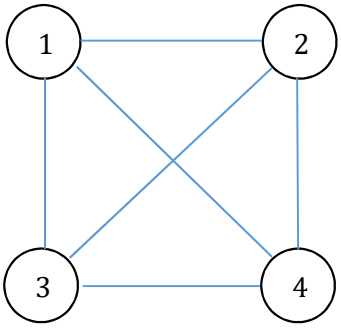


Figura 5.21: Grafo G.

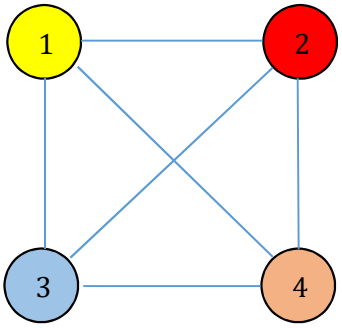


Figura 5.22: Grafo G coloreado.

$C1 = \{1,2,3\}$
$C2 = \{1,2,3,4\}$
$C3 = \{2,3,4\}$

$X = 2$
$N1 = \{1,3,4\}$
$Nt\{\ \emptyset\ \}$

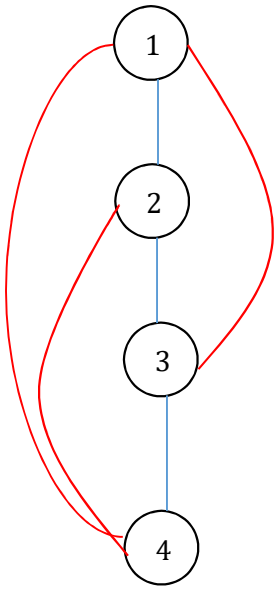


Figura 5.23: dfs(G).

$C1 = \{1,3,4\}$

$X = 1$
$N1 = \{1,3,4\}$
$Nt\{\ \emptyset\ \}$
X=1

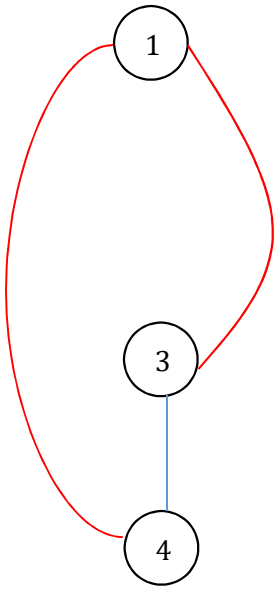


Figura 5.24: Grafo $G - 2$.

El subgrafo ya es Bipartito.

Figura 5.25: Grafo $G-1$.

5.9 ANÁLISIS DE LA COMPLEJIDAD EN TIEMPO

En esta sección presentamos el análisis de la complejidad en tiempo de nuestra propuesta.

Consideremos que la entrada a nuestro algoritmo es un grafo $G(V,E)$, con $n=|V|$ y $m=|E|$.

Es conocido que la búsqueda a lo profundo ($dfs(G)$) tiene una complejidad de orden $O(m)$.

El cálculo de la primera y segunda vecindad $N_1(x)$ y $N_t(x)$ para cualquier nodo $x \in V(G)$ se puede calcular en tiempo $O(n)\ y\ O(n^2)$, respectivamente.

La construcción del grafo $[{}^{G}/_{N_1(x)}\ U\ N_t(x)\ ]$ que es el grafo inducido por la primera y segunda vecindad de x, se construye en tiempo lineal sobre m, esto es, en tiempo de orden $O(m)$.

Al hacer el recorrido en profundo sobre $[\ {}^{G}/_{N_1(x)}\ U\ N_t(x)\ ]$ se puede asociar a cada uno de sus nodos x un contador x_n que cuente en cuantos ciclos de longitud impar x participa. Así, la verificación de que hay ciclos de longitud impar en este grafo inducido, no requiere de más de $O(m*n)$ operaciones básicas, y al mismo tiempo se calculan los contadores $x_n$para todo nodo del sub-grafo.

Los contadores anteriormente calculados son cruciales para que al realizar el paso 7 a) en nuestro algoritmo anterior, este no requiera de más del orden de $O(m*n)$. De hecho, este es el proceso más costoso en todo nuestro algoritmo, por lo que le llamaremos el *núcleo central* de nuestro proceso de coloreo.

