BENEMÉRITA UNIVERSIDAD AUTÓNOMA
DE PUEBLA
FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

"3-COLOREO DE GRAFOS BASADO EN CICLOS"

TESIS PROFESIONAL
PARA OBTENER EL TITULO DE
MAESTRO EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA

JAVIER AURELIO MARTÍNEZ CASTILLO

ASESOR

DR. GUILLERMO DE ITA LUNA

NOVIEMBRE 2012

AGRADECIMIENTOS

A Dios sobre todas las cosas, todo lo bueno y todo lo malo, por la fortaleza y el discernimiento, por diariamente guiar mi camino y ponerme en los sitios y con las personas con las que debo estar para aprender y ser mejor.

A mis padres, todo su esfuerzo, por sacarnos siempre adelante, por impulsarnos y darnos oportunidades, por sus consejos, por su apoyo, sus desvelos, por todo lo que me han brindado, porque gracias a ellos soy lo que soy.

A mis hermanos, por ser parte de este camino, por esos momentos que vivimos y seguimos viviendo, por todas esas experiencias asimiladas a lado de ellos, esas cosas que nos forjan y nos hacen apreciar la vida y aprender de ella.

A mi novia, por tantas cosas vividas juntos, por los años de aprendizaje, de paciencia, por darme motivos para ser y hacer, por todo su amor, apoyo y los ánimos que siempre me brinda para seguir adelante.

A mi asesor de tesis, gracias Dr. Guillermo de Ita Luna, por todo lo aprendido con usted, por su guía, por su apoyo, por el tiempo que me dedicó, tanto en el aula como siendo mi asesor y por su labor para conmigo como profesor y como persona.

CONTENIDO

I) Introducción

Un grafo es una representación matemática de sumo interés para su estudio, ya que a través de él, se pueden modelar diversas situaciones de la vida real, tales como líneas de autobuses o del metro (Fig. 1.), calendarización de tareas "scheduling" (Fig. 2), etc. Esto nos permite plantear y solucionar problemas relacionados a estos modelos utilizando la teoría de grafos.

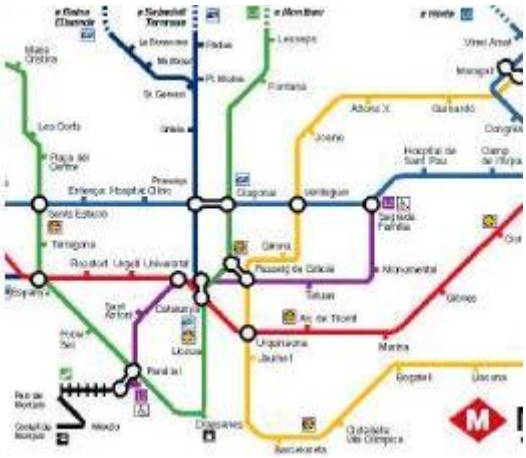


Fig. 1. Líneas del metro.

Se describe a continuación un grafo de manera informal para efectos de introducir al lector al problema, sin embargo más adelante será detallado de manera formal. Un grafo consiste en un conjunto de nodos o vértices unidos a través de aristas las cuales pueden o no llevar una dirección, esto implica una clasificación de los grafos en grafos dirigidos o no dirigidos, para nuestro caso de estudio trabajaremos exclusivamente con grafos no dirigidos (Fig. 3.).

Actividad	a1	a2	a3	a4	a5	a6	a7	a8
Tiempo	4	3	7	4	6	5	2	5
Prerrequisitos			a1	a1	a2	a4, a5	a3, a6	a4, a5

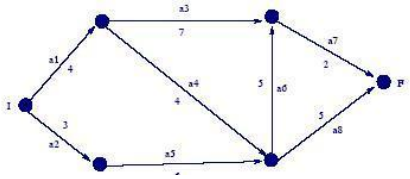


Fig. 2. Programación de actividades.

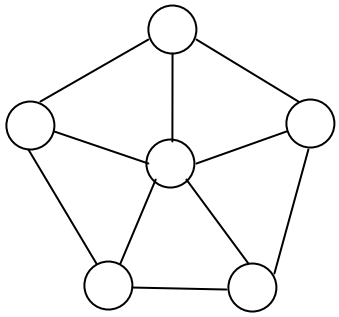


Fig. 3. Grafo no dirigido.

1.1 Planteamiento del Problema

Definido lo anterior podemos hablar del coloreo de grafos, el cual es un problema que consiste en asignar o colorear los nodos de un grafo. La meta central en el coloreo de grafos consiste en colorear los nodos de forma que ningún par de nodos adyacentes (es decir dos nodos unidos por una arista) tengan el mismo color, a esto se le conoce como coloreo propio. El *k*-coloreo propio consiste en colorear un grafo estrictamente con *k* colores sin asignar el mismo color a dos nodos adyacentes. El problema sobre el cual se enfoca nuestra investigación es el 3-coloreo propio. En la figura 4 se muestra un grafo con un 3-coloreo propio.

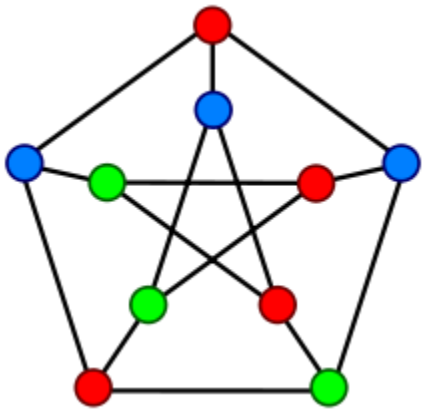


Fig. 4. Grafo coloreado con 3 colores.

Denotamos ahora el número cromático de un grafo ***G***, de la siguiente manera, ***χ(G)*** esta notación representa el número mínimo de colores con los cuales puede ser coloreado el grafo ***G***.

El plantear el problema de coloreo de grafos es algo realmente sencillo, sin embargo su solución no lo es tanto, de hecho, determinar el número cromático ***χ(G)*** es uno de los problemas más difíciles de atacar, sobre todo por el tiempo computacional que se requiere para su solución.

A la fecha, determinar si ***χ(G)*** = 2, es decir, que un grafo dado ***G*** puede ser coloreado propiamente solo con dos colores, es un problema que tiene una solución computacional que requiere tiempo de ejecución de orden polinomial, por lo tanto, se dice que tiene una solución computacionalmente eficiente. Sin embargo, al trabajar con grafos cuyo número cromático es ***χ(G)*** es mayor o igual a 3, encontramos que hasta ahora, todas las propuestas para determinarlo no son eficientes, ya que el tiempo que requieren para ello es de complejidad exponencial.

Es aquí donde se entiende porqué el interés dedicado a este trabajo a la investigación del 3-coloreo, ya que al no ser resuelto eficientemente aún, seguirá requiriendo de nuevas propuestas algorítmicas, como las planteadas en esta tesis.

1.2 Estado del Arte

El trabajo de Leonhard Euler, en 1736, sobre el problema de los puentes de Königsberg es considerado como uno de los primeros resultados de la teoría de grafos [18]. También se considera uno de los primeros resultados topológicos en geometría (que no depende de ninguna medida). Este ejemplo ilustra la profunda relación entre la teoría de grafos y la topología.

En 1845 Gustav Kirchhoff publicó sus leyes de los circuitos para calcular el voltaje y la corriente en los circuitos eléctricos [18].

En 1852 Francis Guthrie plantea el problema de los cuatro colores [18], utilizando solo cuatro colores, colorear cualquier mapa de países de tal forma que dos países vecinos nunca tenga el mismo color. Este problema, que no fue resuelto hasta un siglo después por Kenneth Appel y Wolfgang Haken [19], puede ser considerado como el nacimiento de la teoría de grafos. Al tratar de resolverlo, los matemáticos definieron términos y conceptos teóricos fundamentales de los grafos.

Determinar el número cromático $\chi(G)$ de un grafo G se presenta en una variedad de aplicaciones, y fue uno de los 22 problemas planteados como NP-completos en la lista de Karp [1]. La determinación de $\chi(G)$ es polinomialmente computable cuando $\chi(G) \leq 2$, pero cuando $\chi(G) \geq 3$, el problema se convierte en NP-completo, incluso para los grafos G con grado $\Delta(G) \geq 3$.

Posteriormente, se dedicó mucho esfuerzo en tratar de diseñar algoritmos de aproximación eficientes, es decir, dado un grafo k-coloreable, tratar de colorearlo de manera eficiente, y con l colores, $l \geq k$, y donde l es tan pequeño como sea posible. Y en esta misma línea, varias investigaciones se han realizado para determinar cuándo un grafo es 3-coloreable.

En el área de los algoritmos exactos para calcular el número cromático de un grafo, el primer algoritmo fue diseñado por Lawler con una complejidad en tiempo de $O(2.4423^n)$, los factores polinomiales son ignorados. Después de su presentación, el algoritmo de Lawler no fue mejorado durante 25 años [2].

Siguiendo la línea de algoritmos exactos y utilizando máximo conjuntos independientes para calcular el número cromático, Eppstein estableció un algoritmo con complejidad en tiempo de $O(2.4151^n)$ [3].

Posteriormente, Byskov proporciona un algoritmo con complejidad en tiempo de $O(2.24023^n)$ [4]. Todos los algoritmos utilizan una combinación de límites superiores basados en mejores aproximaciones del número de conjuntos independientes máximos de tamaño a lo mucho *k*, y algunos cambios en la manera de llenar la tabla de sub-soluciones en un enfoque de programación dinámica.

Cuando el número de colores es fijo, algoritmos más rápidos han sido diseñados. Uno de los primeros para grafos 3-coloreables se estableció por Wigderson [5] quien mostró cómo colorear un grafo 3-coloreable con a lo sumo $3\cdot\sqrt{n}$ donde *n* es el número de nodos en el grafo. Blum y Karger [6] aplicaron la programación semidefinida (SDP) para mejorar a $O(n^{3/14})$, donde se suprimen los factores polilogarítmicos. Arora, et. Al. Aplicando programación semidefinida (SDP) ha mejorado la propuesta de solución del coloreo de grafos con cotas superiores de tiempo de orden $O(n^{0.2111})$ colores [7].

Las cotas superiores de tiempo más conocidas en la actualidad para el *k*-coloreo, son las siguientes: $O(1.3289^n)$ para $k = 3$. Para $k = 4$, $O(1.7504^n)$ el algoritmo Byskov, y para $k = 5$, $O(2.1020^n)$ del algoritmo de Byskov y Eppstein.

Alon describe una técnica para colorear aleatoriamente grafos 3-cromáticos en tiempo polinomial. Vlasie ha descrito una clase de casos que son (a diferencia de los grafos aleatorios 3-cromáticos) difíciles de colorear [8].

Aunque determinar el 3-coloreo de un grafo es un problema NP-completo, en la actualidad se han reconocido varios patrones topológicos en los grafos que permiten decidir la 3-coloreabilidad. Uno de los Primeros Resultados en esta línea, es el famoso teorema de Grotzsch [9], Que garantiza que cada grafo plano libre de triángulos es 3 coloreable. A partir de este trabajo pionero varias propuestas se han desarrollado a través de reconocer los patrones en el grafo para determinar la 3 coloración.

Por ejemplo, Borodin et.al han demostrado que grafos planos sin ciclos de tamaño de 4 a 7 son 3 coloreables [10]. Gimbel y Thomasson [11] encontraron una 3 coloreabilidad para grafos planos con proyecciones libres de triángulos.

Dvorak [12] encontró un algoritmo de tiempo lineal para decidir si un grafo sin triángulos en una superficie general Σ es 3 coloreable.

Por otro lado, para $k \leq 2$ el problema es solucionable polinomialmente, como es el problema general de coloreo para muchas clases de grafos tales como: grafos de intervalos, grafos acordes, grafos de comparación [13], el 3-coloreo de grafos libres de AT [14] y, en general, para grafos perfectos [15]. En estos casos, las topologías especiales del grafo que se han encontrado, permiten identificar tanto a los tipos de grafos como también permiten diseñar algoritmos de tiempo polinomial para su coloreo.

1.3 Relevancia del Problema

Las aplicaciones del 3-coloreo de grafos son demasiadas, a continuación se presentan solo algunos ejemplos.

Telefonía celular. El sistema de telefonía móvil está dispuesto a través de celdas, en este sistema se espera que los canales disponibles sean asignados de forma que cada celda tenga un grupo único de frecuencias y no haya colisiones entre células adyacentes. El coloreo de grafos aplicado en esta área puede ayudar a reducir el número de frecuencias cuando se diseña una red de telefonía para un área específica, esto por consecuencia nos conduce a un ahorro, al poder reutilizar frecuencias.

Programación de horarios. Este problema aplica para un sinfín de casos: asignar horarios de asesorías para alumnos, citas en dependencias gubernamentales, asignación de horarios para el uso de equipo por parte de trabajadores en empresas, asignación de tiempos para usar recursos o servicios por los usuarios de los mismos, etc. Cualquiera que sea el caso, la reducción a un problema de coloreo de grafos puede plantearse más o menos de la siguiente manera: El grafo representa la forma en la que se asignan los horarios considerando que cada vértice de V es un recurso (computadora, profesor, salón, maquinaria) que puede asignarse, y las aristas indican los recursos que no pueden ser asignados al mismo tiempo. Por lo tanto los colores son los horarios que se asignarán a cada nodo considerando que no se puede asignar el mismo horario a dos recursos adyacentes.

Compiladores. Cuando un compilador calcula la serie de instrucciones para máquina abstracta, lo siguiente es la asignación de registros de la CPU a cada variable, para esto se toma en cuenta si se necesita cada una de las variables y cuando deja de utilizarse, esto nos permite optimizar la ejecución y acceder a memoria un menor número de veces posibles. Para reducir este problema a un coloreo de grafos, se genera una matriz de interferencias, la cual se traduce a un grafo de interferencias, donde cada vértice corresponde a una variable y las aristas unen variables que están vivas al mismo tiempo, es decir, variables que deben estar en registros distintos del CPU. Por lo tanto, el coloreo de tal grafo, dará como resultado la asignación del menor número de registros de la CPU a las variables.

Asignación de Vuelos. Si se quiere asignar vuelos a un determinado número de aviones donde un vuelo específico dura cierto intervalo de tiempo, debemos considerar que si dos vuelos

se empalman no podemos asignarles el mismo avión. Representando el problema como un grafo, los vértices serían los vuelos, y las aristas unen vuelos que se empalman. Los colores serían los aviones que se asignan a los vuelos, obviamente al utilizar el menor número de aviones estamos reduciendo costos.

