Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Computación

Doctorado en Ingeniería del Lenguaje y del Conocimiento

TESIS DOCTORAL

Modelado del 3-coloreo de grafos planares usando Satisfactibilidad Incremental

Tesis para obtener el grado de Doctor en Ingeniería del Lenguaje y del Conocimiento

---

Presenta:
Cristina López Ramírez

Asesor:
Dr. Guillermo De Ita Luna

Enero, 2021

Índice general

Índice de figuras

Tabla de acrónimos

IA: Inteligencia Artificial.

PD: Problema de decisión.

mt: Máquina de Turing.

mtd: Máquina de Turing determinista.

mtnd: Máquina de Turing no determinista.

N(x): La vecindad de x.

FNC: Forma Normal Conjuntiva.

2-FNC: 2 Forma Normal Conjuntiva.

3-FNC: 3 Forma Normal Conjuntiva.

SAT: Satisfactibilidad.

ISAT: Satisfactibilidad Incremental.

2-ISAT: Satisfactibilidad Incremental a partir de una 2 Forma Normal Conjuntiva.

BPF: Búsqueda primero en profundidad.

TTG: Grafo dos terminal.

TTSPG: Grafo serial-paralelo de dos terminales.

UP: Propagación unitaria.

MIN: Mínimo.

Ext(*G*): Vértices externos del grafo G.

Int(*G*): Vértices internos del grafo G.

C_k : Ciclo simple de longitud k.

$P_{k,t}$: Cadena poligonal con t polígonos de tamaño k.

G : Grafo.

G_o : Grafo outerplanar.

G_p : Árbol poligonal.

G_f : Grafo de caras.

V: Vértices.

4CT: Teorema de los 4 colores.

Parte I

Introducción y Fundamentos Teóricos

Capítulo 1

Introducción

Uno de los problemas fundamentales en el razonamiento automático es el problema de satisfactibilidad proposicional (SAT), que determina si una fórmula proposicional F es (o no) satisfactible. Esta es una tarea relevante con repercusiones en otros problemas estudiados en la Inteligencia Artificial (IA) (Fermé (2007), Chávez & Pozos (2014)) la estimación del grado de creencia, para revisar o actualizar las creencias, en la explicación abductiva, en el diagnóstico lógico y para otros procedimientos utilizados en el área de la Inteligencia Artificial.
Sea F una Forma Normal Conjuntiva (FNC), esto es, una conjunción de disyunciones de literales. Si F es una 2-FNC (2-SAT), puede ser resuelto mediante procesos deterministas en un tiempo polinomial, siendo de este modo un problema tratable. En tanto que se sabe que SAT(F) es NP-completo o intratable (Garey & David (1978)) a partir de tener cláusulas en F con al menos 3 literales. Los problemas NP-completos son aquellos que se resuelven en tiempo polinomial mediante procesos no deterministas, mientras que un proceso determinista requiere un tiempo de orden exponencial. Las variantes de 2-SAT en las áreas de optimización y conteo han sido esenciales para establecer las fronteras entre problemas tratables e intratables en cada una de esas áreas.

La complejidad computacional asociada con la solución de problemas NP-completos, ha motivado la búsqueda de métodos alternativos, que permitan la solución en tiempo polinomial de instancias especiales de problemas NP-completos Cook (1971).

A pesar de la dificultad teórica del problema SAT, los procedimientos actuales de decisión conocidos como solucionadores SAT, han sido sorprendentemente eficientes. Estos solucionadores se han utilizado en aplicaciones industriales Alexander et al. (2016). En estas aplicaciones, rara vez se limita a resolver sólo un problema de satisfactibilidad, en

su lugar, estas aplicaciones normalmente requieren revisar la satisfactibilidad de una secuencia de problemas relacionados. Los solucionadores modernos para SAT Auderman & Laurent (2003) manejan una secuencia de problemas de satisfactibilidad como una instancia del problema de satisfactibilidad incremental (ISAT).
El problema de satisfactibilidad incremental (ISAT) Mouhoub & Sadaoui (2007) consiste en comprobar si la satisfactibilidad se mantiene cuando se añaden nuevas cláusulas a una fórmula inicial que es satisfactible, es decir se considera una generalización del problema SAT al permitir cambios en el tiempo sobre la fórmula de entrada.
Hooker (1993) es uno los primeros investigadores en plantear el problema ISAT.
Una de las aplicaciones de ISAT es en el problema de coloreo de grafos el cual consiste en colorear los vértices de un grafo con el menor número de colores posibles, de forma que 2 vértices adyacentes no pueden tener un mismo color. Si existe tal coloreo con k colores, se dice que el grafo es k-coloreable.

En este sentido el 2-coloreo se resuelve en tiempo polinomial. Se ha encontrado que el 3-coloreo puede resolverse en tiempo polinomial para algunas topologías de grafos, como es el caso de grafos AT-free y grafos perfectos.
El coloreo de vértices en grafos es un campo activo de investigación con muchos subproblemas interesantes.

Tanto los problemas ISAT como el 3-coloreo han mostrado ser problemas difíciles (problemas NP-completos). En este trabajo de tesis se ha desarrollado la propuesta de complejidad computacional del problema ISAT. Se inicia considerando fórmulas en 2-FNC, que son reconocidas porque para estas fórmulas, SAT se resuelve en tiempo polinomial. Posteriormente, se va incrementado la 2-FNC de prueba al adicionar cláusulas unarias, binarias o ternarias. El objetivo es determinar la complejidad en tiempo de esta versión de ISAT. También se han desarrollado propuestas algorítmicas basadas en la aplicación del problema ISAT para modelar instancias del coloreo de grafos. En particular, el trabajo se enfoca en encontrar instancias (o topologías) de grafos que permitan determinar $X(G)$ (el número óptimo para colorear G) en tiempos de cómputo acotados polinomialmente, al considerar como método de modelación del problema de coloración, al problema 2-ISAT.

1.1. Justificación

SAT es un problema de la clase de complejidad NP-Completo. Las aplicaciones de SAT rara vez se limitan a resolver sólo una fórmula de entrada, una aplicación normalmente resolverá una secuencia de fórmulas relacionadas. Los solucionadores del SAT moderno manejan estas secuencias de fórmulas como una instancia al problema de

satisfactibilidad incremental (ISAT). Por lo que es de interés en la comunidad el encontrar métodos para resolver ISAT.
ISAT es de interés a una gran variedad de aplicaciones que necesitan revisar la satisfactibilidad de fórmulas en un entorno evolutivo. Esto podría ser el caso de aplicaciones en la planificación y programación reactiva, en la optimización combinatoria dinámica, en la revisión de fallas en circuitos combinatorios João & Monteiro (2017), en la satisfacción de restricciones dinámicas Silvio et al. (2017) y en el aprendizaje de máquinas en un entorno dinámico Mouhoub & Sadaoui (2007).

Los algoritmos actuales aplicados en la solución de ISAT Florian & Marques-Silva (2008), Mohamed-El (2004) no precisan la complejidad en tiempo del mismo problema ISAT, aún más, se reconoce que este problema requiere de propuestas más adecuadas según su área de aplicación.
De igual modo se ha reconocido la dificultad computacional para resolver ISAT Hooker (1993), Mouhoub & Sadaoui (2007), sin embargo, no se conocen estudios formales que precisen la clase de complejidad para ISAT, así como para sus variantes, por ejemplo, cuando ISAT inicia con una 2-FNC (2-ISAT). La intención en este trabajo es precisar la complejidad computacional de 2-ISAT.