En el caso de la primera versión: *algoritmo heurístico de 3 coloreo*, sólo se tiene un ciclo donde en cada iteración se va a colorear un nodo, hasta que no más nodos pueden colorearse con el mismo tercer color o bien, ya se formó un grafo bipartito. Todos los demás procesos analizados quedan dentro de este ciclo, por lo que la complejidad en tiempo de la primera versión es de orden $O(n\ *\ m*n) = O(m*n^2)$, Lo que resulta en una heurística de complejidad polinomial en tiempo.

Para el caso de nuestra segunda propuesta: *una heurística para un k-coloreo*, se realizan tantas iteraciones sobre el *núcleo central* de nuestro programa como *k* colores tenga el grafo, así que tendríamos a lo más k repeticiones de la complejidad de nuestro proceso de 3-coloreo, esto es del orden $O(k*m*n^2)$ operaciones.

Como *k* se considera una constante que no depende de n ni de m (que definen el tamaño de la instancia de entrada) entonces podemos considerar su complejidad en tiempo como: $O(m*n^2)$. Salvo que estemos en el caso de un grafo completo, en donde *k* es exactamente n, y sólo para estos casos, se tiene que la complejidad de nuestra heurística para el coloreo, en el peor de los casos es de orden $O(m*n^3)$.

6 EXPERIMENTACIÓN

A continuación se mostrará y detallará el funcionamiento de la aplicación que ha sido elaborada, con el fin de mostrar la ejecución de la implementación de nuestro algoritmo.

6.1 ANÁLISIS DE LA APLICACIÓN

Se ha implementado el algoritmo heurístico propuesto en esta tesis en el lenguaje $C\#$, esta aplicación se encarga de validar y mostrar de forma gráfica los resultados de la heurística.

La aplicación permite dibujar un grafo, este es almacenado en una estructura (en este caso se utilizara una lista de adyacencia), una vez obtenidos los datos, se aplica el algoritmo heurístico propuesto, y se muestra gráficamente el resultado de nuestra heurística.

Ingresando los grafos estudiados en la sección 5.6 que son 3-coloreables, encontramos que la aplicación nos muestra el grafo coloreado como se analizó anteriormente, comprobando así, que el algoritmo heurístico resuelve satisfactoriamente el 3-coloreo para las instancias de prueba. No se puede concluir que la heurística colorea todo tipo de grafos 3 coloreables, ya que este algoritmo no es exacto y se trata de una heurística de complejidad polinomial para un problema cásico de la clase de complejidad NP-completo.

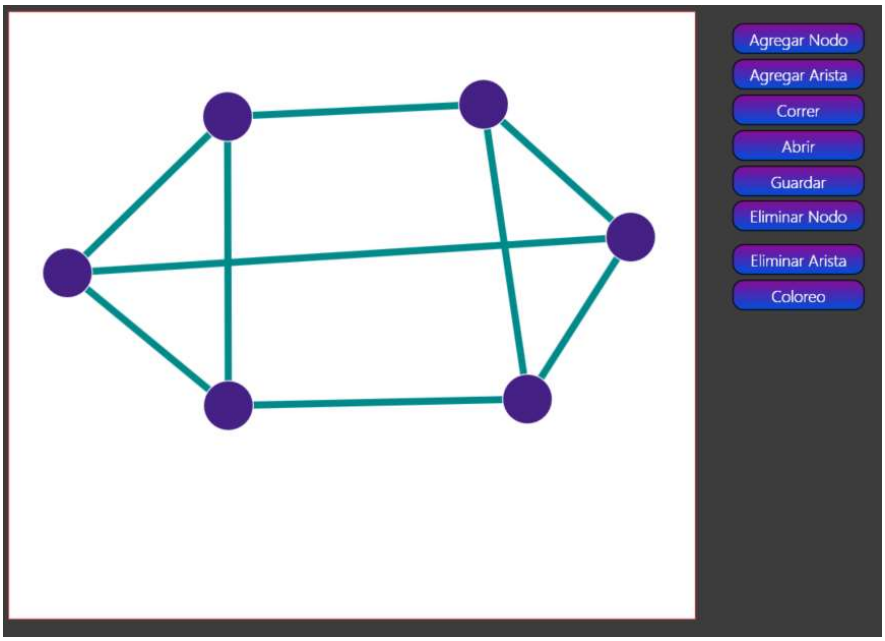


Figura 6.1: Grafo 3 coloreable (ejemplo 1, sección 5.6).