Como ya se habrá dado cuenta, al resolver el problema del coloreo de grafos, se abarca al mismo tiempo la solución de miles de problemas de la vida real que pueden ser modelados a través de un grafo.

El coloreo de grafos es especialmente útil para situaciones de la vida real en las cuales se requiera optimizar recursos, es por ello que la relevancia de este problema es demasiada si consideramos que este tipo de situaciones son innumerables.

En la actualidad el estudio del coloreo de grafos se desarrolla de manera constante, no son pocos los trabajos e investigaciones relacionados a este tema. Esto se puede observar claramente en la cantidad de artículos publicados en diferentes congresos internacionales. Esta actividad también es evidente en varias universidades como Western Michigan University por poner solo un ejemplo, que cuentan en sus filas con académicos dedicados de lleno al estudio de grafos, y más específicamente al coloreo de los mismos.

1.4 Objetivos del Proyecto

- Generar un aporte en el área de la Teoría de grafos, más específicamente, en el 3-coloreo de grafos.
- Analizar y caracterizar una propuesta actual basada en modelar el problema de coloreo de grafos usando fórmulas dos conjuntivas.
- Una vez caracterizada la propuesta antes mencionada, refinarla considerando las diferentes instancias que podrían no quedar adecuadamente caracterizadas.
- Generar un nuevo algoritmo basado en ciclos usando la búsqueda a lo profundo, que también trabaje sobre cláusulas conjuntivas.
- Analizar el algoritmo mencionado en el punto anterior.

Con lo anterior se busca contribuir en el desarrollo y análisis de algoritmos como parte del grupo de investigación del cuerpo académico de algoritmos combinatorios y aprendizaje de la Facultad de Ciencias de la Computación, particularmente en el campo del coloreo de grafos.

II) Teoría de Grafos y Técnicas de Búsqueda

A continuación se hablará de la Teoría de Grafos, la cual es un área comprendida tanto en matemáticas como en ciencias de la computación. Estudia una estructura matemática especial llamada grafo, así como sus propiedades. Un grafo es un conjunto de nodos o vértices unidos por medio de aristas, estas pueden ser o no dirigidas. Generalmente se representa un grafo como puntos unidos por líneas.

2.1 Conceptos Preliminares de la Teoría de Grafos

Sea *G* = (*V, E*) un grafo no dirigido simple (es decir, finito, sin loops y sin aristas múltiples) con conjunto de vértices (o conjunto de nodos) *V* y conjunto de aristas *E*. *E*(*G*) y *V*(*G*) son usados para enfatizar los conjuntos de aristas y vértices de un grafo determinado *G*. Dos nodos *v* y *w* que pertenecen a *V* se llaman adyacentes si hay una arista {*v*, *w*} ∈ *E*, uniéndolos.

El vecindario de $x \in V$ es $N\ x = y \in V: x,y \in E$ y su vecindario cerrado es *N*(*x*) ∪ {*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* ∈ *V*, denotado por *δ*(*x*), es |*N*(*x*)|. El tamaño del vecindario de *x*, *δ*(*N*(*x*)), es $\delta\ N\ x = {}_{y \in N\ x}\ \delta\ y$. 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 $\delta min(G) = min\{\delta(x): x \in V\}$ y con $\delta(G) = (2 \cdot |E|) / |V|$ el grado promedio del grafo.

Dado un subconjunto de vértices *S* ⊆ *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: $v_0, v_1, v_2, ..., v_{n-1}, v_n$. 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 $v_0, v_1, ..., v_{n-1}, v_n$ 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 $n\cdot(n\text{-}1)/2$ 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 subgrafo 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 acíclico 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 $\{1, 2,\dots, k\}$ 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 U_1 y U_2, llamados conjuntos partitos, de manera que todas las aristas de G se unen a un vértice de U_1 con un vértice de U_2.

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 $V_1, V_2, \dots, V_k$, llamados clases de color, pero no es posible particionar V en k-1 conjuntos independientes.

2.2 Recorrido a lo Profundo en un Grafo

Una idea simple de como considerar el recorrido a lo profundo, es que se comienza el recorrido 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 este a ser el vértice activo. Cuando todos los vecinos al vértice activo hayan sido ya visitados, se retrocede al vértice X desde el que se alcanzó el vértice activo y se prosigue siendo ahora X el vértice activo.

- **ALGORITMO *dfs***:

Sea $G = (V, A)$ un grafo conexo, $V' = V$ un conjunto de vértice, A' un vector de arcos inicialmente vacío y P un vector auxiliar inicialmente vacío:

1. Se introduce el vértice inicial en P y se elimina del conjunto V'.
2. Mientras V' no sea vacío repetir los puntos 3 y 4. En otro caso parar.
3. Se toma el último elemento de P como vértice activo.
4. Si el vértice activo tiene algún vértice adyacente que se encuentre en V':

   Se toma el de menor índice.

   Se inserta en P como último elemento.

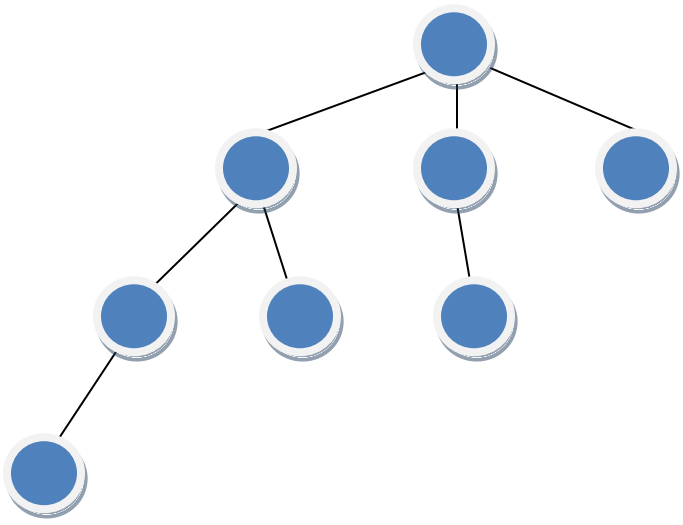
   Se elimina de V'.

   Se inserta en A' el arco que lo une con el vértice activo.

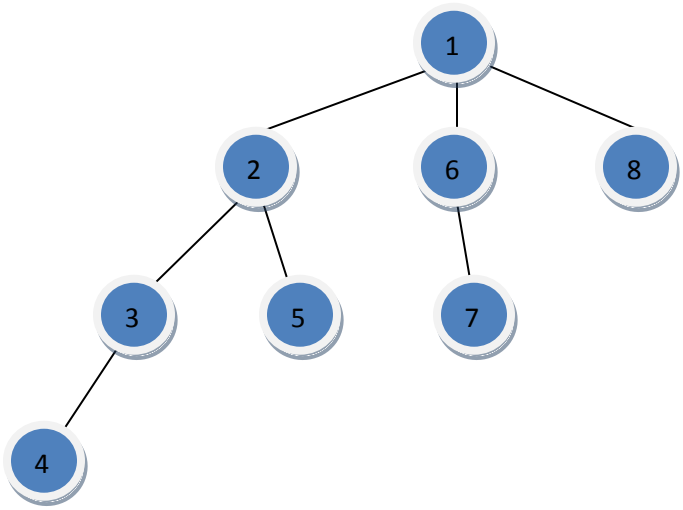
   Si el vértice activo no tiene adyacentes se elimina de P.

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 árbol para mostrar cómo es el recorrido en profundidad:



Se comienza a partir del primer nodo y se recorre el árbol abarcando los hijos de forma que se regrese a cada padre solo después de haber visitado los nodos más profundos, quedando el árbol recorrido en el siguiente orden:



2.2.1 Árbol Generador de un Grafo y Ciclos básicos

Así a través de una búsqueda a lo profundo se genera un árbol que denotaremos con T_G. T_G se forma por los nodos y las aristas de árbol cuando se aplica el procedimiento ***dfs*** (búsqueda a lo profundo) sobre el grafo G. Dado un grafo no dirigido conexo $G = (V, E)$ el aplicar 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 llaman aristas de retroceso.

Sea T el árbol generador de un grafo conexo $G= (V,E)$. Cualquier arista $\{v,w\}$ que pertenece a $E(G - T)$ que se agregue a T formará un ciclo único que se denota como $C_{v,w}$. Donde v y w se conectan por un camino único que denotaremos como $P_{v,w}$. en T. De esta forma $P_{v,w} + \{v,w\}$ forman el ciclo $C_{v,w}$ el cual llamaremos ciclo básico de v a w , con respecto al árbol generador T. Entonces, sea $e \in E(G)$-$E(T_G)$ una arista de retroceso dada. 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 T_G.

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.

Las aristas de retroceso que sean encontradas durante la búsqueda a lo profundo darán como resultado un ciclo básico. Fig 5.

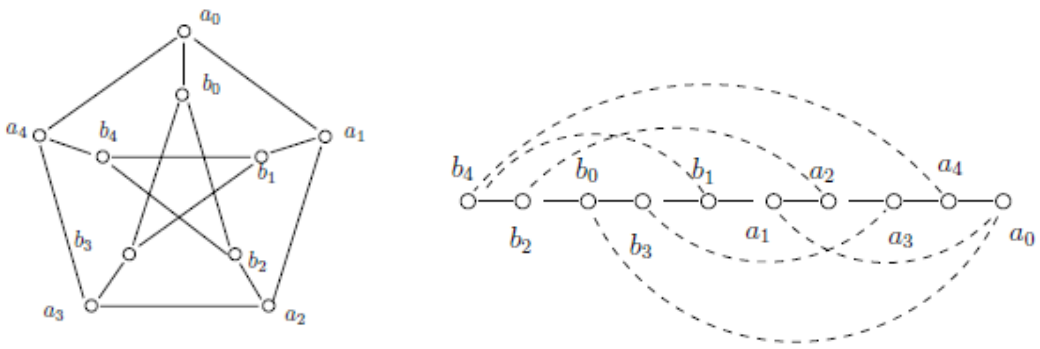


Fig. 5. Grafo de Petersen con su respectivo árbol generador.

Nótese que 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. En la siguiente sección analizaremos las operaciones de unión simétrica entre ciclos.

2.3 Ciclos Intersectados y compuestos en un grafo

Una vez definidos los ciclos básicos procedemos a la identificación de ciclos intersectados, lo cual nos servirá de introducción para definir los ciclos compuestos.

Sea $C = \{C_1, C_2, ..., C_k\}$ el conjunto de los ciclos básicos encontrados durante la búsqueda en profundidad en *G*. Dado un par de ciclos básicos C_i y C_j del grafo *G*, si C_i y C_j comparten aristas comunes se les llama ciclos intersectados, de lo contrario se les llama ciclos independientes.

En particular, si dos ciclos comparten una sola arista se llaman adyacentes. Se dice que un ciclo básico $C_i \in C$ es independiente si C_i y $C_j \in C$ son disjuntos para cada $j <> i$.

Entonces, la operación de diferencia simétrica entre dos ciclos intersectados: $C_i \Delta C_j$ forma un nuevo ciclo. Δ denota la operación diferencia simétrica entre el conjunto de aristas de ambos ciclos, que consiste en unir las aristas de ambos ciclos y restarle a esta unión las aristas comunes de ambos ciclos. Así, $C_i \Delta C_j = (E(C_i) \cup E(C_j)) - (E(C_i) \cap E(C_j))$.

Un ciclo compuesto se forman al aplicar el operador Δ entre cada par de ciclos básicos intersectados: $C_i, C_j \in C$. El resultado de aplicar la operación Δ construye un nuevo ciclo compuesto que contiene las aristas de C_i y C_j excepto las aristas que comparten. Ver Fig. 6.

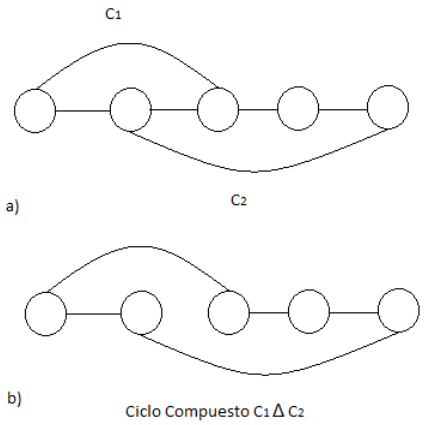


Fig. 6. a) Se muestran dos ciclos básicos C_1 y C_2 intersectados.
b) se muestra el ciclo compuesto generado al aplicar el operador Δ a C_1 y C_2.

Se sabe que cualquier grafo acíclico es 2-coloreable y por lo tanto, 3 coloreable. Además, si un grafo *G* tiene sólo ciclos de longitud par entonces *G* es 2-coloreable ya que alternar dos colores con respecto a los niveles del árbol T_G construye un 2-coloreo propio.

Ciclos de longitud impar requieren por lo menos 3 colores para tener un coloreo propio. Por lo tanto, las estructuras de subgrafos que generan conflicto para el 3-coloreo, son los ciclos impares. Se muestra en las siguientes secciones cómo tratar estas estructuras para la construcción de un 3-Coloreo.

2.4 Problemas en las Clases P y NP para el Coloreo de Grafos

Por conveniencia, la teoría de la complejidad está diseñada especialmente para aplicarse a problemas de decisión. Un *problema de decisión PD* [9] 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*>, donde: *D* - Dominio de valores posibles y $S \subseteq D$ - son las instancias que resuelven el problema. El planteamiento del *PD* es:

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

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 $f: V \rightarrow \{1,2,...,k\}$ 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 [16].

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 *I* de un problema de decisión *PD* con *m* datos de entrada; (d_1, ..., d_m) se define la longitud de la instancia como: $long(I) = \sum_{i=1}^{m} long(d_i)$, donde $long(d_i)$ es la longitud de cada uno de los datos de entrada.

Por ejemplo, si se tuviera una instancia *I* 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:

$$\text{DLOG} = \{PD \mid \exists\ AD \text{ que resuelve } PD \text{ usando espacio logarítmico}\}$$
$$\text{P} = \{PD \mid \exists\ AD \text{ que resuelve } PD \text{ en tiempo polinomial}\}$$
$$NP \quad \{PD \mid \exists\ AnD \text{ que resuelve } PD \text{ en tiempo polinomial}\}$$
$$\text{EXP} = \{PD \mid \exists\ AD \text{ que resuelve } PD \text{ en tiempo exponencial}\}$$

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 [16]. 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, aún 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 y que 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.

Aún y cuando ciertos algoritmos de complejidad exponencial para determinados problemas tienen un buen comportamiento en la práctica, 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 sobre 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 abiertos centrales en la ciencia de la computación [17].