1.2. Problema de investigación

Se sabe que el problema 2-ISAT consta de 2 fases; en la primera fase se da como entrada una 2-FNC K para determinar SAT(K) en tiempo polinomial. En esta primera fase se construyen estructuras combinatorias que serán usadas en la segunda fase del problema. En la segunda fase del problema 2-ISAT, se adiciona una fórmula ϕ que esta en 3-FNC, y se cuestiona sobre SAT($K \wedge \phi$).

En esta investigación se propone una estrategia para utilizar estructuras que se formen en la fase 1 de 2-ISAT y que sean usadas durante la fase 2 del problema. Se propone revisar si pueden haber estructuras computacionales que creadas en la primer fase, resuelvan 2-ISAT en tiempo polinomial, o bien, el problema 2-ISAT es de complejidad NP. Se realizará el análisis de la complejidad en tiempo en ambas fases del problema para validar la complejidad en tiempo del problema 2-ISAT.

Por otro lado es fácil (en tiempo lineal sobre el tamaño del grafo) reconocer si un grafo de entrada es 2-coloreable, ya que implica el reconocimiento de que solo ocurran ciclos de longitud par en el grafo. Similarmente se sabe por el teorema de Grötzsch que cualquier grafo planar libre de triángulos es 3-coloreable Grötzsch (1959). Sin embargo,

el reconocimiento del 3-coloreo de un grafo planar es un clásico problema NP-Completo. Por ello es difícil reconocer entre si se deben usar 3 o 4 colores para un grafo planar cuando este contiene triángulos, debido a que no se conoce (al menos hasta ahora) una condición suficiente para reconocer la 3-colorabilidad de un grafo planar.

Para esto último, se propone reconocer las topologías de grafos planares cuyo modelado a través de SAT permita proponer nuevos algoritmos competitivos con el estado del arte. Se realizarán propuestas algorítmicas de coloreo identificando para que tipo de grafos planares, se resuelve de manera eficiente dichas propuestas. Asi mismo se validará al analizar la complejidad en tiempo de los algoritmos en base a la longitud del grafo.

1.3. Hipótesis

- Hipótesis 1: Cuando K es una 2-FNC y ϕ es una 3-FNC y no hay ningún tipo de restricciones sobre K y ϕ, entonces SAT($K \wedge \phi$) es un problema NP-completo, excepto los casos especiales en los que se resuelve eficientemente.
- Hipótesis 2: Es posible modelar el coloreo de grafos con topologías especiales utilizando Satisfactibilidad Incremental.
- Hipótesis 3: Al extender SAT con restricciones lógicas entre variables simbólicas se podría modelar el coloreo de grafos planares.

1.4. Objetivos de la investigación

Objetivo general:

Modelar el problema del coloreo de grafos planares en base a la revisón de Satisfactibilidad (problema SAT proposicional) de las restricciones que determina la topología del grafo planar, con la intención de analizar la complejidad computacional de la 2-satisfactibilidad incremental (2-ISAT).

Objetivos específicos:

- Determinar la complejidad computacional del problema 2-ISAT.
- Aplicar SAT para modelar el problema del coloreo sobre grafos planares.

- Determinar topologías de grafos planares donde su modelado vía Satisfactibilidad permita determinar su coloreo de forma eficiente.

1.5. Preguntas de investigación

- ¿Las estructuras creadas al resolver SAT(K), K una 2-FNC, facilitarán la revisión de SAT($K \wedge \phi$), ϕ una 3-FNC?.
- ¿Cuál es la complejidad computacional en tiempo del problema de Satisfactibilidad Incremental (2-ISAT)?.
- ¿Es posible que la especificación lógica de las restricciones para un 3-coloreo sobre grafos planares, permita el desarrollo de algoritmos de complejidad polinomial en tiempo para resolver el problema de coloreo de grafos?.
- ¿Para que topologías de grafos planares es posible proponer algoritmos eficientes basados en el problema SAT, para hallar el coloreo de sus vértices?

1.6. Organización de la tesis

La tesis está compuesta por los siguientes capítulos:

En el capítulo 2 se incluyen conceptos básicos de las áreas de investigación que involucra esta tesis: problema de satisfactibilidad sobre una fórmula proposicional F en Forma Normal Conjuntiva, así como clases de complejidad, reglas de simplificación sobre FNC y teoría de grafos.
En tanto que en el capítulo 3 se presenta la revisión del estado del arte relacionado con la teoría del problema de satisfactibilidad desde sus orígenes hasta la actualidad.
En el capítulo 4 se presenta la metodología de investigación en la que se describen cada uno de los pasos realizados en el proyecto. Se describen las principales etapas en el desarrollo del proyecto de tesis, incluyendo técnicas y métodos.
Los resultados obtenidos en relación a la complejidad computacional del problema 2-ISAT se incluyen en el capítulo 5. De igual modo, las propuestas algorítmicas para el problema del coloreo de grafos planares se describen en el capítulo 6. Finalmente las conclusiones de los resultados obtenidos y las lineas abiertas de estudio son expuestas en el capítulo 7.

Capítulo 2

Marco teórico

Se presenta en este capítulo el sustento teórico de esta investigación.

2.1. Preliminares

Una instancia de una Forma Normal Conjuntiva (FNC) se compone por un conjunto U={u_1,...,u_n} de variables Booleanas y una colección F={C_1,...,C_m} de cláusulas definidas sobre U. Asi, una cláusula es una disyunción de literales $C = l_1 \vee \ldots \vee l_k$ y es verdadera si alguna de sus literales lo es.

Una literal denotada a través de la constante V(verdadero) o es una expresión de la forma u o $\neg u$, donde u es elemento de U.
Forma Normal conjuntiva (FNC) es una conjunción de cláusulas. Una k-FNC, $k \in N$, es una FNC con exactamente k literales por cada cláusula, de tal forma que 2-FNC, $2 \in N$ es una FNC con exactamente 2 literales por cada cláusula y 3-FNC, $3 \in N$ es una FNC con exactamente 3 literales por cada cláusula.
Una FNC F es satisfactible si y solo si existe una asignación de valores de verdad para U que simultáneamente satisfaga a cada una de las cláusulas en F, por lo que una asignación es una función $t : U \longrightarrow \{F, V\}$.

2.2. Satisfactibilidad

Proposición 2.1. Toda fórmula proposicional es lógicamente equivalente a una FNC, y de hecho la FNC equivalente es algorítmicamente calculable Galliere (1988).

La pregunta de satisfactibilidad consiste en decidir si 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.

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:

$F(X_1, \ldots, X_7) = (\neg X_1 \vee \neg X_2) \wedge (X_1 \vee X_3 \vee X_4) \wedge (X_5 \vee \neg X_3) \wedge (\neg X_5 \vee \neg X_6) \wedge (X_1 \vee X_6 \vee X_7) \wedge (\neg X_5 \vee \neg X_7) \wedge (\neg X_3 \vee \neg X_7)$.

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 2.1), se determine si acaso existe una asignación de U que haga que F tome el valor verdadero.
Se denota con SAT(F) al conjunto de modelos que satisfacen a la fórmula F. En caso de que para una fórmula F, SAT(F) = $\emptyset$ se dice entonces que F es una contradicción o que es insatisfactible como se muestra en la Figura 2.1.