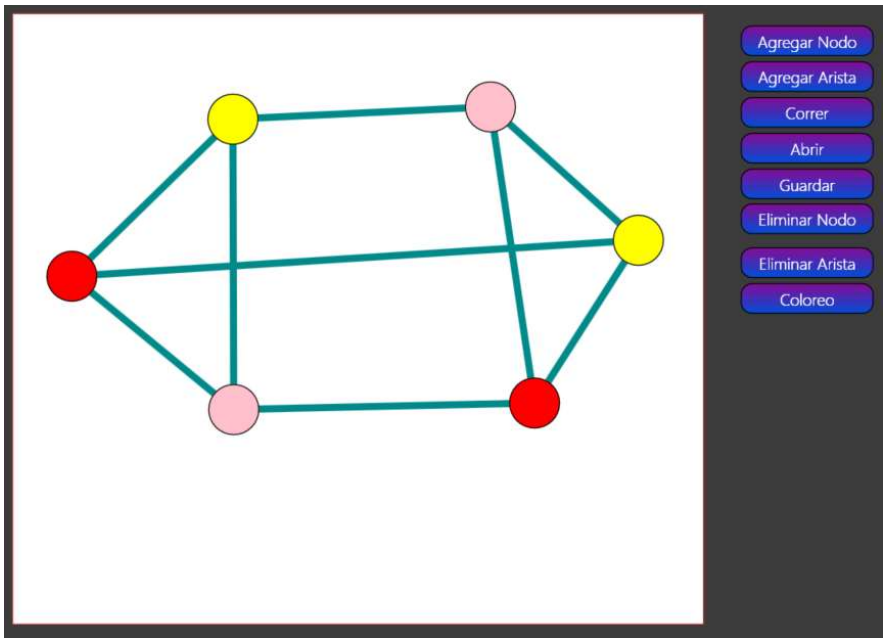


Figura 6.2: Grafo coloreado satisfactoriamente.

Con el fin de comprobar la valides del algoritmo, se igresaron grafos de mayor tamaño de forma aleatoria.

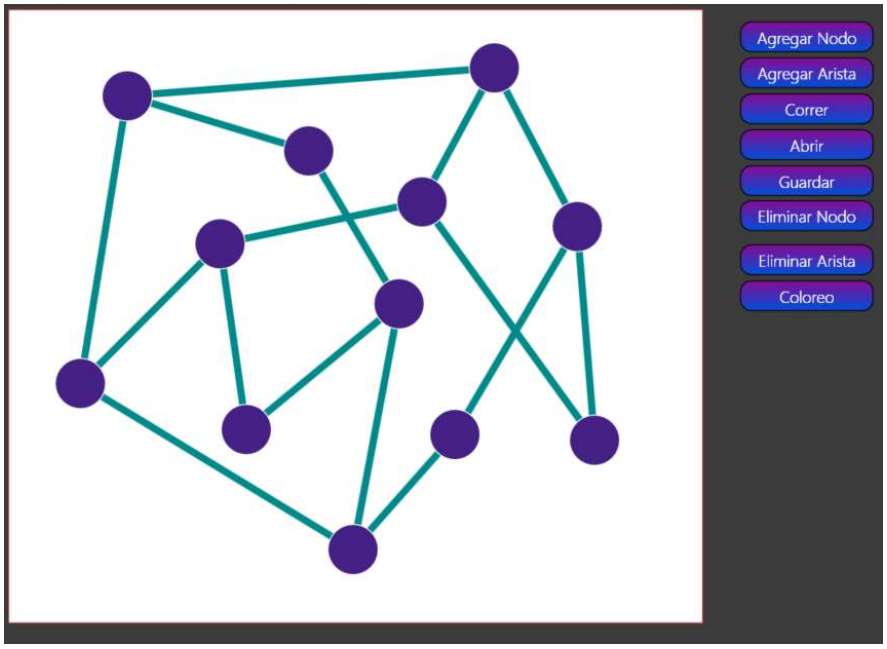


Figura 6.3: Grafo ingreado aleatoriamente.