En lo que respecta al problema del coloreo de un grafo, es conocido que 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 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 a 3, 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.

Sin embargo, aparte de grafos bipartitos, se han determinado topologías de grafos que permiten determinar la 3-colorabilidad de los mismos en un tiempo polinomial. Por ejemplo, en el trabajo pionero de Grotzsch [9], se demuestra que cada grafo plano libre de triángulos es 3 coloreable. A partir de este trabajo, algunas otras investigaciones han logrado determinar deferentes topologías de grafos que permiten su 3-colorabilidad en un tiempo polinomial. Estos trabajos se han desarrollado a través de reconocer los patrones subyacentes en el grafo que permiten su 3 coloración de forma eficiente.

Por ejemplo, Borodin et.al han demostrado que grafos planos, sin ciclos de tamaño de 4 a 7 son 3 coloreables [10]. Gimbel y Thomasson [11] encontraron una 3 coloreabilidad para grafos planos de proyección libres de triángulos. Mientras, Dvorak [12] encontró un algoritmo de tiempo lineal para decidir si un grafo sin triángulos en una superficie general Σ es 3 coloreable. En todos estos casos, las estructuras especiales (patrones) que subyacen en las topologías de los grafos permiten diseñar algoritmos de tiempo polinomial para su 3-coloreo.

En el capítulo siguiente presentaremos algunos de los métodos y condiciones utilizadas en los algoritmos que permiten la 3-coloración de un grafo de manera eficiente. Así como también presentaremos algunas topologías simples de grafos que determinan que 3 colores no son suficientes para el coloreo del grafo.

III) Propuestas Algorítmicas para el 3-coloreo de un Grafo

En este capítulo presentaremos topologías de grafos conocidos por ser 3 coloreables, así como condiciones en grafos que impiden el 3-coloreo, con estos dos puntos definidos en los subcapítulos 3.1 y 3.2 procedemos a presentar nuestras propuestas para atacar el problema del 3-coloreo de un grafo.

3.1 Condiciones Necesarias para el 3 Coloreo

Existen topologías de grafos que han sido identificadas por ser 2-coloreables o 3-coloreables. La ventaja de la existencia de estas topologías es que pueden ser reconocidas en un tiempo polinomial en base al tamaño del grafo. Enumeramos aquí, varios casos que describen condiciones necesarias para que un grafo *G* sea 3-coloreable.

1. Si T_G no tiene ciclos básicos impares entonces *G* es 2-coloreable. Esta opción considera el caso de que T_G es un grafo bipartito. Dado que alternar dos colores con respecto a los niveles de T_G construye un árbol 2-coloreable propio, (Fig. 7).

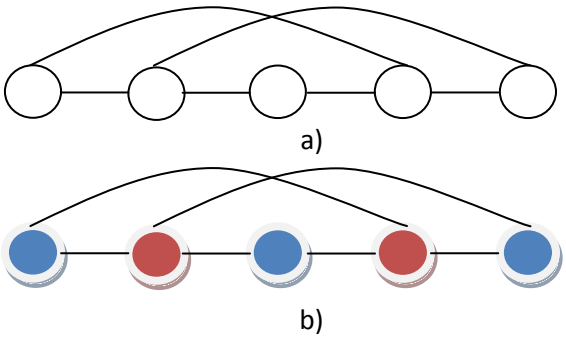


Fig. 7. a) Grafo con dos ciclos básicos, ninguno es impar, por lo tanto es 2-coloreable.
b) El mismo grafo coloreado con 2 colores.

2. Si G es un grafo plano sin triángulos entonces G es 3-coloreable [9].

3. 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, (Fig. 8).

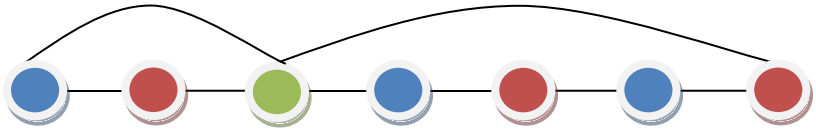


Fig. 8. Grafo con dos ciclos impares que no
comparten ninguna arista (independientes),
por lo tanto de puede colorear con 3 colores como se muestra,
asignando el tercer color al nodo final del ciclo.

4. 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 (Fig. 9).

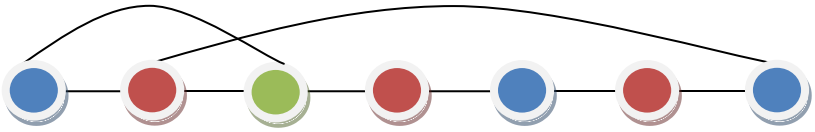


Fig. 9. Ciclo impar intersectado con un ciclo par.
Se colorea con verde el nodo final común y el resto es dos coloreado.

5. 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 más interno al ciclo externo. (Fig. 10). Cuando empezamos a colorear un nuevo ciclo, se utiliza un color diferente a los nodos vecinos (un máximo de dos nodos vecinos), ya que consideramos que sólo los nodos en los ciclos internos ya han sido coloreados. En este caso, podemos considerar a T_G como un ejemplo de un grafo serial paralelo, el cual es conocido por ser coloreado en tiempo polinomial [20].

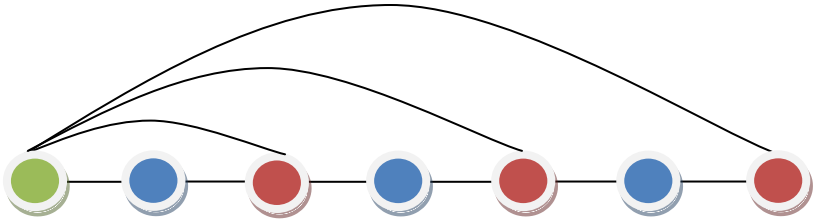


Fig. 10. Ejemplo de ciclos impares embebidos.

3.2 Topologías de Grafos que no son 3-coloreables.

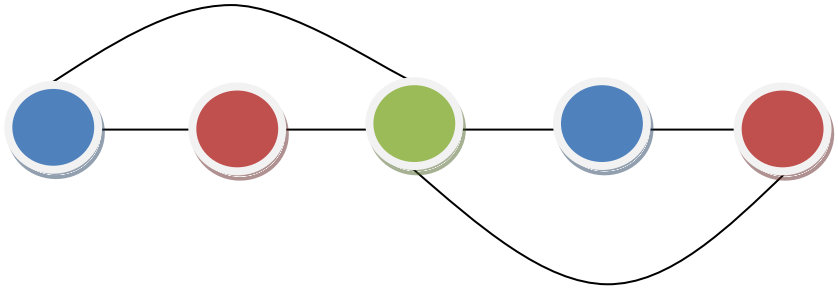
En el apartado anterior se presentaron instancias de grafos a partir de los cuales se puede considerar que un grafo es 3 coloreable.

En la sección anterior se abordó el siguiente caso: Si un ciclo impar en G se cruza solo con ciclos pares o a lo mucho con dos ciclos intersectados, entonces T_G es 3-coloreable.

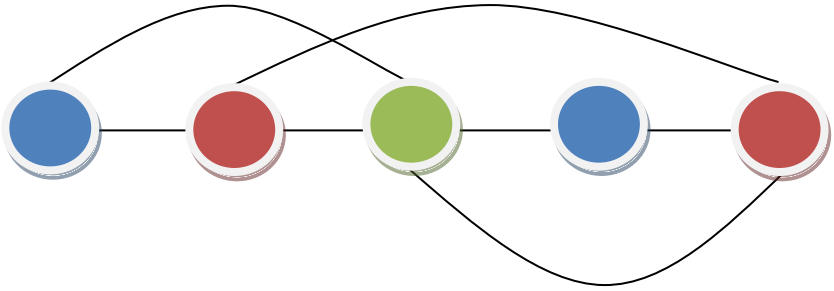
Partimos de este punto para plantear la topología de los grafos que no son 3 coloreables.

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.

Ejemplo:



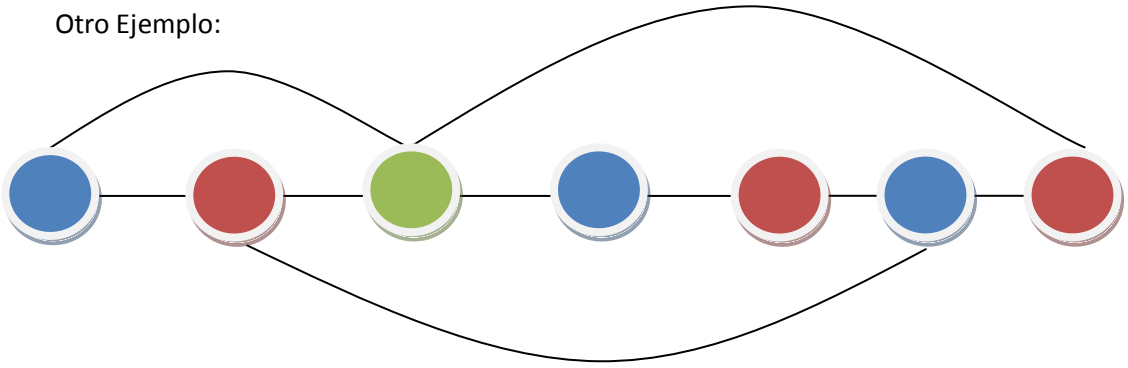
El grafo anterior es 3-coloreable si cerramos un ciclo par cuyos extremos toquen el extremo de los impares, entonces el grafo deja de ser 3 coloreable, de la siguiente forma:



El nuevo ciclo impide que cualquiera de los dos nodos rojos cambie de color para evitar la repetición de color ya que ambos se encuentran tocando tanto nodos de color azul como verde.

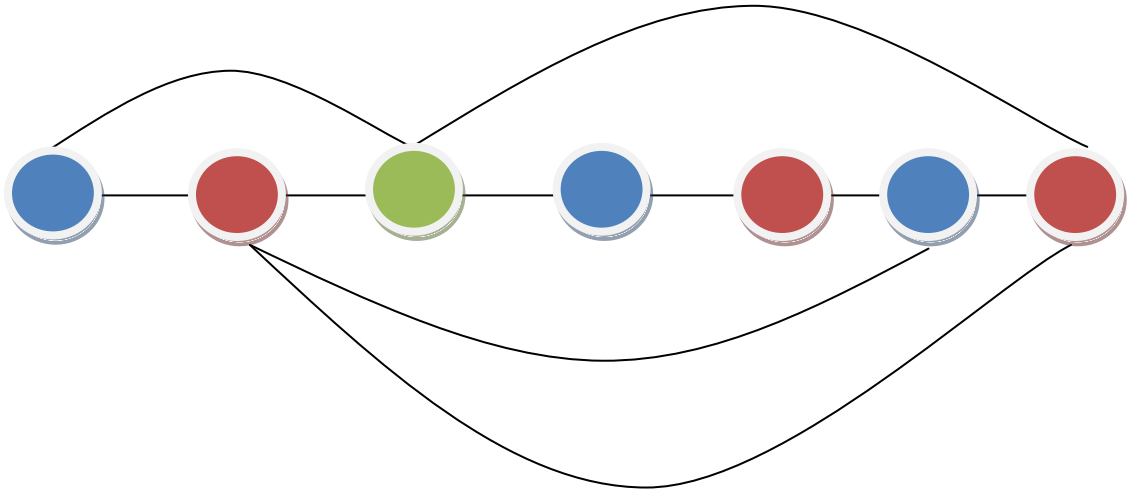
Por lo tanto no es 3 coloreable.

Otro Ejemplo:



Este grafo es 3 coloreable.

Al cerrar el ciclo par intersectado deja de serlo, de la siguiente forma:



Los nodos rojos se hacen adyacentes y no pueden cambiar el color ya que tocan al verde y al azul también, viéndonos obligados a usar un color extra para colorear el grafo de forma propia.

3.3 El Problema de Satisfactibilidad en el Cálculo Proposicional (SAT)

El problema SAT es el primer problema identificado como NP-Completo, esta demostración se le debe a Stephen Cook, hasta ese momento no se encontraba definido el concepto de problema NP-Completo.

El problema SAT consiste en saber si a una expresión booleana con variables y no cuantificada se le puede asignar valores a dichas variables de tal forma que esta asignación haga que la expresión sea verdadera. Es decir, encontrar una instancia para la expresión booleana que la satisfaga.

El problema de decisión asociado al problema de satisfactibilidad (SAT) de una fórmula proposicional en forma normal conjuntiva, puede expresarse como:

Instancia: Un conjunto $U = (u_1, \ldots, u_n)$ de variables booleanas y una colección $F = \{C_1, \ldots, C_m\}$ de cláusulas definidas sobre *U*.

Pregunta: ¿Existe una asignación de valores de verdad a los elementos de *U*, que haga que todas y cada una de las cláusulas de *F* tomen valor verdadero?

Notación:

Asignación: Es una función $t{:}U \to \{F, V\}$.

Literal: El valor verdadero *V* o una expresión de la forma *u* o $\bar{u}$, donde *u* es elemento de *U.*

Frase: Es una conjunción de literales $F = l_1 \wedge \cdots \wedge l_k$. Una frase es verdadera si todas sus literales lo son.

Cláusula: Es una disyunción de literales, $C = l_1 \vee \cdots \vee l_k$. Una cláusula es verdadera si alguna de sus literales lo es.

Forma normal conjuntiva (FNC): Es una conjunción de cláusulas.

Una FNC-*k*, $k \in N$, es una FNC con exactamente *k* literales por cada cláusula.

Forma normal disyuntiva (FND): Es una disyunción de frases.

Una FND-*k*, $k \in N$, es una FND con exactamente *k* literales por cada frase.

Una fórmula es monótona positiva si todas las variables en la fórmula aparecen solo de forma positiva, es monótona negativa cuando todas las variables aparecen solo de forma negativa. Una variable es monótona si aparece de manera pura en la formula, es decir, solo aparece o bien de forma positiva o bien de forma negativa, pero no de ambas formas en la misma fórmula. Diremos que una cláusula es monótona cuando todas sus literales en ella, aparecen bien sólo de forma positiva o bien sólo de forma negativa. Note que la conjunción de cláusulas monótonas no necesariamente conforma una fórmula monótona (ya sea positiva o negativa).

$\bar{l}$ denota la negación de la literal *l*. Para cualquier variable *x* el par {*x*, $\bar{x}$} es *complementario*. Una variable *x* aparece en una cláusula *C* si cualquiera *x* o $\bar{x}$ es un elemento de *C*. Sea *v*(*c*)={$x \in X$: *x* aparece en *c*}. Una literal *l* es pura en una fórmula *F* si *l* aparece en *F* y su negación $\bar{l}$ no aparece en *F*.