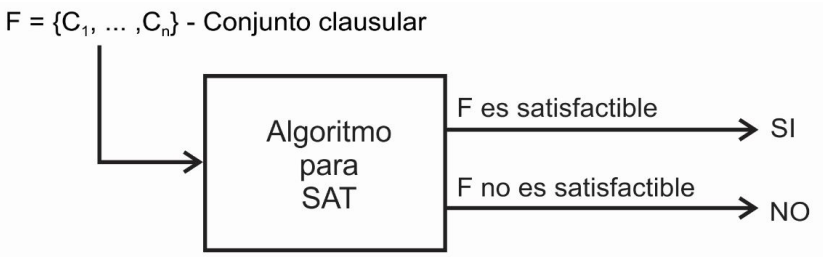


Figura 2.1: Planteamiento del problema SAT

Un ejemplo sencillo del proceso de revisión de la satisfactibilidad incremental, es el siguiente:
Sea K una 2-FNC y sea ϕ nueva información:
K=$C_1 \wedge C_2$. Donde $C_1 = (\neg a \vee b)$ y $C_2 = (\neg a \vee c)$. Tales cláusulas pueden escribirse, como: $a \longrightarrow b$, $a \longrightarrow c$.

Puede verse que K es satisfactible, las únicas asignaciones que falsifican a K, son:

a	b	c	falsifican a
1	0	*	$(\neg a \vee b)$
1	*	0	$(\neg a \vee c)$

Donde * indica que la variable puede tomar cualquier valor de 0 o 1. Consideramos que se va a agregar la siguiente información (ϕ): $D1 = (\neg b \vee \neg c)$. $D2 = (a)$. $D3 = (\neg a \vee b \vee \neg c)$.
Consideremos que las variables se ordenan como: a, b, c, para construir las asignaciones satisfactibles.
Nótese que: ($K \wedge D1 \wedge D3$) es satisfactible, por el modelo: 0,1,0
($K \wedge D2 \wedge D3$) es satisfactible, por el modelo: 1,1,1
($D1 \wedge D2 \wedge D3$) es satisfactible, por el modelo: 1,1,0
Pero ($K \wedge D1 \wedge D2 \wedge D3$) es insatisfactible.
En el estado del arte se cuestiona sobre el problema de satisfactibilidad incremental (ISAT)
¿($K \wedge \phi$) mantiene la satisfactibilidad (consistencia)?
¿Si se van agregando las nuevas cláusulas de una en una, en que orden es mejor agregarlas, para que se mantenga la satisfactibilidad de ($K \wedge \phi$) el mayor tiempo posible?.
¿Cuál es el máximo subconjunto de $S \subseteq \phi$ que permite mantener a ($K \wedge S$) satisfactible.?

2.2.1. Satisfactibilidad Incremental

Sea K una FNC, por ejemplo: $K = \wedge_{i=1}^{m} C_i$ donde cada C_i $i = 1, \ldots, m$ es una disyunción de literales.
Introducimos el problema principal a estudiar, denotado como satisfactibilidad incremental.
Instancia: Sea K una 2-FNC, ϕ una 3-FNC, donde las variables de ϕ son variables que aparecen en K, $v(\phi) \subseteq v(K)$.
Problema: Determinar SAT($K \wedge \phi$).
Presentamos en esta sección, reglas de simplificación sobre una FNC, manteniendo sólo las subfórmulas que determinan la satisfactibilidad de la fórmula original. Por ejemplo, para una FNC es común el eliminar todas las cláusulas redundantes, así como cláusulas tautológicas, cláusulas repetidas y cláusulas con literales puras.

2.2.2. Regla de cláusula subsumida:

Es una regla usada para simplificar la fórmula lógica de entrada. Son reglas a ser utilizadas para realizar el proceso central llamado propagación unitaria, de nuestros algoritmos. Dadas dos cláusulas C_i,C_j de una FNC F, si $Lit(C_i) \subseteq Lit(C_j)$, entonces C_j

es subsumida por C_i, y entonces C_j pude ser eliminada de F, porque toda asignación que satisface a C_j también satisface a C_i, esto es $Sat(C_j) \subseteq Sat(C_i)$. Por lo que es suficiente sólo mantener a C_i en la FNC.
Aún más, esta regla de cláusula subsumida puede combinarse con resolución para simplificar la FNC ϕ, como se muestra en el lema siguiente:

Lema 2.4 Sea $(x, y) \in K$ y una cláusula $(\neg x, y, z) \in \phi$ entonces se resuelve (y, z) que es una cláusula binaria que puede ser adicionada a K y su padre $(\neg x, y, z)$ puede ser eliminado de ϕ.

Prueba.
Se tiene que $((x \vee y) \wedge (\neg x \vee y \vee z)) \equiv ((x \wedge y) \vee (x \wedge z) \vee y)$ por absorción, y esto equivale a $((x \wedge (y \vee z)) \vee y \equiv (x \vee y) \wedge (y \vee z \vee y) \equiv (x \vee y) \wedge (z \vee y))$. Por lo que $(\neg x \vee y \vee z)$ puede borrarse de ϕ, porque es subsumida por $(y \vee z) \in K$ y por lo tanto, el conjunto de modelos de ($K \wedge \phi$) es preservado sin cambios.

2.2.3. Regla de la literal Pura:

Es una regla usada para simplificar la fórmula lógica de entrada. Son reglas a ser utilizadas para realizar el proceso central llamado propagación unitaria, de nuestros algoritmos. Sea F una FNC, $l \in Lit(F)$ si l aparece en F pero $\neg l$ no aparece en F, se dice que entonces l es una literal pura en F.
Si una cláusula contiene una literal pura, tal cláusula puede ser eliminada de F, manteniéndose el valor lógico de F. Porque si la literal l es verdadera, la cláusula conteniendo l es también verdadera, por tanto esto puede ser eliminado de F.
Sin embargo estas reglas tienen que ser aplicadas con cuidado en el caso de 2-ISAT, ya que en el proceso de adicionar nuevas clausulas ϕ a K, las literales que inicialmente eran puras, podrían dejar de serlo en $K \cup \phi$.
Por lo tanto, para eliminar cláusulas con literales puras, éstas tendrán un alcance local, trabajando en cada instancia ($K \wedge \phi$). Pero, si un nuevo conjunto de cláusulas ϕ_{i+1} es considerada, entonces todas las literales puras que fueron eliminadas de $K \wedge \phi$ deben ser retornadas a ϕ_{i+1}.