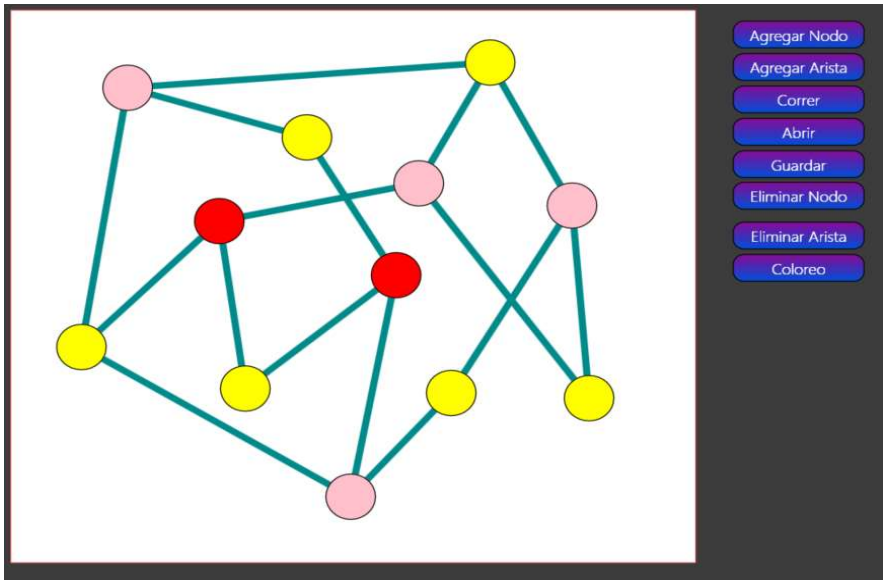


Figura 6.4: Grafo 3 coloreable.

El propósito de crear la segunda propuesta del algoritmo, fue no limitarse al caso 3-coloreable, dándonos como resultado una heurística aplicable a grafos *k*-coloreables, por lo que también analizaremos grafos con un número cromático mayor a 3.

Ingresando a la aplicación los grafos propuesto en el capítulo 5.6, encontramos que los resultados son los esperados (Véase figura 6.6).

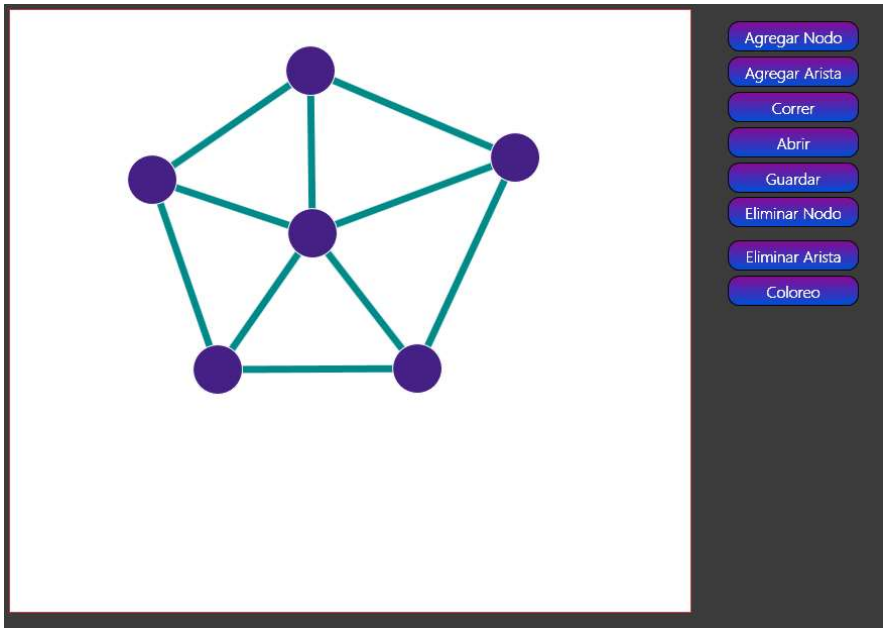


Figura 6.5: Grafo 4 coloreable (ejemplo 3, sección 5.8).