Una cláusula vacía es una contradicción. Una cláusula es tautológica si tiene un par complementario de literales. Una cláusula es unitaria cuando tiene solo una literal, y es binaria cuando tiene exactamente dos literales. Una cláusula es de Horn si contiene a lo más una literal de forma positiva.

Una asignación *s* puede ser considerada como un conjunto de literales donde no hay un par complementario de literales. Si $l \in s$, entonces *l* se hace verdadera y se hace falsa $\bar{l}$.

Una cláusula *c* se satisface por una asignación *s* si y solo si ($C \cap s$) $\neq \varnothing$. De otra forma podemos decir que se hace falsa o se contradice por *s*. Una fórmula conjuntiva *F* se satisface por una asignación *s* si cada cláusula de *F* se satisface por *s*. *F* se hace falsa por *s* si cualquier cláusula de *F* se contradice por *s*. Un modelo de *F* es una asignación *M* sobre *v*(*F*) que satisface *F*, y es denotado por $M \models F$, indicando que la asignación *M* es un modelo de *F*.

Sea SAT(*F*) el conjunto de modelos que *F* tiene sobre su conjunto de variables. Si SAT(*F*) $\neq \varnothing$ entonces *F* es satisfactible de otra forma *F* es insatisfactible. Para dos cláusulas equivalentes lógicas *F*

and *F*', denotadas como $F \leftrightarrow F'$, ambas tienen el mismo conjunto de modelos, que es, SAT(*F*) = SAT(*F*').

Proposición 1: Toda fórmula proposicional es lógicamente equivalente a una FNC, y de hecho la FNC equivalente es algorítmicamente calculable [21].

Una FNC *F* es *satisfactible* si y sólo si existe una asignación de valores de verdad para *U* que simultáneamente satisfaga a cada una de las cláusulas en *F*.

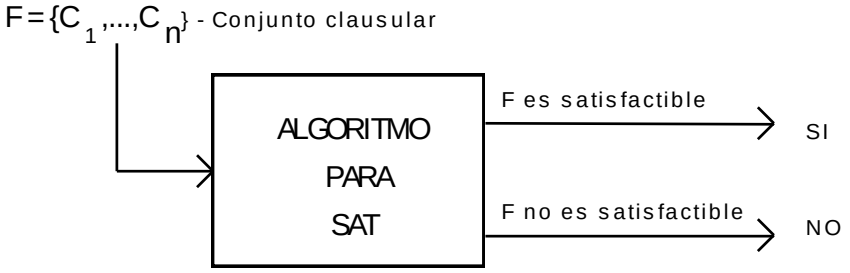
Por ejemplo, una instancia específica del problema SAT puede ser la siguiente fórmula *F*, sobre la que se cuestiona si es o no satisfactible:

$$F(x_1,..., x_7) = (\overline{x_1} \vee \overline{x_2}) \wedge (x_1 \vee x_3 \vee x_4) \wedge (x_5 \vee \overline{x_3}) \wedge (\overline{x_5} \vee \overline{x_6}) \wedge$$
$$(x_1 \vee x_6 \vee x_7) \wedge (\overline{x_5} \vee \overline{x_7}) \wedge x_2 \wedge (\overline{x_3} \vee \overline{x_7})$$

La fórmula anterior es satisfactible con la asignación: $t(x_1) = t(x_3) = t(x_5) = t(x_6) = F$; $t(x_2) = t(x_4) = t(x_7) = V$.

Toda asignación que satisfaga a una fórmula *F* se dice ser un *modelo* para *F.* El problema SAT consiste en decidir, si dada una fórmula *F* de entrada, que sin pérdida de generalidad puede suponerse en FNC (de acuerdo a la proposición 1), se determine si acaso existe una asignación de *U* que haga que *F* tome el valor verdadero.

Denotamos con SAT(*F*) al conjunto de modelos que satisfacen a la fórmula *F*. En caso de que para una fórmula *F*, SAT(*F*) = Ø se dice entonces que *F* es una contradicción o que es insatisfactible.



Planteamiento del problema SAT

El primer problema que se demostró ser NP completo fue el problema SAT [22]. Con esto Cook delimitó la clase de los problemas NP-completos. Max-SAT es la versión optimizada del problema SAT. Max-SAT consiste en determinar el número máximo de cláusulas que pueden ser satisfechas simultáneamente con las asignaciones de *F*. Sea LANG-SAT la notación para el problema SAT, cuando las fórmulas proposicionales en la clase LANG-FC están involucradas i.e. los problemas 2-SAT and Max-2SAT son los problemas SAT and Max-SAT para fórmulas que están en 2-FC, respectivamente. Ambos, SAT and Max-SAT son problemas NP-Completos, incluso para 3-FC's para el problema SAT y para 2-FC's en el caso del problema Max-SAT.

3.4 El Grafo de Dependencias de una 2-FC

Sea $\sum$ una 2-FC, expresamos como $G_{\sum}$ =(V,E) el grafo de dependencias originado por las restricciones de $\sum$. El conjunto de nodos de $G_{\sum}$ es V=*Lit*($\sum$) $\cup${T,$\perp$}, donde $\perp$ y T son constantes en el lenguaje que representan los valores falso y verdadero, respectivamente.

Para cada cláusula binaria c = (x, y) $\in \sum$ hay dos aristas dirigidas en E, dadas por: $\overline{x} \rightarrow y$ y $\overline{y} \rightarrow x$. Para cada cláusula unitaria c = (u) $\in \sum$ hay solo dos aristas dirigidas en E dadas por $T \rightarrow u$ y $\overline{u} \rightarrow \perp$. Estas aristas reflejan el hecho de que cada cláusula es equivalente a un par de implicaciones. Nosotros llamamos a estas dos aristas dirigidas originadas por c, las *aristas representativas* de la cláusula binaria c.

Sea "$\Rightarrow$" la cerradura reflexiva y transitiva de $\rightarrow$ sobre cualquier nodo de V. De ello se deduce inmediatamente que para dos literales x y y, si $x \Rightarrow y$ entonces si una solución factible M (un modelo de $\sum$) tiene a la literal x con valor lógico de verdadero, entonces la literal y tiene también que ser verdadera.

T(x) denota el conjunto de todas las literales y forzadas por x, $\forall x \in$ *Lit*($\sum$), i.e. T(x) ={ $y \in$ *Lit*($\sum$) : $x \Rightarrow y$ }. Como ($x \Rightarrow y$) $\equiv$ ($\overline{y} \Rightarrow \overline{x}$), entonces si $y \in T(x)$ entonces $x \in T(\overline{y})$. Note que $T(\overline{x})$ puede ser visto como el conjunto de cláusulas unitarias resultantes al aplicar la resolución unitaria sobre $\sum$ tomando en cuenta la cláusula unitaria ($\overline{x}$).

Además, si $\overline{x} \in T(x)$ entonces en cualquier modelo de $\sum$, no podemos tener x asignada como verdadera. De igual forma, si $x \in T(\overline{x})$, entonces x no puede ser falsa en cualquier modelo de $\sum$. Para cualquier literal x en $\sum$, T(x) es catalogado como consistente o inconsistente (contradictorio). Es decir que T(x) es inconsistente si $\overline{x} \in T(x)$ o $\perp \in T(x)$, de otra forma se dice que T(x) es consistente.

Note que si $\sum$ tiene una cláusula unitaria (u) entonces $T(\overline{u})$ es inconsistente. Porque $\overline{u} \rightarrow \perp$ y entonces $\perp \in T(u)$. Si hay una variable $x \in v(\sum)$ tal que ambas T(x) y $T(\overline{x})$ son contradictorias, entonces $\sum$ es insatisfactible. Si $\sum$ es satisfactible, entonces $\forall l$, $l \in$ *Lit*($\sum$), $\sum \nvdash l$ si $T(\overline{l})$ es contradictorio.

Sea *Cont*($\sum$)={$l \in$ *Lit*($\sum$): T(l) es contradictorio}. Note que para cualquier literal $l \in$ *Cont*($\sum$), ($\sum \cup$ {(l)}) es insatisfactible. Sea *Base*($\sum$)={$l \in$ *Lit*($\sum$): T(l) es consistente y $T(\overline{l})$ es contradictorio}.

Base($\sum$) determina el conjunto de variables que aparecen en cualquier modelo de $\sum$ con solo uno de los dos posibles valores lógicos. De hecho, el valor lógico con el cual aparecen las variables es el mismo con el que aparecen en *Base*($\sum$). Tenemos que *Base*($\sum$) $\cap$ *Cont*($\sum$) = $\varnothing$.

Sea *Any*($\sum$)=*Lit*($\sum$)-(*Base*($\sum$) $\cup$ *Cont*($\sum$)), entonces $\forall l \in$ *Any*($\sum$), l es una literal que podría aparecer con cualquiera de los dos valores lógicos en algún modelo de $\sum$. Note que si $l \in$ *Cont*($\sum$) entonces $\overline{l} \in$ *Base*($\sum$), así *Base*($\sum$) $\cap$ *Any*($\sum$)= $\varnothing$. Entonces el conjunto {*Cont*($\sum$),*Base*($\sum$),*Any*($\sum$)} es una partición de *Lit*($\sum$). Si SAT($\sum$) $\neq \varnothing$ entonces para cualquier modelo M de $\sum$ tenemos que

$Base(\sum) \subseteq M$ y si $Any(\sum) \neq \varnothing$ entonces $Any(\sum) \cap M \neq \varnothing$. Así, la siguiente observación es pertinente:

Obs. 1. Sí $\sum$ es una ($\leq$2)-FC satisfactible entonces para cualquier literal $l \in (Base(\sum) \cup Any(\sum))$, hay un modelo $M \in$ SAT($\sum$) tal que $l \in M$.

En términos del grafo $G_{\sum}$, dado un nodo (literal) *l*, $T(l)$ es el conjunto de vértices $G_{\sum}$ que pueden ser alcanzados por *l*. Un camino de *l* a cualquier nodo *v* de $G_{\sum}$ es una secuencia de vértices $v_0v_1, v_1v_2,\ldots, v_{k-1}v_k$ donde $l = v_0$ y $v_k = v$. El tamaño del recorrido es *k*. Un recorrido simple es un recorrido tal que $v_o, v_1,\ldots,v_k$ son todos diferentes. Si $T(l)$ es contradictorio, entonces hay al menos un recorrido de *l* donde dos nodos contradictorios: *x* y $\overline{x}$ aparecen en ese camino, decimos que tal camino es contradictorio.

Dada una formula $\sum$ en la clase ($\leq$2)-FC con *m* cláusulas y *n* variables, nos referimos con cerradura al procedimiento que obtiene: $T(x)$ y $T(\overline{x})$, $\forall x \in v(\sum)$. El procedimiento de cerradura tiene una complejidad en tiempo de orden polinomial, de hecho, la complejidad es $O(n \cdot m)$.

Después de calcular las cerraduras: $T(l)$, $\forall\, l \in Lit(\sum)$, también obtenemos los conjuntos de literales: $Base(\sum)$, $Cont(\sum)$ y $Any(\sum)$. Más aún, cuando |SAT($\sum$)| = 1 en el conjunto $Base(\sum)$ es el único modelo de $\sum$ o, si $\sum$ es insatisfactible, $Cont(\sum)$ tiene un par complementario de literales. Así, verificar la satisfactibilidad de $\sum$ se hace en tiempo lineal cuando $\sum$ es una ($\leq$2)-FC. Además, si $\sum$ es satisfactible, la construcción de un modelo *M* of $\sum$ puede ser hecho en tiempo polinomial con el mismo procedimiento de cerradura.

Lema 1. Sea $\sum$ satisfactible ($\leq$2)-FC y sea *c* una ($\leq$2)-cláusula, entonces la comprobación de la satisfactibilidad de $(\sum \wedge c)$ se hace en tiempo polinomial.

Prueba: Como se discutió arriba, El procedimiento de cerradura checa la satisfactibilidad de cualquier ($\leq$2)-FC en tiempo polinomial. Como $(\sum \wedge c)$ es una ($\leq$2)-FC, el lema cumple con eso.

De hecho, si conocemos $T(l)$, $\forall l \in Lit(\sum)$, entonces podemos revisar fácilmente si alguna cláusula *c* es contradictoria con $\sum$, ya que si *c* es contradictoria con $\sum$ entonces $\forall l \in c,\ l \in Cont(\sum)$. Además, si conocemos $Cont(\sum)$ y *F* es una nueva 2-FC, la revisión de la satisfactibilidad de $(\sum \wedge F)$ se hace en tiempo lineal sobre la longitud de *F*.

Ejemplo. Sea $\sum_1=\{c_1=(x_1, x_2), c_2= (x_1, x_3), c_3= (\overline{x_2},\overline{x_3}), c_4= (\overline{x_2},\overline{x_4}), c_5= (\overline{x_2}, x_4)\}$, con aristas representativas:

$c_1: \overline{x_1} \rightarrow x_2$; $\overline{x_2} \rightarrow x_1$, $c_2: \overline{x_1} \rightarrow x_3$; $\overline{x_3} \rightarrow x_1$,

$c_3: x_2 \rightarrow \overline{x_3}$; $x_3 \rightarrow \overline{x_2}$, $c_4: x_2 \rightarrow \overline{x_4}$; $x_4 \rightarrow \overline{x_2}$, $c_5: x_2 \rightarrow \overline{x_4}$; $x_4 \rightarrow \overline{x_2}$.

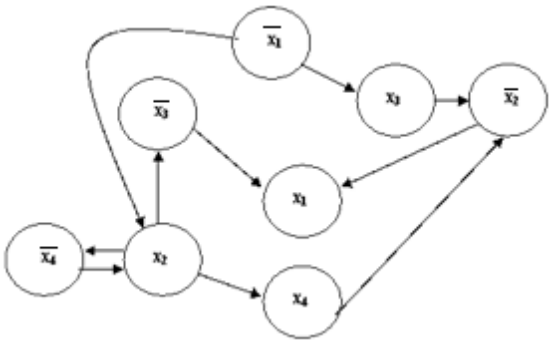


Fig. 11. Grafo de dependencia para la fórmula $\sum_1$

En la Fig. 11, se presenta el grafo de dependencia $G_{\sum 1}$ para la fórmula $\sum_1$. Los diferentes conjuntos $T(l)$, $\forall l \in Lit\ (\sum)$ son mostrados así como: $Base(\sum_1)$, $Cont(\sum_1)$ and $Any(\sum_1)$.