2.2.4. Regla de resolución unitaria:

Sea K una forma normal conjuntiva (FNC) y l una literal, la reducción de K por l, denotado por $K[l]$, es la fórmula generada al eliminar las cláusulas que contienen l de K (subsumida), y eliminando $\neg l$ de las cláusulas restantes (resolución unitaria). La reducción $K[s]$ donde s es un conjunto de literales $s = \{l_1, l_2, \ldots, l_k\}$ se define aplicando sucesivamente $K[l_i]$, l_i, $i = 1, \ldots, k$. La reducción de K por l_1 da la fórmula $K[l_1]$, siguiendo de una reducción de $K[l_1]$ por l_2, dando como resultado $K[l_1, l_2]$ y así sucesivamente. El proceso continúa hasta que $K[s] = K[l_1, \ldots, l_k]$ es alcanzado. En

caso de que $s = \emptyset$ entonces $K[s] = K$.
Sea K una FNC y s una asignación parcial de K. Si se obtiene un par de cláusulas unitarias contradictorias mientras $K[s]$ es calculada, entonces K es falsificada por la asignación s. Además, durante el cálculo de $K[s]$, se pueden generar nuevas cláusulas unitarias. Por lo tanto, la asignación parcial s se extiende agregando las cláusulas unitarias que se van formando, es decir, $s = s \cup \{u\}$ donde $\{u\}$ es una cláusula unitaria. $K[s]$ puede reducirse nuevamente usando las nuevas cláusulas unitarias que se van formando. A este proceso iterativo se le denomina Propagación_Unitaria(K, s). Por simplicidad, se abrevia Propagación_Unitaria(K, s) como UP(K, s).
La aplicación de UP(K, s) genera una nueva asignación s' que extiende a s, y una nueva subformula K' formada por las cláusulas de K que no son satisfactibles por s'. Denotamos a la fórmula K' como $K' = UP(K, s)$. Notar que si s falsifica a K entonces s' puede tener literales complementarias y K' contiene la cláusula nula. Y cuando s satisfice a K, entonces K' es el conjunto vacío.
Al revisar la satisfactibilidad de la nueva fórmula $K' = UP(K, S)$, se revisa si K' contiene literales puras o si en K hay cláusulas subsumidas que puedan eliminarse de K con el fin de simplificar el proceso $UP(K, S)$. El proceso $UP(K, S)$ es parte central de nuestras propuestas para el coloreo de grafos.

2.3. Teoría de la complejidad

En la teoría de la complejidad se manejan conceptos como: problemas de decisión, máquinas de Turing, reducción polinomial y clases de complejidad, todos estos son relevantes para el análisis de la complejidad de los algoritmos, mismos que se explican a continuación.

2.3.1. Planteamiento computacional de problemas de decisión

Por conveniencia, la teoría de la complejidad está diseñada especialmente para aplicarse a problemas de decisión. Un problema de decisión (PD) se plantea como una pregunta general la cual acepta como respuesta sólo una de dos posibilidades, 'SI' o '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:

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

Para el problema de coloreo de grafos, el planteamiento quedaría de la siguiente forma: **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 $K \in Z^+$, con $K \leq |V|$.

En el estado del arte se cuestiona sobre la k colorabilidad ¿Será G K-coloreable ?, es decir, existirá una función $f : V \rightarrow \{1, 2, ..., K\}$ tal que $f(u) \neq f(v)$ siempre que $\{u, v\} \in V$.
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, 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 Garey & David (1978).

2.3.2. Máquinas de Turing y clases de complejidad

Existen varios modelos formales de computación, uno de los más comúnmente usado y que es muy simple de enunciar, es la llamada máquina de Turing (abreviado con mT). Se definen diferentes mT's de acuerdo a la forma en que trabajan. Con el fin de clasificar el esfuerzo computacional que se requiere al resolver los problemas de decisión, originalmente se usó a las máquinas de Turing como el modelo formal de computación Hartmanis & Stearns (1965). El concepto clave fue definir una clase de complejidad en términos de los lenguajes que reconoce una máquina de Turing con recursos acotados (consideraremos aquí sólo los recursos de tiempo y espacio). Las clases de complejidad permiten clasificar los lenguajes (problemas) según la complejidad intrínseca computacional requerida para reconocerlos (resolverlos). De particular interés, son las clases de complejidad, definidas por recursos acotados que crecen logarítmicamente (salvo se indique lo contrario, estaremos hablando de logaritmos base 2) o polinomios sobre la variable n, donde n es la longitud de las instancias de entrada.

Máquinas de Turing determinista

Un modelo básico de una mT, es la llamada determinista (abreviado con mTd), que consiste de un control finito, una cabeza de lectura-escritura, y una cinta dividida en un número infinito de celdas, estas celdas son etiquetadas usando números enteros, tal y como se muestra en la Figura 2.2. De manera formal una mTd M se representa por: $M =< Q, \sum, t, q_0, F >$, donde:

- Q: Es un conjunto finito de estados que indica los diferentes estados en los que puede estar el control finito, este conjunto de estados incluye dos estados distinguidos de parada: q_y y q_n y el estado de inicio q_0.
- q_0: El estado distinguido en el que arranca la mT.

$\sum$: Alfabeto de símbolos, son los símbolos usados en la codificación de la cadena de entrada así como los símbolos que pueden colocarse en la cinta de la mT. Un símbolo distinguido es el espacio en blanco denotado por b.

F: Subconjunto de $Q.F = \{q_y, q_n\}$ que denota a los estados finales o de paro de la mT.

t: Es la función de transición t:$(Q - F)$ X $\sum \rightarrow Q \times \sum \times \{-1, 0, 1\}$. La función de transición codifica al programa que va a ser ejecutado por la máquina de Turing.

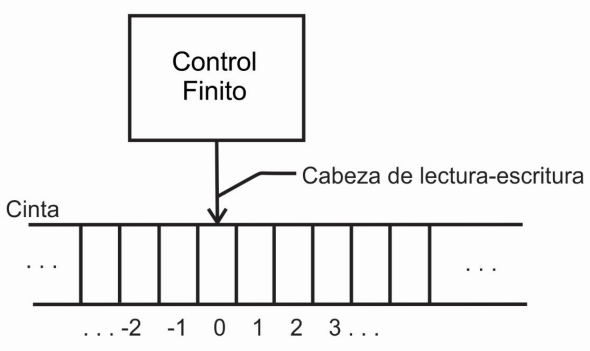


Figura 2.2: Representación esquemática de una máquina de Turing determinista.

Clases de complejidad definidas por mtd

DLOG = $\{L \subseteq \sum^* | \forall x \in L, x$ es aceptada en espacio logarítmico $\}$.
P = $\{L \subseteq \sum^* | \forall x \in L, x$ es aceptada en tiempo polinomial $\}$.
PSPACE = $\{L \subseteq \sum^* | \forall x \in L, x$ es aceptada en espacio polinomial $\}$.

O en términos de los problemas de decisión, podemos definir a las clases de complejidad, de la manera siguiente:

DLOG = $\{PD | \exists M$ mTd que resuelve PD usando espacio logarítmico $\}$.
P = $\{PD | \exists M$ mTd que resuelve PD en tiempo polinomial $\}$.
PSPACE = $\{PD | \exists M$ mTd que resuelve PD usando espacio polinomial $\}$.

Máquinas de Turing no determinista

Usaremos el modelo de máquina de Turing no determinista (abreviado con mTnd) definido en Garey & David (1978), tal modelo tiene la misma estructura que una mTd, excepto que se le adiciona un módulo de adivinanza, el cual tiene su propia cabeza de sólo escritura, tal y como se muestra en la Figura 2.3. El módulo de adivinanza se usa sólo con el propósito de que escriba la cadena que adivina y sólo bajo tal propósito es usado.