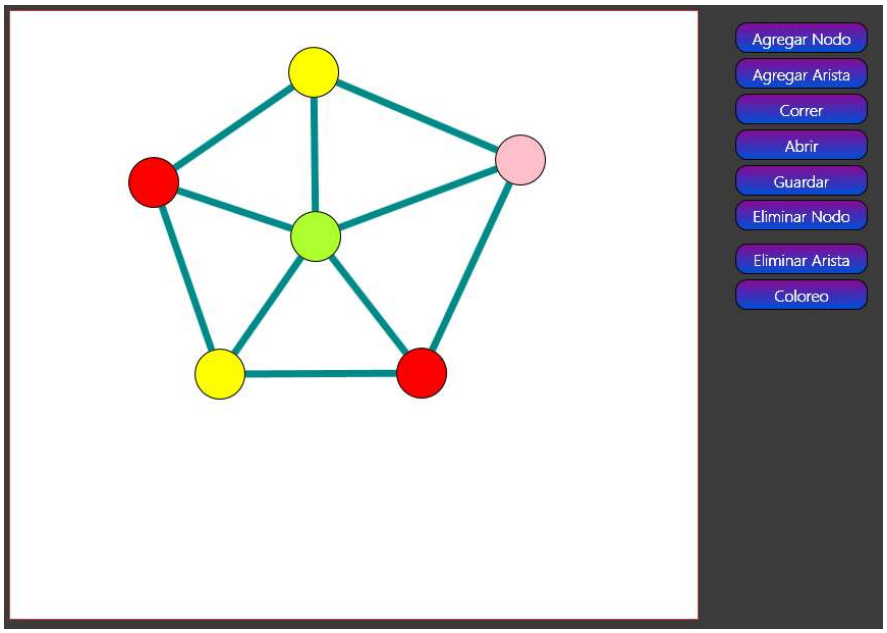


Figura 6.6: Grafo coloreado satisfactoriamente.

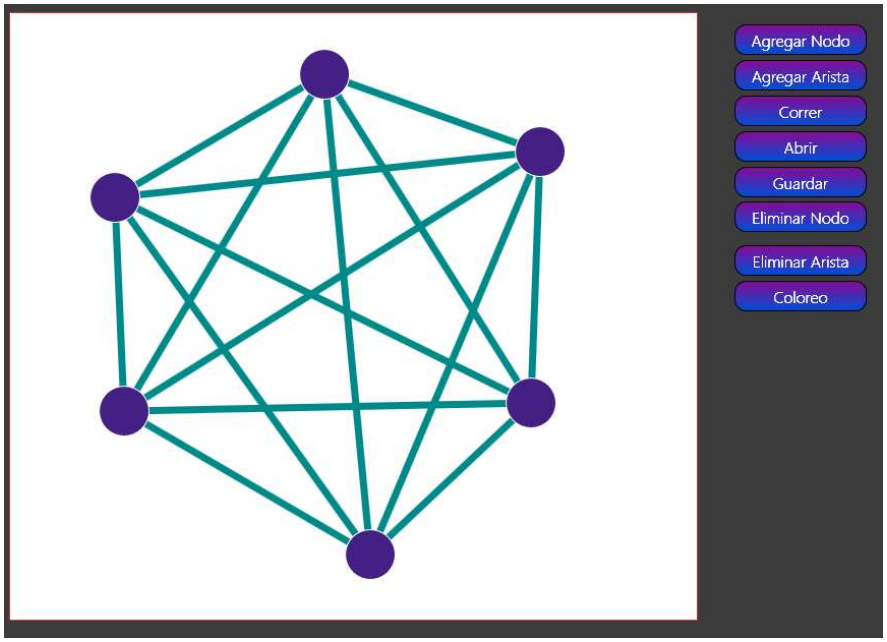


Figura 6.7: Grafo 6 coloreable.