$T(x_1)=\{x_1\}$; $T(\overline{x_1})=\{\overline{x_1}, x_2, x_3, \overline{x_3}, \overline{x_4}, \overline{x_2}, x_1, x_4\}$

$T(x_2)=\{x_2, \overline{x_3}, \overline{x_4}, x_4, x_1, \overline{x_2}\}$; $T(\overline{x_2})=\{\overline{x_2}, x_1\}$

$T(x_3)=\{x_3, \overline{x_2}, x_1\}$; $T(\overline{x_3})=\{\overline{x_3}, x_1\}$

$T(x_4)=\{x_4, \overline{x_2}, x_1\}$; $T(\overline{x_4})=\{\overline{x_4}, \overline{x_2}, x_1\}$.

$T(\overline{x_1})$ y $T(x_2)$ son los únicos conjuntos contradictorios, así $Cont(\sum)=\{\overline{x_1}, x_2\}$, $Base(\sum)=\{x_1, \overline{x_2}\}$ y $Any(\sum)=\{x_3, \overline{x_3}, x_4, \overline{x_4}\}$. Y por tanto la fórmula $\sum$ es satisfactible. Todo modelo de $\sum$ debe contener a las literales de Base($\sum$), las demás literales serán tomadas del conjunto Any, evitando tomar una par de literales complementarias.

3.5 Una Heurística para Construir 3-Coloreos Propios Basado en 2-SAT

El algoritmo que presentamos en esta sección fue ya publicado por nosotros en [23] en donde se puede revisar los detalles de la determinación de la complejidad en tiempo de esta propuesta, así como algunas implicaciones algorítmicas de la propuesta.

Sea $G = (V,E)$ un grafo simple. Supongamos un orden sobre los nodos de V y sobre los cuales se aplica la búsqueda en profundidad. La aplicación de la búsqueda en profundidad sobre G construye un nuevo grafo G' llamado grafo a lo profundo, así como un grafo acíclico (árbol) T_G. Sea $C = \{C_1, C_2, \ldots, C_k\}$ el conjunto de ciclos básicos encontrados durante la búsqueda en profundidad de G.

Asimismo, durante la búsqueda en profundidad, dos conjuntos C_o y C_e se forman: C_o contiene los ciclos básicos de longitud impar en G y C_e los ciclos básicos de longitud par de G. El siguiente procedimiento reduce el problema de 3-coloreo de G a determinar la satisfactibilidad de una F_G que esta en 2-FC.

Procedimiento 3_Col(G)

1. Compruebe si T_G tiene cualquiera de las condiciones mostradas en la sección anterior con el fin de ver si T_G es 3 coloreable. Si G no tiene esas condiciones, continúe con el paso siguiente.

2. Sea $C_E \subseteq C_e$ el conjunto de ciclos de longitud par que intersectan con cualquier ciclo impar en G', i.e.,

$CE = \{C_i \in C_e : C_i$ es de longitud par y $\exists\ C_j \in C_o, C_i \cap C_j <> \emptyset\}$.

3. Asumimos que $k = |C_o|$, hay k ciclos básicos impares en G'. Sea $D_e = \{b_{e1},..., b_{ek}\}$ el conjunto de k aristas de retroceso formadas a partir de C_o, tales que cada $b_{ej} \in D_e$ es la arista de retroceso de $C_j \in C_o$, $j = 1,..., k$.

4. Para cada $b_{ej} = \{x_j, y_j\} \in D_e$, $j = 1,..., k$ se construyen las dos cláusulas binarias:

$$A_j = (x_j \vee y_j) \wedge (\bar{x_j} \vee \bar{y_j}).$$

5. Para cada par diferente de aristas de retroceso en D_e: $b_{el} = \{x_l, y_l\}$ and $b_{ej} = \{x_j, y_j\}$, $l <> j$ donde uno de sus puntos finales (x_j o y_j) es adyacente a (x_l o y_l), por ejemplo, el supuesto de que x_l y x_j son nodos adyacentes, se construye la siguiente cláusula binaria: $B_j = (\bar{x_l} \vee \bar{x_j})$.

6. Para cada arista de retroceso $e_l = \{x_l, y_l\}$ de un ciclo en C_E, se construye la siguiente cláusula binaria: $E_l = (\bar{x_l} \vee \bar{y_l})$.

7. Sea F_G la 2-FC formada por la conjunción de los $A'_i s$ y $B'_j s$ y $E'_l s$ construidas en los pasos anteriores.

8. Determinar si F_G es satisfactible o no. Si F_G es satisfactible entonces G es 3-coloreable, de lo contrario nuestra heurística no puede determinar una adecuada 3-coloración de G.

9. Las variables definidas como verdaderas en un modelo de F_G se corresponden con los nodos que se colorean con el tercer color W. Si tales nodos son eliminados de G el subgrafo restante es bipartito y sus nodos se puede colorear con los dos colores básicos: {G, R}.

Nuestra propuesta algorítmica es consistente. En el sentido de que si la fórmula construida por nuestro algoritmo es satisfactible entonces el grafo es 3-coloreable, y de hecho cualquier modelo de la fórmula así construida presenta una manera de 3-coloreo del grafo. La testificación de esta propiedad se realiza en base al teorema siguiente.

Teorema. Todo modelo de la fórmula F_G construida por el procedimiento 3_Col(*G*) proporciona un 3-coloreo de *G*.

Cláusulas	Significado
A_j's	Para cada arista de retroceso de un ciclo impar solo uno de sus dos nodos debe ser coloreado con *W.*
B_j's	Cualquier par de nodos con color *W* no debe ser adyacente.
E_j's	Los nodos de una arista de retroceso de ciclo de un ciclo par no están asignados con el color *W.*

Tabla 1. Relación entre el tercer color y la satisfactibilidad de las cláusulas.

Demostración

Si hay una asignación *s* que satisface F_G, las variables de *s* a las que se les asigna el valor verdadero se corresponden con los nodos a quienes se les asigna el tercer color W, mientras que las variables a las que se les asigna el valor falso se pueden colorear con los dos colores básicos.

El conjunto de cláusulas en $A'_j s$ garantiza que todos los ciclos impares tienen un color W asignado a uno de sus nodos y por lo tanto, todo ciclo impar ha sido 3-coloreable, mientras que el conjunto de cláusulas en $B'_j s$ y $E'_j s$ prohiben la asignación del tercer color a dos nodos adyacentes.

Por otro lado, si F_G es insatisfactible entonces no hay manera de asignar el color W a uno de los extremos de una arista de retroceso de un ciclo impar y, al mismo tiempo, garantizar que dos nodos de color con W no son adyacentes. Por lo tanto, **3_Col(G)** no puede construir una adecuada 3-coloración de *G*. Aquí se tiene la ventaja que decidir la satisfactibilidad de una 2-FC puede realizarse en tiempo polinomial, basado en el cálculo de la cerradura transitiva de las variables, tal y como se presentó en la sección anterior.

Ejemplo

Tomaremos como ejemplo para aplicar este algoritmo, el grafo de Petersen (Fig. 12).

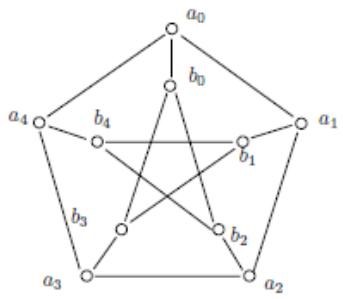


Fig.12. Grafo de Petersen.

Al aplicar sobre este grafo el procedimiento de búsqueda a lo profundo (*dfs*) obtenemos el árbol mostrado en la Fig. 13.

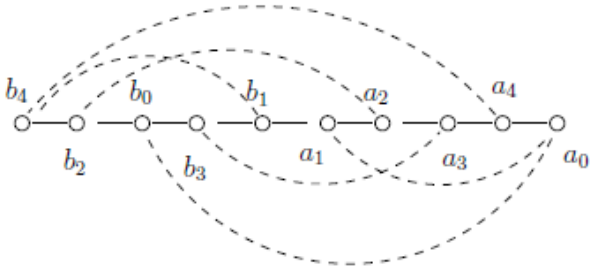


Fig. 13. Árbol generador del grafo de Petersen.

Con esto obtenemos los siguientes conjuntos:

Ciclos básicos de longitud impar C_o

C_o = {(a1, a2, a3, a4, a0),(b4, b2, b0, b3, b1, a1, a2, a3, a4),(b3,b1, a1, a2, a3),(b4, b2, b0, b3, b1)}

Ciclos básicos de longitud par C_e

C_e = {(b0, b3, b1, a1, a2, a3, a4, a0),(b2, b0, b3, b1, a1, a2)}

Aristas de retroceso de los ciclos impares D_e

D_e = {{a1, a0}, {b3, a3}, {b4, a4}, {b1, b4}}

Ciclos pares intersectados C_E

C_E = {(b0, b3, b1, a1, a2, a3, a4, a0), (b2, b0, b3, b1, a1,a2)}

Con lo anterior se generan las cláusulas:

$\boldsymbol{A_i}$ generadas a partir de las aristas de retroceso $\boldsymbol{D_e}$ tal como se describe en el algoritmo.

$A_1 = (a1 \vee a0) \wedge (\overline{a1} \vee \overline{a0})$,
$A_2 = (b3 \vee a3) \wedge (\overline{b3} \vee \overline{a3})$,
$A_3 = (b4 \vee a4) \wedge (\overline{b4} \vee \overline{a4})$,
$A_4 = (b1 \vee b4) \wedge (\overline{b1} \vee \overline{b4})$.

$\boldsymbol{B_i}$ generadas a partir de las aristas de retroceso $\boldsymbol{D_e}$ tal como se describe en el algoritmo.

$B_1 = (\overline{a0} \vee \overline{a4})$,
$B_2 = (\overline{b1} \vee \overline{b3})$,
$B_3 = (\overline{b1} \vee \overline{a1})$.

$\boldsymbol{E_i}$ obtenidas de $\boldsymbol{C_E}$ como se describe en el algoritmo.

$E_1 = (\overline{b0} \vee \overline{a0})$,
$E_2 = (\overline{b2} \vee \overline{a2})$.

Así mismo se deben satisfacer las restricciones restantes que impiden que dos nodos adyacentes tengan el mismo color, en conjunto con nuestra $\boldsymbol{F_G}$ generada.

$$\boldsymbol{F_G} = A1 \wedge A2 \wedge A3 \wedge A4 \wedge B1 \wedge B2 \wedge B3 \wedge E1 \wedge E2$$

Así, $\boldsymbol{F_G}$ es satisfactible, con el siguiente modelo:

$a0 = 1$, $a1 = 0$, $a2 = 0$, $a3 = 0$, $a4 = 0$ y $b0 = 0$, $b2 = 0$, $b3 = 1$, $b4 = 1$.

Como se menciona en el algoritmo el 3-coloreo es obtenido coloreando las variables a las cuales se les ha asignado el valor 1 o verdadero es decir a0, b3 y b4 con el tercer color W. Borramos estos nodos y coloreamos el subgrafo bipartito restante con los dos colores básicos {G,R}.

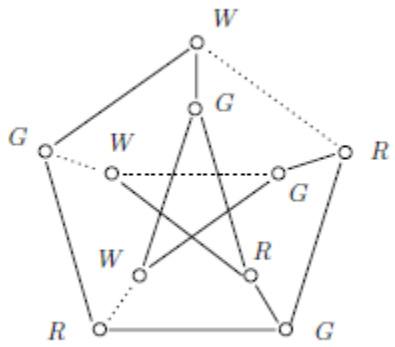


Fig. 14. Grafo de Petersen con la asignación de 3 colores: W, G y R

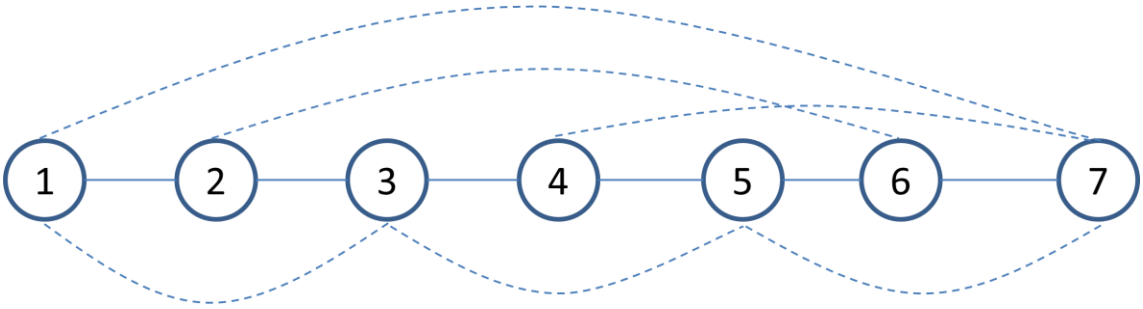
Este algoritmo es analizado a continuación para detectar sus limitaciones y presentar una nueva propuesta.

3.6 Análisis del Procedimiento del 3-coloreo

En el subcapítulo anterior se dejó comentado que la propuesta del procedimiento de 3 coloreo basándose en cláusulas 2-FC sería analizada más a fondo para determinar sus alcances y limitaciones.

Lo que se esperaba encontrar era un grafo, que de antemano se conociera que sí es 3-coloreable, pero que al aplicar el procedimiento 3_Col(*G*), resultara que la fórmula F_G no fuera satisfactible, es decir, que a pesar de ser dicho grafo *G,* 3 coloreable, el procedimiento arrojaría como resultado que no se puede 3-colorear.

A continuación se presenta un grafo, en el cual, nos encontramos con dicha situación:



Aplicando el procedimiento de búsqueda a lo profundo obtenemos lo siguiente:

$\boldsymbol{C_o}$ = {(1, 2, 3), (3, 4, 5), (5, 6, 7), (1,2, 3, 4, 5, 6, 7), (2, 3, 4, 5, 6)} Ciclos básicos impares.

$\boldsymbol{C_e}$ = {(4, 5, 6, 7)} Ciclos básicos pares

$\boldsymbol{D_e}$ = {{1,3}, {3,5}, {5,7}, {2,6}, {1,7} } Aristas de retroceso de los ciclos básicos impares.

$\boldsymbol{C_E}$ = {(4, 5, 6, 7)} Ciclos básicos pares que intersectan con algún ciclo impar.