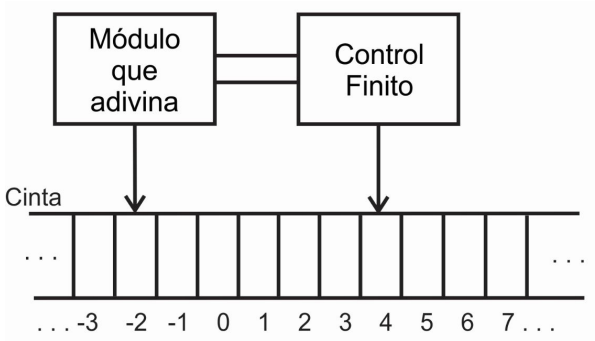


Figura 2.3: Representación esquemática de una máquina de Turing no determinista.

Clase P: Problemas que se resuelven de forma determinista por algoritmos de complejidad 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.
NP-Completo: son problemas que representan a la clase NP, en el sentido de que cualquier otro problema NP puede reducirse en tiempo polinomial a estos.
Clases de complejidad definidas por mtnd

NLOG	=	$\{PD\|\exists M$ mTnd que comprueba soluciones de PD usando espacio logarítmico }.
NP	=	$\{PD\|\exists M$ mTnd que comprueba soluciones de PD en tiempo polinomial }.
NPSPACE	=	$\{PD\|\exists M$ mTnd que comprueba soluciones de PD usando espacio polinomial }.

2.4. Teoría de grafos

Sea $G = (V, E)$ un grafo simple no dirigido (es decir, finito, sin bucles y sin múltiples aristas). $V(oV(G))$ denota el conjunto de vértices y $E(oE(G))$ el conjunto de aristas. Dos vértices v y w son llamados adyacentes, si $\{v, w\} \in E$. Se dice que la arista {v, w} es incidental a los vértices u y v. Dos aristas son adyacentes si tienen un vértice en común.
La vecindad de $x \in V$ es $N(x) = \{y \in V : \{x, y\} \in E\}$, mientras que su vecindad cerrada es $N(x) \cup \{x\}$, que se denota por $N[x]$. La cardinalidad de un conjunto A se denota por |A|. El grado de un vértice $x \in V$, esta dado por $\delta(x)$, es |$N(x)$|. El grado del grafo G es $\Delta(G)$ = max$\{\delta(x) : x \in V\}$ De Ita et al. (2011).
Un camino desde un vértice v a w es una secuencia de aristas: v_0v_1, v_1v_2,. . . ,$v_{n-1}v_n$ tal que $v = v_0$, $v_n = w$, y es tal que v_k es adyacente a v_{k+1} y la longitud del camino es n. Un camino simple es un camino tal que v_0, v_1, . . . , v_{n-1}, v_n son todos diferentes. Un ciclo es un camino no vacío en el que el primer y el último vértice son idénticos. Un ciclo simple es un ciclo en el que no se repiten vértices, excepto el primero y el último vértice. Un

k-ciclo es un ciclo de longitud k (hay k aristas). Un ciclo de longitud impar se denomina ciclo impar, mientras que un ciclo de longitud par se denomina ciclo par. Un grafo sin ciclos se llama acíclico. Dado un subconjunto de vértices $S \subseteq V$, el subgrafo de G donde S es el conjunto de vértices y el conjunto de aristas es $\{\{u, v\} \in E : u, v \in S\}$, se llama el subgrafo de G inducido por S y se denota por $G|S$. $G - S$ denota el grafo $G|(V - S)$. El subgrafo inducido por $N(v)$ se denota como $H(v) = G|N(v)$, el cual contiene todos los nodos de $N(v)$ y todas las aristas que los conectan.
Un subconjunto $S \subseteq V$ es llamado independiente en G, si para cada u, $v \in S$ implica que $\{u, v\} \notin E$. El tamaño de un conjunto independiente es el número de vértices que este contiene. Un conjunto independiente es maximal si no es subconjunto propio de otro conjunto independiente, y es máximo en G si no hay otro conjunto independiente en G con una cardinalidad mayor a $|S|$.

2.4.1. Búsqueda primero en profundidad

Una búsqueda que es sencilla, eficiente(de orden lineal) y que será útil para reconocer el tipo de topología de los grafos a considerar, es la búsqueda primero en profundidad(bpf) Aho et al. (1988). Dado un grafo conectado no dirigido $G = (V, E)$, al aplicar la bpf sobre G se produce un grafo de árbol T_G, donde $V(T_G) = V(G)$. Las aristas en T_G se llaman aristas de árbol, mientras que a las aristas en $E(G) \setminus E(T_G)$ se les llama aristas *frond* (aristas de retroceso).
Sea $e \in E(G) \setminus E(T_G)$ una arista frond, la unión del camino en T_G entre los vértices finales de e con la misma arista e conforma un ciclo simple, dicho ciclo se denomina ciclo básico (o fundamental) de G con respecto a T_G. Cada arista frond indica el camino máximo contenido en el ciclo básico del que forma parte. Los nodos finales de un ciclo son los nodos que forman parte de la arista frond del ciclo.
Sea $C = \{C_1, C_2, \ldots, C_k\}$ el conjunto de ciclos fundamentales formados durante la bpf sobre G. Dado un par de ciclos básicos C_i y C_j de C, si C_i y C_j comparten alguna arista, a los ciclos se les llama *intersectados*; de lo contrario, se llaman ciclos *independientes*. Un conjunto de ciclos es independiente si no se intersectan cualesquiera dos ciclos del conjunto. En particular, si dos ciclos comparten solo una arista, se les llama ciclos *adyacentes*.
Supongamos una entrada de un grafo conectado $G = (V, E)$, con $n = |V|$ y $m = |E|$, así como un orden sobre los vértices que permite aplicar la bpf de manera determinista. Por ejemplo, comenzando la búsqueda con un nodo $v \in V$ de grado mínimo y visitar el nodo de grado más bajo cada vez que hay múltiples posibles nodos para visitar y con una etiqueta menor para el caso de varios vértices con mismo grado minimal. Denotemos esta búsqueda en profundidad como $bpf(G)$. Sea $G' = bpf(G)$ el nuevo grafo generado por la búsqueda primero en profundidad. La búsqueda $bpf(G)$ permite detectar si G tiene ciclos o no, y si estos ciclos son de longitud par o impar, todo esto en un tiempo de computación de orden $O(m \bullet n)$.
Dado $G' = bpf(G)$, Sea T_G el árbol de expansión de G. Note que $V(T_G) = V(G)$. Y

sea $C = \{C_1, C_2, \ldots, C_k\}$ el conjunto de ciclos fundamentales encontrados durante la $bpf(G)$. Si G es un grafo acíclico, entonces $T_G = G'$ y $C = \emptyset$.
Si G' es acíclico o contiene solo ciclos fundamentales de longitud par, entonces G es bipartito y por tanto es 2-coloreable, ya que los vértices en $G' = bpf(G)$ pueden ser coloreados por niveles, es decir, todos los vértices en el mismo nivel tienen el mismo color y los nodos de dos niveles consecutivos estarán coloreados con los dos colores de forma alterna.