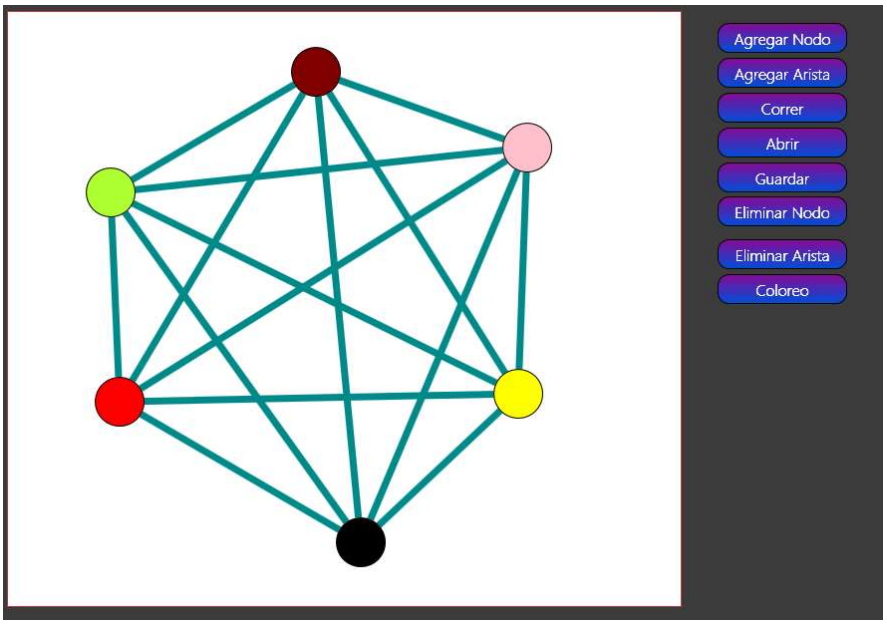


Figura 6.8: Grafo coloreado satisfactoriamente.

7 CONCLUSIONES

En ciencias de la computación, problemas en la clase de complejidad NP - Completo son objetos de estudio de gran importancia. Entre las razones de esta importancia, se tiene que muchos de estos problemas son la base en la resolución de problemas de interés práctico, por ejemplo, en este documento de tesis, se mencionaron algunos problemas de interés práctico basados en el coloreo de un grafo.

Otra razón es que, hasta el momento, no se ha construido un método completo y eficiente que resuelva de forma exacta alguno de los problemas en esta clase *NP*- Completo, lo que los hace, un gran reto para las investigaciones en el área del diseño y análisis de algoritmos.

En este trabajo de tesis se propone una nueva heurística para el cálculo eficiente del número cromático de un grafo.

En la primera heurística propuesta, la complejidad es polinomial, aunque su aplicación es limitada ya que considera sólo el caso del 3-coloreo de grafos. Sin embargo, también su orden de complejidad en tiempo que está acotada por $O(m * n^2)$, nos muestra que el orden del polinomio es a lo más cúbico de acuerdo al tamaño de sus entradas, lo que de cierta forma, es una algoritmo que ejecuta rápidamente.

En cierta forma, podemos considerar nuestras propuestas algorítmicas, como una búsqueda local sobre vecindades de segundo orden. Esto es, una vez que se decidió colorear a un nodo x, se salta sobre sus nodos vecinos $N(x)$, y se busca seleccionar entre los vecinos de los vecinos $N(N(x))$, que nodo y aparece en un número máximo de ciclos intersectados y de longitud impar, y además que aparte de no estar coloreado, no sea parte de $N(x)$.

El seleccionar el nodo y que aparece en un número máximo de ciclos intersectados y de longitud impar, inmediatamente después de que se encuentra, nos determina una heurística ávida, que es caracterizable por ser una propuesta eficiente, como lo hemos

mostrado en la sección del análisis de la complejidad en tiempo de nuestros algoritmos, aunque no garantiza que siempre encontrará el coloreo óptimo.

La segunda versión de nuestra propuesta heurística, corrige la limitante de ser aplicada sólo al 3-coloreo, permitiendo así un *k*- coloreo, la complejidad en tiempo del algoritmo resultante incrementa en uno el grado del polinomio que acota superiormente la complejidad en tiempo, esto es de orden $O(m * n^3)$, en el peor caso. Por lo que nuestra segunda propuesta se mantiene como un algoritmo eficiente.