Al aplicar el procedimiento *3_Col(G)* obtenemos las siguientes cláusulas:

$\boldsymbol{A_1}$= (1v3) ∧ ($\bar{1}$v$\bar{3}$)

$\boldsymbol{A_2}$= (3v5) ∧ ($\bar{3}$v$\bar{5}$)

$\boldsymbol{A_3}$= (5v7) ∧ ($\bar{5}$v$\bar{7}$)

$\boldsymbol{A_4}$= (2v6) ∧ ($\bar{2}$v$\bar{6}$)

$\boldsymbol{A_5}$= (1v7) ∧ ($\bar{1}$v$\bar{7}$)

Y las cláusulas que aseguran que ningún nodo adyacente sea del mismo color:

$\boldsymbol{B_1}$ = ($\bar{1}$v$\bar{2}$)

$\boldsymbol{B_2}$ = ($\bar{2}$v$\bar{3}$)

$\boldsymbol{B_3}$ = ($\bar{3}$v$\bar{4}$)

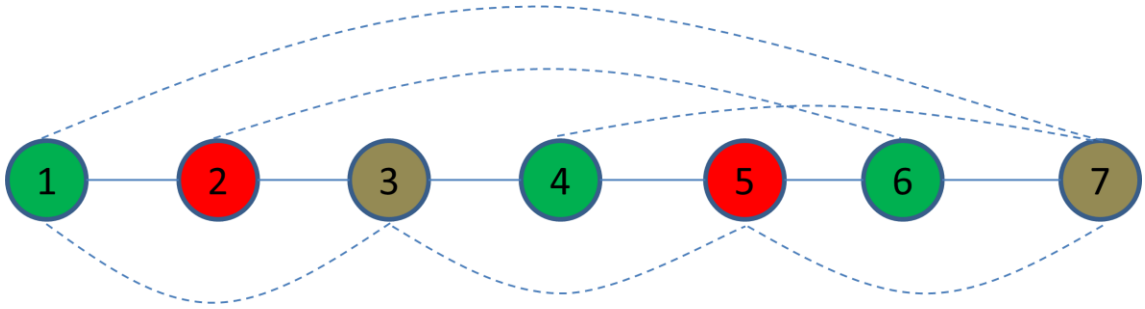
$\boldsymbol{B_4}$ = ($\bar{4}$v$\bar{5}$)

$\boldsymbol{B_5}$ = ($\bar{5}$v$\bar{6}$)

$\boldsymbol{B_6}$ = ($\bar{6}$v$\bar{7}$)

$\boldsymbol{B_7}$ = ($\bar{7}$v$\bar{4}$)

Las cláusulas anteriores no se pueden satisfacer todas a la vez, por lo cual el procedimiento arrojaría que no existe un 3-coloreo propio, sin embargo este grafo si es 3-coloreable.



Con lo cual sabemos ahora que el procedimiento *3_Col(G)* no cubre todos los casos para encontrar un 3-coloreo, esto es, nuestro procedimiento no es completo.

IV) Coloreo de Grafos y el problema de Satisfactibilidad

Una vez detectado que la propuesta algorítmica de la sección anterior no es completa, se procedió a trabajar con una nueva propuesta basada en usar cláusulas *n*-conjuntivas, en lugar de sólo 2-conjuntivas.

4.1 Reduciendo el Problema del Coloreo de Grafos al Problema SAT

Es conocido que el problema SAT es NP-completo cuando no hay restricciones sobre las longitudes y forma de las cláusulas. En nuestro caso, tendremos fórmulas con cláusulas de dos tipos:

Tipo A) La longitud de cláusulas es impar y todas las literales aparecen de forma positiva.

Tipo B) Cláusulas binarias donde ambas literales aparecen de forma negativa.

Por la singularidad del tipo de fórmulas a resolver, proponemos un algoritmo para resolver SAT para este tipo de fórmulas. Sea C el conjunto de cláusulas impares tanto simples como compuestas

Sea frec(X_j) la frecuencia de la variable X_j, es decir el número de veces que aparece en la fórmula conjuntiva.

Dada una literal X_j que aparece en una fórmula, sea $T(X_j)$ la cerradura reflexiva y transitiva de X_j, calculada como fue mostrado en capítulos anteriores. $T(X_j)$={ X_k: donde X_k son los nodos adyacentes a X_j}.

Procedimiento Sat(C)

1. Tomamos la cláusula min{$|C_i|$}, es decir la cláusula con la longitud mínima.

2. Ordenar los elementos X_j de min{$|C_i|$} de mayor a menor según su frec(X_j).

3. Para cada elemento X_j de min{$|C_i|$}, tomados en el orden anterior:

   Se crea un nodo hijo $nodo_j$ a partir del nodo actual y se asigna a X_j el valor verdadero

   Del conjunto C eliminamos las cláusulas que se satisfacen con esta asignación

   Si el conjunto de cláusulas restantes es igual a vacío entonces la fórmula es Satisfactible.

Sino

Eliminar de las cláusulas restantes las variables que no pueden llevar el tercer color, es decir, eliminar las variables que aparecen en $T(X_j)$

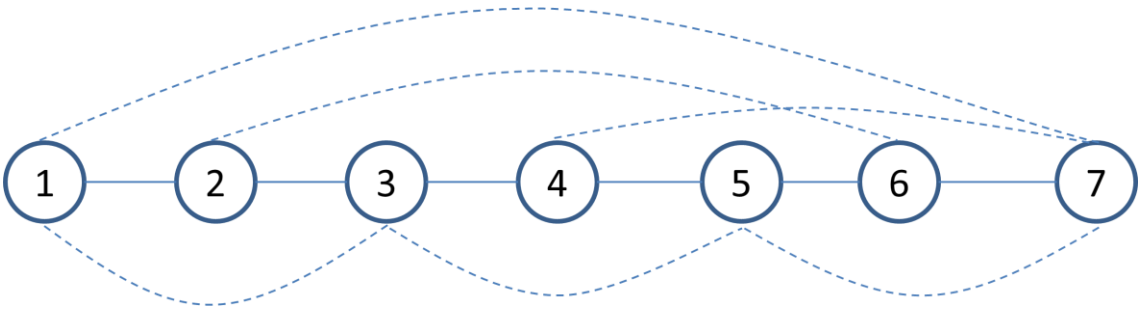
Si en alguna de las cláusulas se eliminan todas las variables entonces esta asignación no es válida, aplicar retroceso regresando al nodo padre del nodo actual.

4. Si no se han agotado todas las variables de la cláusula actual regresamos al paso 1. En otro caso no existe asignación que satisfaga a C y a $T(X_i)$ en cuyo caso, se puede elegir uno de los dos casos:

a) Terminar indicando que el grafo no es 3 (o k) coloreable.

b) Incrementar el color actual en 1 para satisfacer el conjunto C y a $T(X_i)$ con un color mas.

A continuación presentamos un ejemplo de cómo trabaja este algoritmo:



Sea C el conjunto de cláusulas impares tanto simples como compuestas:

$A_1 = X_1 \vee X_2 \vee X_3$
$A_2 = X_3 \vee X_4 \vee X_5$
$A_3 = X_5 \vee X_6 \vee X_7$
$A_4 = X_1 \vee X_2 \vee X_3 \vee X_4 \vee X_5 \vee X_6 \vee X_7$
$A_5 = X_2 \vee X_3 \vee X_4 \vee X_7 \vee X_6$
$A_6 = X_4 \vee X_5 \vee X_7$
$A_7 = X_3 \vee X_4 \vee X_7 \vee X_6 \vee X_5$
$A_8 = X_2 \vee X_3 \vee X_4 \vee X_5 \vee X_6$

Sea frec(X_j) la frecuencia de la variable X_j, es decir el número de veces que aparece en las cláusulas.

frec(X_1) = 2
frec(X_2) = 4
frec(X_3) = 6
frec(X_4) = 6
frec(X_5) = 6
frec(X_6) = 5
frec(X_7) = 5

Sea $T(X_j)$ la cerradura de X_j, $T(X_j)$={ X_j, X_k} donde X_k son los nodos adyacentes a X_j.

$T(X_1)=\{X_1, \overline{X}_2, \overline{X}_3, \overline{X}_7\}$

$T(X_2)=\{X_2, \overline{X}_1, \overline{X}_3, \overline{X}_6\}$

$T(X_3)=\{X_3, \overline{X}_1, \overline{X}_2, \overline{X}_4, \overline{X}_5\}$

$T(X_4)=\{X_4, \overline{X}_3, \overline{X}_5, \overline{X}_7\}$

$T(X_5)=\{X_5, \overline{X}_3, \overline{X}_4, \overline{X}_6, \overline{X}_7\}$

$T(X_6)=\{X_6, \overline{X}_2, \overline{X}_5, \overline{X}_7\}$

$T(X_7)=\{X_7, \overline{X}_1, \overline{X}_6, \overline{X}_4, \overline{X}_5\}$

Tomamos la cláusula min{|C_i|}, la cláusula con la longitud mínima que pertenece a C.

En este caso tomamos: $A_1 = X_1 \vee X_2 \vee X_3$

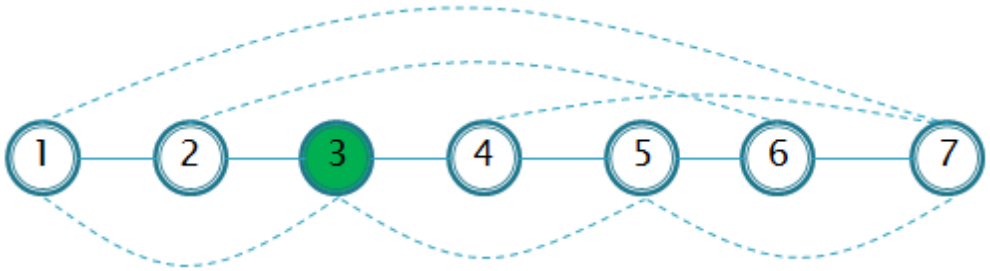
Ordenar los elementos X_j de min{|C_i|} de mayor a menor según su frec(X_j), quedando como sigue:

$$A_1 = X_3 \vee X_2 \vee X_1$$

Para cada elemento X_j de min{|C_i|}, tomados en el orden anterior:

Se crea un nodo hijo $nodo_j$ a partir del nodo actual y se asigna a X_j el valor verdadero.

En este caso es X_3 por lo tanto el grafo queda de la siguiente manera:



Del conjunto *C* eliminamos las cláusulas que se satisfacen con esta asignación:

~~$A_1 = X_3 \vee X_2 \vee X_1$~~
~~$A_2 = X_3 \vee X_4 \vee X_5$~~
$A_3 = X_5 \vee X_6 \vee X_7$
~~$A_4 = X_1 \vee X_2 \vee X_3 \vee X_4 \vee X_5 \vee X_6 \vee X_7$~~
~~$A_5 = X_2 \vee X_3 \vee X_4 \vee X_7 \vee X_6$~~
$A_6 = X_4 \vee X_5 \vee X_7$
~~$A_7 = X_3 \vee X_4 \vee X_7 \vee X_6 \vee X_5$~~
~~$A_8 = X_2 \vee X_3 \vee X_4 \vee X_5 \vee X_6$~~

Si el conjunto de cláusulas restantes es igual a vacío entonces la fórmula es SAT.

Como el conjunto de cláusulas aún no es vacío, ya que quedan dos cláusulas: A_3 y A_6, seguimos:

Eliminar de las cláusulas restantes las literales adyacentes, es decir, las literales que con el signo inverso de las literales que aparecen en $T(X_j)$.

$T(X_3)=\{X_3, \overline{X}_1, \overline{X}_2, \overline{X}_4, \overline{X}_5\}$, por lo tanto tenemos que eliminar de A_3 y A_6 las variables X_1, X_2, X_4, y X_5

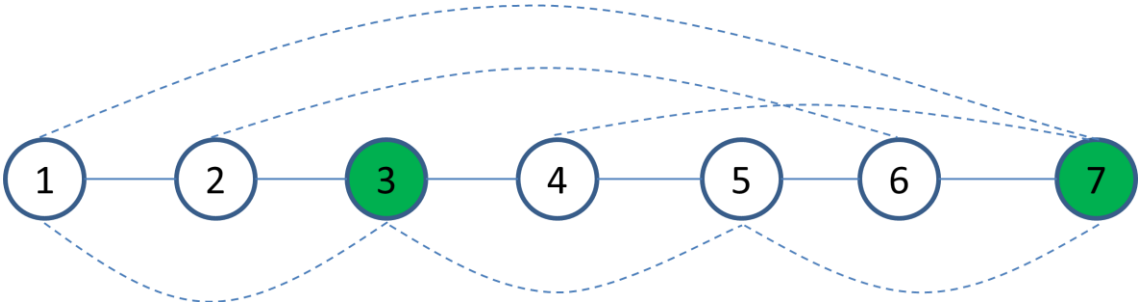
$A_3 = $ ~~X_5~~ $\vee X_6 \vee X_7$
$A_6 = $ ~~X_4~~ $\vee$ ~~X_5~~ $\vee X_7$

Como en ninguna de las dos cláusulas se eliminaron todos los elementos y aún no se han agotado todas las variables de la cláusula actual A_1 entonces regresamos al paso 1.

Tomamos la cláusula min{$|C_i|$} en este caso sería:

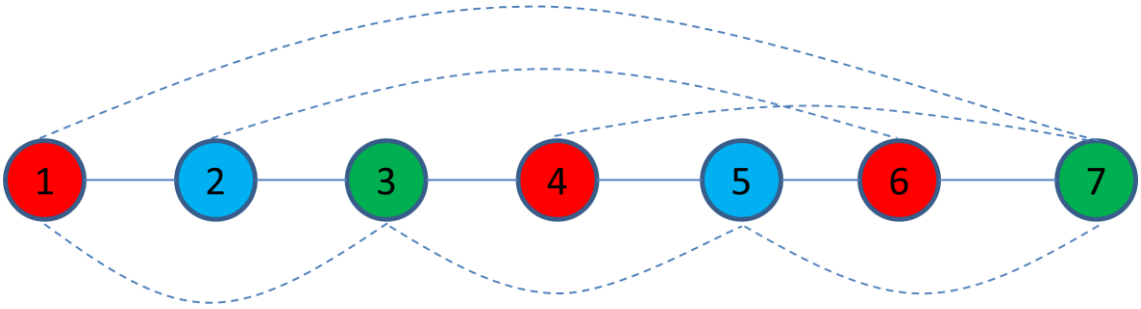
$$A_6 = X_7$$

Al ser unitaria automáticamente se le asigna el valor verdadero a la variable X_7 por lo cual se colorea con el tercer color quedando el grafo como sigue:



El siguiente paso es eliminar las cláusulas que se satisfacen con esta asignación y como ambas cláusulas A_3 y A_6 se satisfacen, el conjunto de cláusulas C queda vacio, concluyéndose que todas las cláusulas fueron satisfechas por lo tanto el problema es SAT.