Análisis de la búsqueda en profundidad
Todas las llamadas a bpf en la búsqueda en profundidad de un grafo con a arcos y n<=a vértices lleva un tiempo $O(a)$ (ver algoritmo 1). Por ejemplo, obsérvese que en ningún vértice se llama a bpf más de una vez, porque tan pronto como se llama a $bpf(v)$ se hace $marca(v)$ igual a visitado en la línea (1), y nunca se llama a bpf en un vértice que antes tenía su $marca$ igual a $visitado$. Así, el tiempo total consumido en las líneas (2) y (3) recorriendo las listas de adyacencias es proporcional a la suma de las longitudes de dichas listas, esto es, $O(a)$. De esta forma, suponiendo que $n <= a$, el tiempo total consumido por la búsqueda en profundidad de un grafo completo es $O(a)$, lo cual es, hasta un factor constante, el tiempo necesario para recorrer cada arco.

Algoritmo 1 búsqueda primero en profundidad

```
procedure bpf(v: vértice);
   var
     w:vértice;
   begin
(1)      marca[w]:= visitador
(2)     for cada vértice w en L[v]
(3)       if marca[w] = no_visitado then
(4)           bpf(w)
   end; bpf
```

2.4.2. Grafos Planares

Un dibujo Γ de un grafo planar G asigna cada vértice v a un punto distinto $\Gamma(v)$ del plano y cada arista $\{u, v\}$ a una línea simple o curva de Jordan $\Gamma(u, v)$ con los puntos finales $\Gamma(u)$ y $\Gamma(v)$. Un dibujo es planar si no hay cruce de aristas, excepto posiblemente, en puntos finales comunes. Un grafo es planar si admite un dibujo planar. Hablaremos del embebimiento de un grafo G, cuando G es planar y se ha dibujado en el plano sin cruce de sus aristas.
Dado un dibujo plano, el orden circular (en el sentido de las agujas del reloj) de los bordes que inciden en cada vértice es fijo. Dos dibujos planos son equivalentes si determinan los mismos ordenamientos circulares de los bordes que inciden en cada vértice (a veces llamado esquema de rotación). Un planar embebido es una clase equivalente de dibujos

planares y se describe mediante el orden circular en el sentido de las agujas del reloj de las aristas que inciden en cada vértice. Un grafo junto con uno de estos planares embebidos se denomina grafo plano.
Un dibujo planar divide el plano en regiones conectadas llamadas caras. La cara sin límites usualmente es llamada cara externa o cara del exterior. Si todos los vértices son incidentales a la cara exterior, el grafo es llamado $outerplanar$ Oubiña & Zucchello (1984).
Fijando un embebimiento de un grafo planar G, G tiene un conjunto de regiones internas cerradas no intersectadas, llamadas caras internas de G: $R = \{f_1, \ldots, fk\}$. Cada cara f_i está representada por el conjunto de aristas que delimita su zona interior. La cara exterior es la única cara ilimitada de G. Toda arista $\{u, v\}$ en G que no es el borde de alguna cara de G, se llaman aristas acíclicas.
Así se define un grafo planar donde se toman en cuenta las caras internas del grafo.
Definición: Sea G un grafo planar, el grafo dual-interno (o solo grafo dual), denotado por G^d, se forma al representar cada cara interna de G por un nodo en G^d. Caras internas adyacentes en G definen una arista en G^d entre los nodos que representan tales caras. Dado un grafo dual G^d, llamaremos a G el grafo base de G^d, como se muestra en la Figura 2.4.

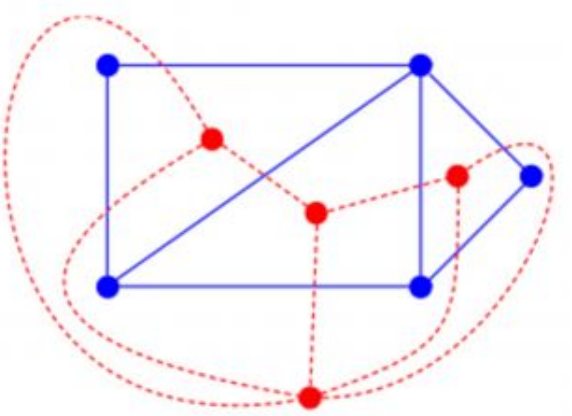


Figura 2.4: Las lineas continuas definen al grafo base, mientras las aristas en puntos definen al grafo dual.

Los grafos planares juegan un papel importante tanto en las áreas de la teoría de grafos como en la de dibujo de grafos. De hecho, los grafos planares tienen varias propiedades interesantes Patrignani (2004), son grafos dispersos, cuatro coloreables, permiten que se realicen varias operaciones de manera más eficiente que para los grafos generales y su estructura interna puede describirse de manera breve y elegante.
Una representación planar de un grafo G es un conjunto de puntos en el plano que se corresponden con los vértices de G unidos por curvas que se corresponden con las aristas de G, sin que estas se crucen entre sí. Un grafo es planar si admite una representación planar que se ha dibujado sin el cruce de aristas Bekos et al. (2017)

como se muestra en la Figura 2.5.

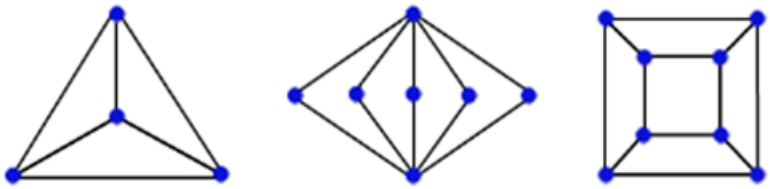


Figura 2.5: Grafos planares.

Un grafo no plano es un grafo que puede tener cualquier número de aristas cruzadas entre sí.
Por ejemplo en la Figura 2.6 estos dos grafos no son planos.

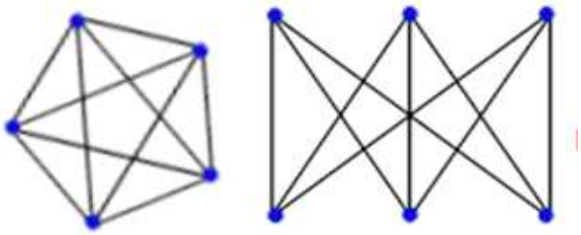


Figura 2.6: Grafos no planares.

2.4.3. Cadena Poligonal

Una cadena poligonal $P_{k,t}$ es un grafo obtenido al identificar un número finito de t polígonos de tamaño al menos k, de modo que cada polígono, excepto el primero y el último, sean exactamente adyacentes a dos polígonos. Cuando cada polígono en $P_{k,t}$ tiene el mismo número k de vértices, entonces a $P_{k,t}$ se le llama una cadena poligonal con t k-gons, y se denotará por H_t. Sea C_k un ciclo simple de un grafo de longitud k. A C_k también se le llama un poligono de tamaño k De Ita et al. (2018).
Sea $H_t = h_1, h_2 \dots h_t$ una cadena poligonal con t polígonos, donde cada h_i y h_{i+1} comparten exactamente una arista común e_i, $i = 1, 2, \dots, t-1$. Una cadena poligonal con al menos dos polígonos tienen dos polígonos-finales: h_1 y h_t. Mientas tanto $h_2, \dots, h_{t1}$ son los polígonos internos de la cadena. En una cadena poligonal, cada vértice tiene grado 2 o 3. Los vértices de grado 3 son exactamente los puntos finales de las aristas comunes entre dos polígonos consecutivos como se ve en la Figura 2.7.