Ambos algoritmos, fueron probados con diferentes instancias, obteniéndose resultados satisfactorios. Sin embargo, al ser propuestas heurísticas, se espera que para instancias difíciles de prueba, puedan no hallar los resultados óptimos.

Podemos concluir que nuestras propuestas algorítmicas, abonan en el campo del diseño de heurísticas para la solución de problemas de la clase *NP*-completos, haciendo nuevas propuestas en el diseño eficiente de búsquedas en estructuras que tienen un número exponencial de posibles combinaciones de sus elementos, en nuestro caso, un número exponencial de posibles coloreos sobre los nodos de un grafo.

8 REFERENCIAS

[1] Gary Chartrand, Ping Zhang, *Introduction to Graph Theory*, Mc Graw-Hill, Int. Edition, New York, (2005).

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

[3] Richard M. Karp, *Reducibility Among Combinatorial Problems*, In R. E. Miller and J. W. Thatcher, Complexity of Computer Computations. New York ,Plenum (1972), pp. 85–103.

[4] Byskov J.M., *Exact Algorithms for graph coloring and exact satisfiability*, Phd thesis, University of Aarbus, Denmark, (2005).

[5] Alfred V. Aho, Ravi Sethi, *Compiladores: Principios, técnicas y herramientas,* AT &T Bell Laboratories Murray Hill, NJ (1990).

[6] Fausto A. Toranzos, *Introducción a la teoría de grafos*, Programa regional de desarrollo científico y tecnológico. Departamento de asuntos científicos. Secretaria general de la O. E. A. Washington, D.C (1976).

[7] Reinhard Diestel, *Graph Theory,* Springer, New York, (2010).

[8] G. Hernández*, Grafos: Teoría y Algoritmos*, Servicio de Publicaciones, Facultad de Informática, UPM, (2003).

[9] Golumbic, M.C., *Algorithmic Graph Theory and Perfect Graphs*, 2nd edn. North Holland (2004).

[10] Abraham Iniesto Díaz, Juan Carlos Delgado Núñez, *Algoritmos de coloración*, http://www.dma.fi.upm.es/gregorio/grafos/IAGraph/coloracion.html

[11]Adam Drozdek, *Estructuras de datos y algoritmos con Java*, edición 2, International Thomson Editores, (2007).

[12] Guillermo De Ita Luna, Raymundo Marcial-Romero, Yolanda Moyao, *AnApproximate Algorithm for theChromatic Number of Graphs*, Electronic Noteson Discrete Mathematics, Vol. 46 (2014), pp. 89–96.

[13] Angelica Guzman Ponce, José Raymundo Marcial-Romero, José A. Hernández, Guillermo De Ita Luna, *An algorithm to approximate the chromatic number of graphs*, Electronics, Communications and Computers (CONIELECOMP), International Conference on, (2015),pp 110-115.

[14] Arora S, Chlamtac E, Charikar M, *New Approximation Guarantee for ChromaticNumber*, Proceedings STOC 2006, (2006).

[15] Johnson D, *The NP-Completeness Column: An Ongoing Guide,* Jour. of Algorithms 6,(1985), pp.434 - 451.

[16] Lawler E, *A note on the complexity of the chromatic number problem*, Information Processing Letters 5, (1976), pp.66-67.

[17] Mabrouk B. B, Hasni H, Mahjoub Z, *On a parallel genetic-tabu search based algorithmfor solving the graph colouring problem*, European Journal of Operational Research,Elsevier, (2009), pp. 1192-1201.

[18] Porumbel D. C, Hao J. K, Kuntz P, *An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring*, Computers & Operations Reserach, Elsevier, (2010), pp. 1822-1832.

[19] Vlasie R. D, *Systematic generation of very hard cases for graph 3-colourability*, Proc. 7th-IEEE Int. Conf. Tools with Artificial Intelligence, (1995), pp. 114-119.

[20] Held S, Cook W, Sewell Edward C, *Maximum-weight stable sets and safe lower bounds for graph coloring*, Springer and Mathematical Optimization Society, (2012).