Por último solo resta colorear el grafo bipartito restante con dos colores, quedando finalmente 3 coloreado de la siguiente manera:



4.2 Un algoritmo para colorear Grafos en base al problema SAT

Sea $G = (V,E)$ un grafo simple. Supongamos que se aplica una búsqueda en profundidad sobre los nodos de V con la estrategia de elegir siempre al nodo de menor grado y de entre ellos, el que tenga la etiqueta con valor menor en el paso de selección de visitar nodos aún no visitados.

La aplicación de la búsqueda en profundidad sobre G construye un nuevo grafo a profundidad G' y un árbol abarcador T_G del grafo inicial G. Sea $C = \{C_1, C_2,...,C_k\}$ el conjunto de ciclos básicos encontrados durante la búsqueda en profundidad en G.

Durante la búsqueda en profundidad, dos conjuntos C_o y C_e se pueden formar: C_o conteniendo los ciclos básicos de longitud impar en G y C_e conteniendo los ciclos básicos de longitud par de G.

Sea C_o' el conjunto de ciclos compuestos de longitud impar que se forman al aplicar el operador diferencia simétrica entre cada par de ciclos básicos: C_i, $C_j \in$ C. Proponemos ahora un algoritmo para trasladar cualquier grafo a una forma conjuntiva del tipo que se consideró en el capítulo anterior. Es decir, presentamos ahora una reducción algorítmica de un grafo a una fórmula conjuntiva.

Procedimiento que reduce el problema de 3-coloreo de G para determinar la satisfactibilidad de una fórmula conjuntiva F_G.

1. Compruebe si *G*' tiene cualquiera de las condiciones mostradas en la sección anterior con el fin de ver si *G* es 3-coloreable. Si *G* es 3-coloreable termine y utilice el algoritmo de 3-coloreo, en otro caso continúe con el paso siguiente.

2. Asumimos que $k = |C_o \cup C_o'|$, hay *k* ciclos impares tanto básicos como compuestos,

Para cada ciclo C_j , $j = 1, \ldots, k$

se generan las cláusulas:

$A_j = X_1 \vee \ldots \vee X_n$

Donde *n* es el número de nodos que hay en el ciclo C_j y cada $X_1,\ldots, X_n$ son las variables de los nodos que conforman el ciclo C_j.

3. Sea $E = \{E_1,\ldots,E_p\}$ el conjunto de *p* aristas del grafo.

Para cada arista $e_i = \{X_l, Y_l\} \in$ E, se construye la cláusula binaria:

$$d_i = (\overline{X}_l \vee \overline{Y}_l).$$

5. Sea F_G la FC formada por la conjunción de los *Aj*'s y d_i's construida en los pasos anteriores.

6. Determinar si F_G es satisfactible o no. Si F_G es satisfactible entonces *G* es 3-coloreable, de lo contrario nuestra heurística no puede determinar una adecuada 3-coloración de *G*.

Nótese que el tipo de fórmulas generadas por nuestro algoritmo es del tipo descrito en la sección anterior donde se tienen cláusulas con sólo literales en forma positiva y cláusulas con solo literales en forma negativa, por tanto no existen cláusulas donde haya literales mezcladas, de forma y positiva y negativa en la misma cláusula.

Además, se puede demostrar que cualquier modelo de la fórmula así construida determina un 3-coloreo para el grafo original, de tal forma que las variables que aparecen de forma positiva en un modelo de F_G se corresponden con los nodos que se colorean con el tercer color W. Si tales nodos son eliminados de *G* el subgrafo restante es bipartito y sus nodos se pueden colorear con los dos colores básicos: {G, R}.

Teorema: Todo modelo de F_G determina un 3-coloreo para *G*.

Demostración

Si existe una asignación *s* que satisface a F_G. Los variables a las que se les asigna el valor verdadero corresponden a los nodos de G que se colorean con el tercer color W. Por otro lado las variables a las que se les asigna el valor falso corresponden a los nodos coloreados con los 2 colores restantes.

El conjunto de cláusulas A_j'*s* conteniendo sólo literales positivas codifican a los ciclos de longitud impar del grafo *G*, incluyendo los ciclos compuestos impares. Como las cláusulas A_j'*s* se satisfacen entonces hay al menos un nodo para cada ciclo impar de *G* que toma el 3er. color, lo que asegura que todo ciclo (básico y compuesto) de longitud impar tenga al menos un nodo con el 3er. color.

Al eliminarse de *G* los nodos con el 3er. color, se forma un subgrafo *G*' que o no tiene ciclos o solo contiene ciclos pares, ya que por cada ciclo impar (básico y compuesto) de *G* al quitárseles un nodo entonces sólo se forman ciclos de longitud par. En cualquier caso, el grafo *G*' es ahora bipartito y puede ser dos coloreable.

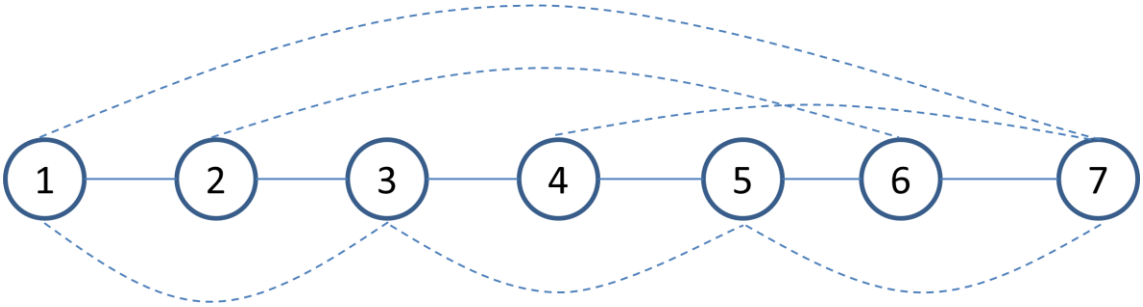
Por otro lado, el satisfacer al conjunto de cláusulas d_i's asegura que no se asigne el 3er. color a un par de nodos adyacentes.

Por último si la fórmula F_G no se satisface, es porque no hay forma de asignar el tercer color a las variables de los cláusulas A_j'*s* y asegurar al mismo tiempo que no es adyacente a una variable ya asignada previamente con el mismo color. Esto significa que no tenemos forma de colorear uno de los nodos de todo ciclo impar (básico y compuesto) de G con el 3er. color.

4.3 Análisis de Nuestra Propuesta Algorítmica

Ejemplo

Aplicando este nuevo procedimiento al grafo que se usó para mostrar la no completes de nuestra primera propuesta algorítmica, tenemos:



Co = {*(1, 2, 3), (3, 4, 5), (5, 6, 7), (1,2, 3, 4, 5, 6, 7), (2, 3, 4, 5, 6)*}

Ce = {*(4, 5, 6, 7)*}

Co' = {*(2, 3, 4, 7, 6), (4, 5, 7), (3, 4, 7, 6, 5)*} Conjunto de ciclos impares compuestos.

Además de las cláusulas que aseguran que ningún par de nodos adyacentes tenga el mismo color, tenemos las siguientes cláusulas:

$A_1 = X_1 \vee X_2 \vee X_3$

$A_2 = X_3 \vee X_4 \vee X_5$

$A_3 = X_5 \vee X_6 \vee X_7$

$A_4 = X_2 \vee X_3 \vee X_4 \vee X_5 \vee X_6$

$A_5 = X_1 \vee X_2 \vee X_3 \vee X_4 \vee X_5 \vee X_6 \vee X_7$

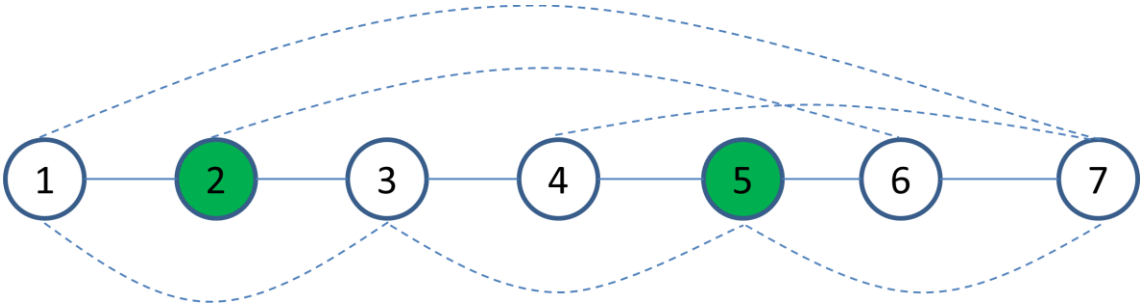
$A_6 = X_2 \vee X_3 \vee X_4 \vee X_7 \vee X_6$

$A_7 = X_4 \vee X_5 \vee X_7$

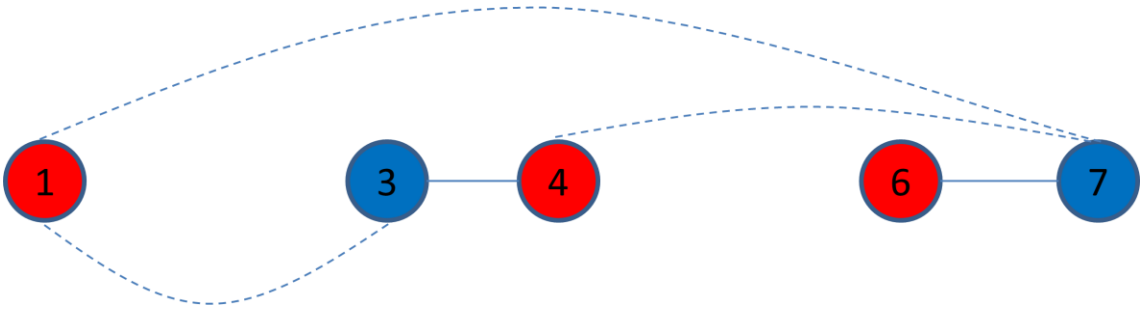
$A_8 = X_3 \vee X_4 \vee X_7 \vee X_6 \vee X_5$

Asignarle valor verdadero a X_5 satisface las cláusulas A_2, A_3, A_4, A_5, A_7 y A_8.

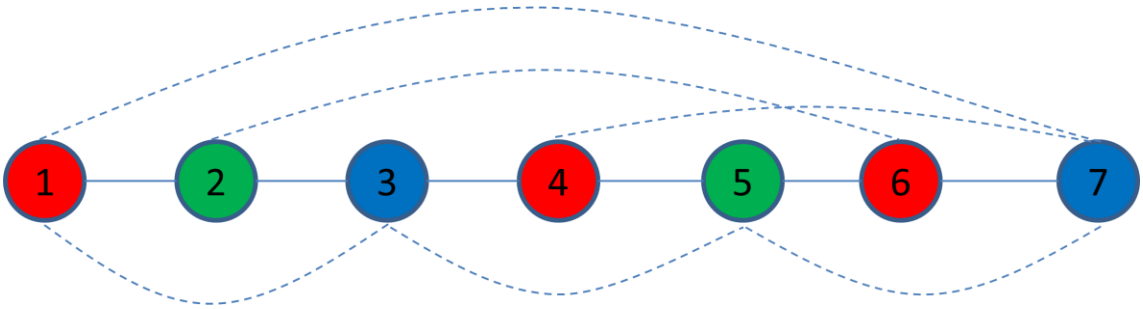
Para satisfacer A_1 y A_6 se asigna a X_2 el valor verdadero.



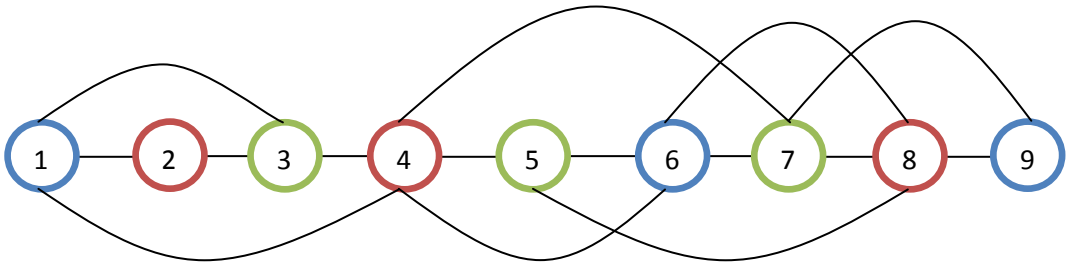
Al eliminar ambos nodos el grafo restante queda como sigue:



Este grafo restante es bipartito, por lo tanto es 2-coloreable. Quedando el grafo 3-coloreado de la siguiente manera:



Ejemplo 2: Este ejemplo se revisará más a detalle incluyendo la propuesta del algoritmo para resolver SAT.



Co = {*(1, 2, 3), (4, 5, 6), (6, 7, 8), (7, 8, 9)*}

Ce = {*(1, 2, 3,4), (4, 5, 6, 7), (5, 6, 7, 8)*}

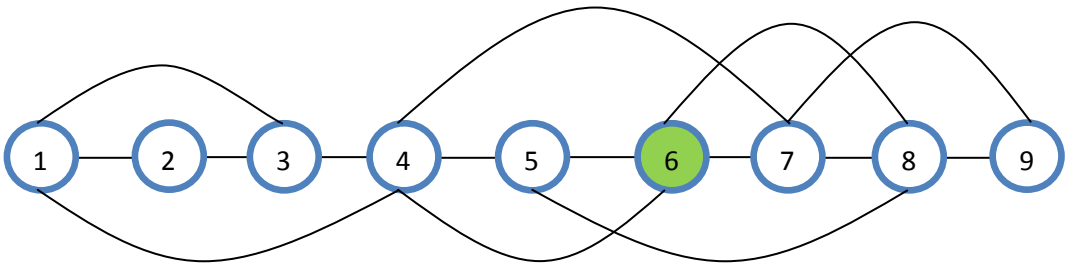
Co' = {*(1, 3, 4), (4, 6, 7), (4, 5, 6, 8, 7), (5, 6, 8), (5, 6, 7, 8, 9)*}

Los ciclos impares tanto simples como compuestos darían como resultado las siguientes cláusulas:

$A_1 = X_1 \vee X_2 \vee X_3$
$A_2 = X_4 \vee X_5 \vee X_6$
$A_3 = X_6 \vee X_7 \vee X_8$
$A_4 = X_7 \vee X_8 \vee X_9$
$A_5 = X_1 \vee X_3 \vee X_4$
$A_6 = X_4 \vee X_6 \vee X_7$
$A_7 = X_4 \vee X_5 \vee X_6 \vee X_7 \vee X_8$
$A_8 = X_5 \vee X_6 \vee X_8$
$A_9 = X_5 \vee X_6 \vee X_7 \vee X_8 \vee X_9$

frec(X_1) = 2, frec(X_2) = 1, frec(X_3) = 2, frec(X_4) = 4, frec(X_5) = 4, frec(X_6) = 6, frec(X_7) = 5
frec(X_8) = 5, frec(X_2) = 2.