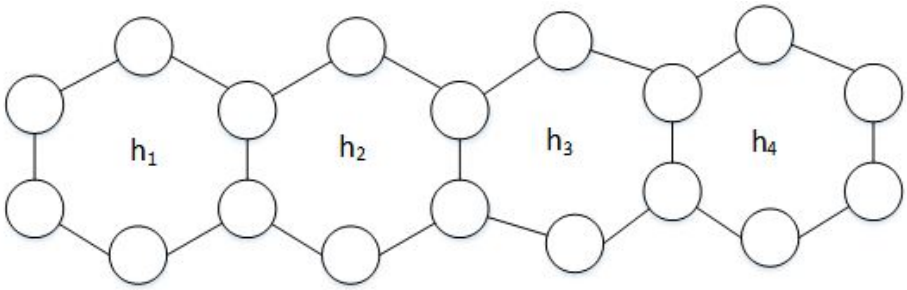


Figura 2.7: Cadena poligonal formada por 6-gons.

2.4.4. Grafos Outerplanar

En teoría de grafos, un grafo outerplanar es un grafo que tiene un dibujo plano para el cual todos los vértices pertenecen a la cara externa del dibujo como se muestra en la Figura 2.8. Los grafos outerplanar son caracterizados por prohibirse como minors a K_4 y a $K_{2,3}$ Chartrand & Harary (1967), como se muestra en la Figura 2.9. Un grafo H es un minor de otro grafo G, si H se puede obtener de G al eliminar aristas y vértices, y por contraer aristas. Una arista $\{x, y\}$ se contrae al eliminar x, y, $\{x, y\}$ del grafo, y formar un nuevo vértice z con vecindad $(N(x) \cup N(y))$. Un grafo outerplanar tiene un ciclo hamiltoniano si y solo si el grafo es biconectado, en cuyo caso, la cara externa forma un único ciclo hamiltoniano. Cada grafo outerplanar es 3-colorable Andrzej & Maciej (1986). Los grafos outerplanar son un subconjunto de los grafos planares.

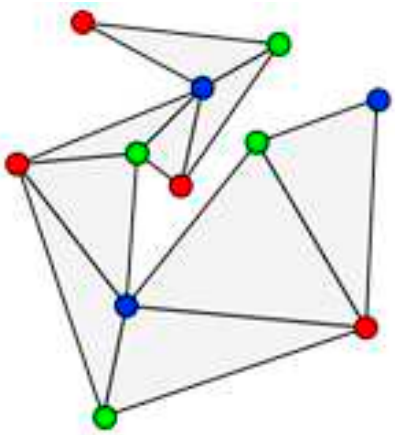


Figura 2.8: Grafo Outerplanar.

2.4.5. Grafos Serial Paralelo

Un grafo dos - terminal (TTG) es un grafo con 2 vértices distinguidos: s, t llamados Fuente(s) y objeto (t). Un grafo es llamado serial-paralelo Bubeck & Kleine (2009) si

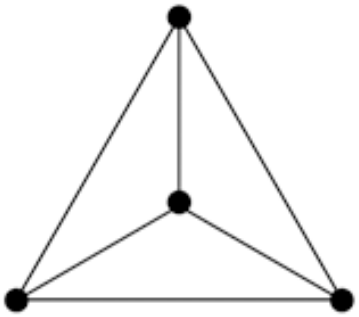


Figura 2.9: Grafo K_4 no es outerplanar.

es un TTG y cuando dos de sus vértices se consideran Fuente y objeto(sink), como se muestra en la Figura 2.10.
Un grafo serial - paralelo, de dos terminales TTG es un grafo que puede ser construido por una secuencia de composición: Serial y/o paralelo iniciando de un conjunto de copias del grafo K_2: identificando sus dos terminales.
La composición paralela $P_c = P_c(x, y)$ de dos TTG's x y y es un TTG creado de la unión disjunta de los grafos x y y uniendo las fuentes de x y y para crear la Fuente de P_c y uniendo los objetos(sinks) de x y y para crear el objeto(sink) de P_c.
La composición serie $S_c = S_c(x, y)$ de los TTG's x, y al unir el sink (t) de x con el Fuente (s) y. La Fuente de x se convierte en la Fuente de S_c y el sink de y se convierte en el sink de S_c.

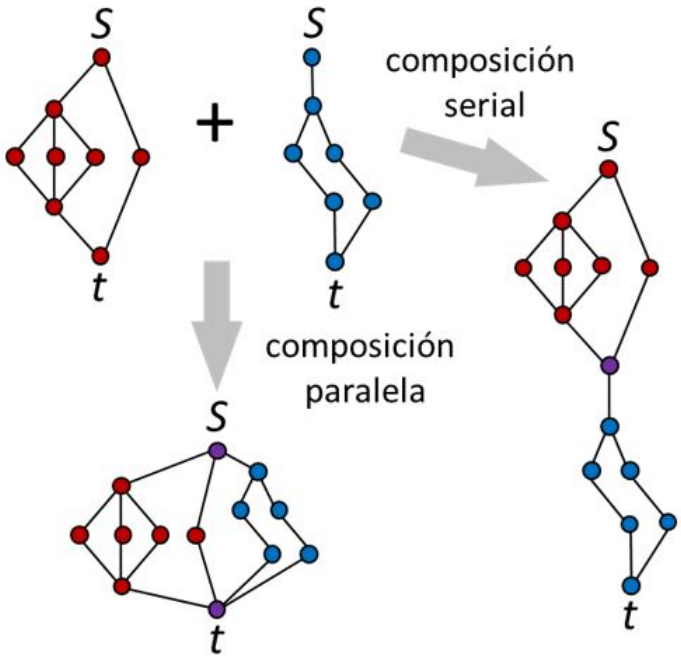


Figura 2.10: Composición serial y paralela.

2.4.6. Coloreo de grafos

El problema de coloreo de grafos tiene aplicaciones en áreas tales como: problemas de programación, asignación de frecuencias, planificación Byskov (2005), Dvorák et al. (2009), Mertzios & Spirakis (2014).

El coloreo de un grafo $G = (V, E)$ es una asignación de colores a sus vértices. El coloreo es válido si los vértices adyacentes siempre tienen colores diferentes. Un k-coloreo de G es un mapeo de V al conjunto {1, . . . ,k} de k colores. El número cromático de G denotado por (G) es el valor mínimo k tal que G tiene un k-coloreo adecuado. Si $(G) = k$, entonces se dice que G es k-cromático o k-coloreable.
Sea $G = (V, E)$ un grafo. G es un grafo bipartito si V se puede dividir en dos subconjuntos U_1 y U_2, llamados conjuntos de partes, de modo que cada arista de G une a un vértice de U_1 con un vértice de U_2. Si $G = (V, E)$ es un grafo k-cromático, entonces es posible dividir V en k conjuntos independientes V_1, V_2, . . . ,V_k, llamadas clases de color, pero no es posible dividir V en k - 1 conjuntos independientes.

Capítulo 3

Estado del arte

En esta sección se mencionan algunos de los resultados presentados en la literatura de artículos científicos y trabajos de investigación relacionados con el tema del problema de satisfactibilidad proposicional, así como del problema de satisfactibilidad incremental. También se describe una primera clasificación (Figura 3.1) sobre los aspectos más relevantes al tratar estos problemas. La revisión del estado del arte se ha estructurado considerando la clasificación mostrada en la (Figura 3.1). Cada uno de los aspectos incluídos se analizan mas a detalle en las siguientes secciones.