La literal con más frecuencia es X_6 por lo tanto satisface el mayor número de cláusulas, así que se le asigna el valor verdadero, con lo cual el nodo respectivo queda coloreado con el tercer color.



Se eliminan las cláusulas que se satisfacen, quedando restantes:

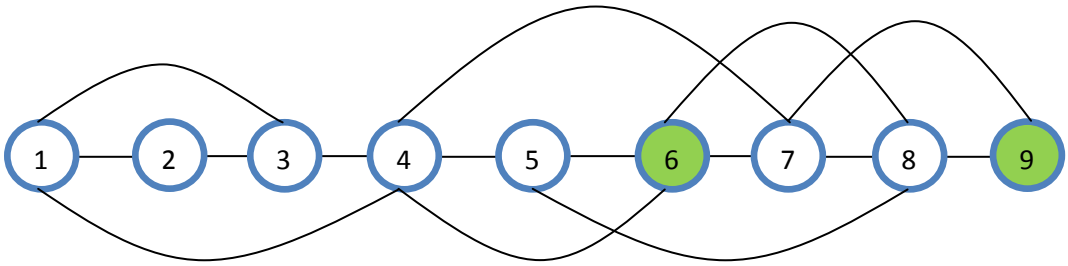
$A_1 = X_1 \vee X_2 \vee X_3$
$A_4 = X_7 \vee X_8 \vee X_9$
$A_5 = X_1 \vee X_3 \vee X_4$

Por la cerradura de X_6, tenemos que $T(X_6) = \{X_6, \overline{X_4}, \overline{X_5}, \overline{X_7}, \overline{X_8}\}$

Eliminando las literales adyacentes a X_6, es decir que no pueden tomar el valor verdadero, de las cláusulas, nos queda como sigue:

$A_1 = X_1 \vee X_2 \vee X_3$
$A_4 = X_9$
$A_5 = X_1 \vee X_3$

De aquí tomando la cláusula de menor tamaño, al ser A_4 unitaria, se le asigna el valor verdadero a X_9 quedando el grafo coloreado de la siguiente manera:

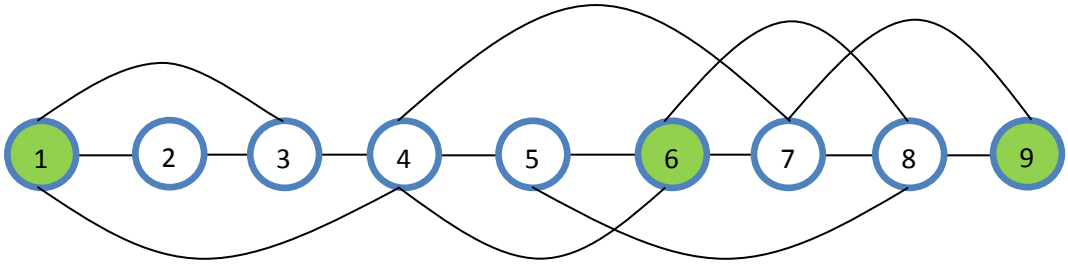


Eliminando las cláusulas que se satisfacen con esta asignación tenemos:

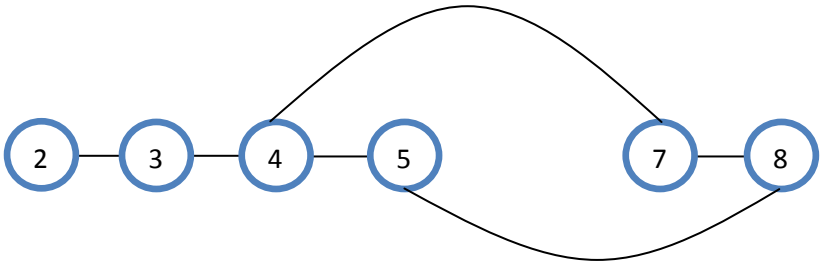
$A_1 = X_1 \vee X_2 \vee X_3$
$A_5 = X_1 \vee X_3$

Por la cerradura de X_9, tenemos que $T(X_9) = \{X_9, X_7, \overline{X_8}\}$, por lo tanto no se elimina ninguna literal de las dos cláusulas anteriores.

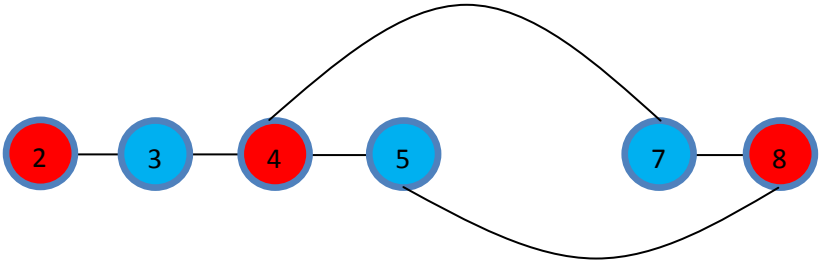
Tomamos la de menor longitud, es decir A_5 como ambas literales X_1 y X_3 tienen la misma frecuencia, le asignamos el valor verdadero a la de menor índice en este caso X_1. Con esta asignación se satisfacen las dos últimas cláusulas quedando el grafo como sigue:



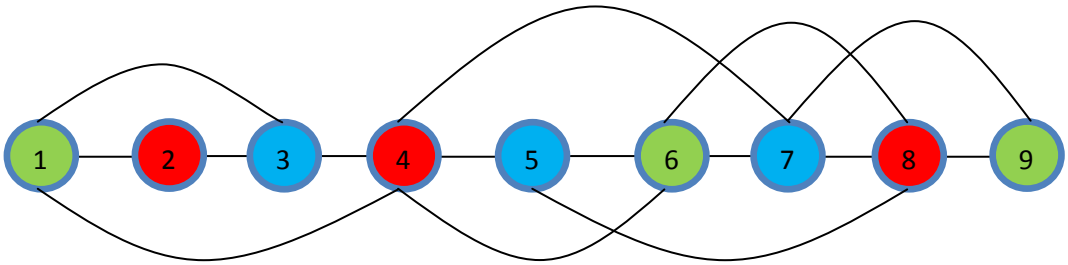
Al eliminar los nodos asignados con el valor verdadero nos queda el siguiente grafo:



El cual es bipartito y puede colorearse con los dos colores restantes así:



Quedando el grafo completo 3-coloreado de la siguiente forma:



V) Conclusiones

El problema del coloreo propio de vértices en un grafo es un problema que mantiene desafíos matemáticos y computacionales actuales. Aún y cuando se llevan ya varias décadas en su análisis y en el desarrollo de métodos de solución, el problema del coloreo, al igual que cualquier problema de la clase de complejidad NP-completo, siempre serán objeto de estudio, y mientras no se responda con precisión la pregunta ¿es P = NP?, habrá necesidad de mejorar las propuestas algorítmicas para resolver los problemas de la clase NP.

En este documento se presentan dos formas diferentes para traducir el coloreo de ciclos de grafos usando cláusulas monótonas. Se demuestra que nuestra primera propuesta no es un método completo pero la determinación de satisfactibilidad de las instancias generadas es realizable en tiempo polinomial, al ser reducido el problema de 3-coloreo a revisar la satisfactibilidad de fórmulas en dos forma conjuntiva.

Nuestra segunda propuesta corrige muchos de los casos que la primera propuesta no puede resolver, pero deja entonces de ser un procedimiento de complejidad polinomial la revisión de la satisfactibilidad de las fórmulas Booleanas construidas. Aún así, las características que presentan las fórmulas Booleanas construidas, de contener sólo cláusulas monótonas (con sólo variables en forma positiva o bien sólo variables en forma negativa) donde no aparecen variables con signos mezclados; positivos y negativos en una misma cláusula, permitió proponer un algoritmo más ad-hoc para la determinación de satisfactibilidad de este tipo de cláusulas.

5.1 Aportaciones

Específicamente, con este trabajo se logró aportar en el área de diseño de algoritmos al proponer un nuevo algoritmo para atacar el problema del 3-coloreo de grafos. Se abordó un nuevo método para la solución del coloreo de grafos basándose en los ciclos básicos que hay en el grafo y reduciendo el coloreo de ciclos al problema SAT. El algoritmo propuesto es novedoso, y además de su aportación en sí mismo para abordar el problema del coloreo de grafos, deja abierta una puerta para continuar atacando este problema sobre la línea propuesta por dicho algoritmo.

Como ya se mencionó, el problema del coloreo de grafos se mapea al problema SAT, por lo que otro aporte fue la propuesta de un nuevo algoritmo para atacar el problema SAT, particularmente para considerar una clase especial de fórmulas Booleanas que se construyen a partir de considerar el coloreo de ciclos en un grafo.

El hecho de que tengamos fórmulas conjuntivas formadas por cláusulas con sólo literales positivas o bien con sólo literales negativas, permite diseñar propuestas específicas para resolver estas instancias del problema SAT (el cual en su expresión general es un problema de la clase de complejidad NP-completo), por lo tanto, la aportación de esta propuesta para SAT también es de suma importancia ya que esta se genera en un entorno específico y apoya directamente el campo de la teoría de grafos. En particular, presenta una forma de codificar el coloreo de ciclos usando sólo cláusulas monótonas.

Podemos concluir que se logró satisfacer los objetivos iniciales planteados en este trabajo de tesis, al poder desarrollar propuestas algorítmicas novedosas que traducen el problema del coloreo de grafos al problema de satisfactibilidad de fórmulas conjuntivas. Además de determinar la complejidad en tiempo de estas propuestas se analizaron comportamientos generales de las propuestas como fue el caso de la incompletés de nuestro primer algoritmo.

Este trabajo se engloba dentro de los esfuerzos del cuerpo académico de algoritmos combinatorios de la Facultad de Ciencias de la Computación, donde se desarrollan propuestas algorítmicas para la resolución de problemas combinatorios, con énfasis particular en problemas de la clase de complejidad NP.

5.2 Trabajo a Futuro

Es necesario trabajar más a fondo en el análisis de la segunda propuesta algorítmica de coloreo de grafos presentada en este documento, para caracterizarlo y de esta forma conocer a cabalidad sus limitaciones, lo cual será de gran apoyo para poder avanzar en este campo específico de la teoría de grafos, ya que de esta forma se generarán nuevas propuestas para comprender de mejor manera ciertos problema de la clase de complejidad NP-completo.

Este trabajo permitió ahondar en el problema SAT y genera a su vez una nueva propuesta de solución a este problema, misma que puede ser analizada más a fondo para determinar con exactitud el orden de complejidad de los algoritmos que lo solucionen. Tomando en cuenta que el problema SAT como tal es un problema NP-Completo, sobra decir que aún queda trabajo de análisis en esta línea de investigación.

Una línea interesante de exploración es la utilización de la codificación de los ciclos de un grafo a través de las fórmulas Booleanas propuestas, para plantear la solución de otros problemas típicos en teoría de grafos.

VI) Bibliografía Y Referencias

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

[2] Bodlaender H.L., Kratsch D., An exact algorithm for graph colouring with polynomial memory, Technical Report UU-CS-2006-015 (2006), *www.cs.uu.nl*.

[3] Beigel R., Eppstein, D., 3-colouring in time *O*(1.3289*n*), *Journal of Algorithms* 54 (2), (2005), pp. 168–204.

[4] Byskov J.M., Exact Algorithms for graph colouring and exact satisifiability, Phd thesis, University of Aarbus, Denmark, 2005.

[5] Wigderson A. Improving the performance guarantee of approximate graph coloring, *Jour. of the ACM*,30 (4):729-735, 1983.

[6] Blum A., Karger, D., An *O*(*n*3*/*14)-colouring algorithm for 3-colourable graphs, *Inf. Proc. Lett. 61(1)*, (1997), pp.49-53. http://www.cs.cmu.edu/_avrim/Papers/colour new.ps.gz.

[7] Arora S., Chlamtac E., Charikar M., New Approximation Guarantee for Chromatic Number, *Proceedings STOC 2006*, May 2006.

[8] 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.

[9] Grötzsch H., Ein Dreifarbensatz fur dreikreisfreie Netze auf der Kugel, Wiss. Z. Martin-Luther-Univ. Halle-Wittenberg Math.-Natur. Reihe 8 (1959), pp. 109120.

[10] Borodin O.V., Glebov A.N., Raspaud A., Salavatipour M.R., Planar graphs without cycles of length from 4 to 7 are 3-colorable, *Journal of Combinatorial Theory, Series B 93*, (2005) pp. 303 311.

[11] Gimbel J., Thomassen C., Coloring graphs with fixed genus and girth, *Trans. Amer. Math. Soc.* 349 (1997), pp. 45554564.

[12] Dvorak Z., Král D., Thomas R., Three-coloring triangle-free graphs on surfaces, Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), NY (2009), pp. 120129.

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

[14] Stacho J., 3-Colouring AT-free graphs in polynomial time, Algorithms and Computation (ISAAC 2010), *Lecture Notes in Computer Science 6507*, (2010) pp. 144– 155.

[15] Grötschel M., Lovasz L., Schrijver A., The ellipsoid method and its consequences in combinatorial optimization, *Combinatorica 1*, (1981), pp.169-197.

[16] Garey M., Johnson D., Computers and Intractability a Guide to the Theory of NP-Completeness, W.H. Freeman and Co., 1979.

[17] J-Carlson, A.Jaffe, A.Wiles, The millenium Priza Problems, Edit. Clay Mathematical Institute, Cambridge Massachusetts, 2006.

[18] Gary Chartrand, Ping Zhang, Introduction to Graph Theory, Mc Graw-Hill Int. Edition, 2005.

[19] Appel, Kenneth & Haken, Wolfgang & Koch, John, *Every Planar map is Four Colorable*, Illinois: Journal of Mathematics: vol.21: pp.439-567, December 1977.

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

[21] Galliere Jean H., Logic for Computer Science: Foundation of Automatic Theorem Proving, Wiley 1987.

[22] S. Cook. The Complexity of Theorem Proving Procedures. In Proceedings of the Third Simposium of the ACM on the Theory of Computing, pages 151-158, ACM Press, New York,1971.

[23] Guillermo De Ita, Javier A. Castillo, Recognizing 3-colorings cycle-patterns on graphs, Jour. Pattern Recognition Letters, Octubre 2012, http://dx.doi.org/10.1016/j.patrec.2012.10.001.