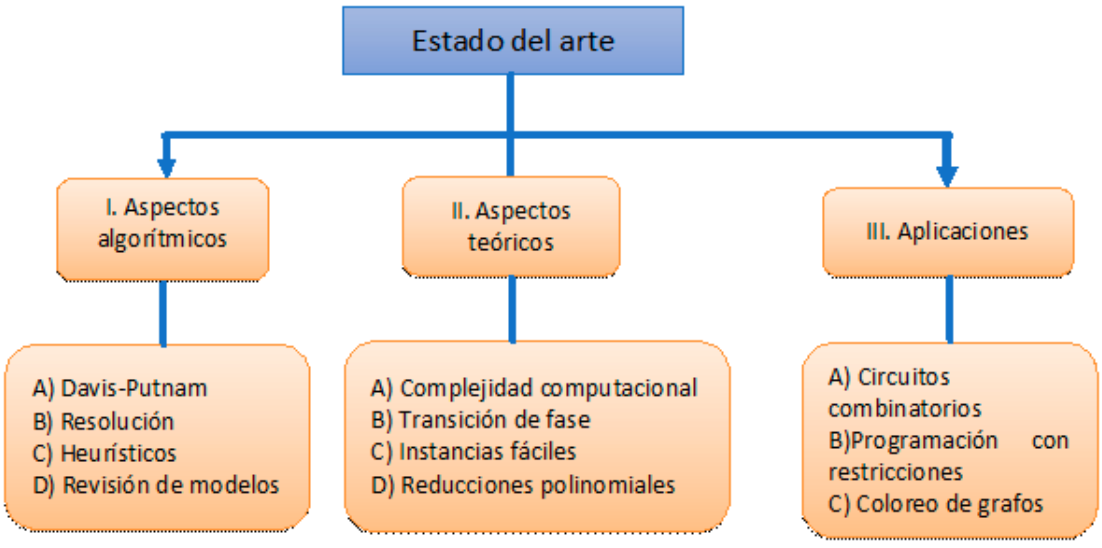


Figura 3.1: Diferentes aspectos tratados en la Satisfactibilidad Incremental.

3.1. Aspectos algorítmicos

Davis & Putman (1960) desarrollaron un algoritmo para comprobar la satisfactibilidad de fórmulas de lógica proposicional en FNC. El algoritmo usa una forma de resolución en la cual las variables son elegidas iterativamente y eliminadas mediante la resolución sobre el conjunto de cláusulas de la fórmula.
Davis et al. (1962) desarrolló el algoritmo DPLL, un algoritmo completo basado en el retroceso hacia atrás (backtracking) que sirve para decidir la satisfactibilidad de las fórmulas de lógica proposicional en forma normal conjuntiva; es decir, para resolver el problema SAT, al igual que lo hacía el algoritmo anterior de Davis y Putnam.
El algoritmo de Davis y Putnam se convirtió en uno de los algoritmos cláicos para decidir la Satisfactibilidad de fórmulas proposicionales y suele ser un proceso común a aplicarse en la mayoría de los algoritmos completos, como es el caso, en los algoritmos de SatzLi & Anbulagan (1997), SATOZhang (1997), GRASP Marques-Silva & Sakallah (1999) y ChaffMoskewicz et al. (2001).
En Hooker (1993) se plantea el problema de satisfactibilidad incremental. En el artículo se describe una implementación basada en el método de Davis-Putnam-Loveland para comprobar la satisfactibilidad del conjunto original de cláusulas, y considerando el caso incremental de ir adicionando nuevas cláusulas.
Con respecto a la clase de algoritmos heurísticos que son propuestas que intentan hallar soluciones de manera rápida (en tiempos polinomiales de cómputo), para el problema ISAT se han aplicado búsquedas locales, tales como: GSAT y WalkSAT Ansótegui & Manyà (2003), así como algoritmos evolutivos Mouhoub & Sadaoui (2007) y algoritmos aleatorios Drias (1998).
En Mohamed-El (2004) se propone un método basado en Optimización Extrema para resolver ISAT donde también se proporcionan resultados experimentales para instancias ISAT. Los resultados obtenidos se comparan con los resultados de un algoritmo para SAT convencional.
En Alban & Anbu (2008), se presenta un algoritmo para SAT que se extiende para ISAT y que se basa en el diagnóstico incremental de sistemas de eventos discretos. El tiempo de ejecución de usar un algoritmo de SAT resulta ser menor que en un enfoque no incremental, los resultados obtenidos muestran tiempos de respuesta y exactitud razonables.
En Mouhoub & Sadaoui (2007) se comprueba si una solución (modelo) a un problema de SAT sigue siendo un modelo cada vez que se agrega un nuevo conjunto de cláusulas y también si soluciones iniciales pueden modificarse dinámicamente y de manera eficiente para satisfacer tanto a la antigua fórmula como a las nuevas cláusulas. Se estudia la aplicabilidad de métodos sistemáticos y de aproximación para resolver ISAT.
Nadel et al. (2012) propone un método para evitar la eliminación de variables que se reintroducen en el pre-procesamiento incremental, debido a que no se puede calcular la secuencia de instancias anteriores en base solo a la dependencia entre instancias previas, también presenta una serie de experimentos con benchmarks industriales donde se muestra que la propuesta de estos autores es más rápido que varias de las

alternativas conocidas.
En el trabajo de Auderman & Laurent (2003) se divide el espacio de búsqueda en subespacios, asignados éstos de forma sucesiva a los solucionadores de SAT que permiten la importación y exportación de cláusulas entre los solucionadores.
En Belov et al. (2013) se presentan y se prueban diferentes técnicas de pre-procesamiento que se pueden aplicar en la satisfacción Booleana (SAT), que también se relaciona con la extracción MUS (Subfórmulas Mínimamente Insatisfactibles). La propuesta permite formalizar las condiciones para la corrección en la preservación de las aplicaciones de técnicas de pre-procesamiento que no son aplicables directamente. Se evalúa experimentalmente el efecto del pre-procesamiento en relación a la extracción del grupo MUS.
En Whittemore et al. (2001) se presenta un método para resolver simultáneamente varias instancias SAT estrechamente relacionadas, utilizando técnicas propias de la Satisfactibilidad incremental (ISAT).
HoonSang & Fabio (2005) presenta como filtrar las cláusulas de conflicto que se puedan ir transfiriendo favorablemente a instancias sucesivas, también presenta una técnica para destilar cláusulas que se reenviarán aunque falle la comprobación sintáctica.
En Wieringa (2011) se reduce el número de modelos en base a un enfoque semántico, lo que mejora la calidad de las cláusulas, es decir, su capacidad para evitar el examinar grandes regiones de espacio de búsqueda.
En Niklas & Niklas (2003) se muestra la modificación a un solucionador de SAT moderno, que permite resolver una serie de instancias SAT relacionadas de manera eficiente utilizando técnicas relacionadas con la verificación de modelos acotados (BMC).
En Armin et al. (2006) se comparan enfoques de revisión de modelos acotado (BMC). Las codificaciones BMC son acotadas a un tamaño lineal para ser usadas en una lógica temporal lineal (LTL), lo que ofrece un mejor rendimiento que los enfoques alternativos basados en autómatas.
En McMillan (2003) se propone un método SAT basado en conteo de modelo y el cálculo de interpolación como un proceso para inferir el grado de satisifactibilidad de una fórmula.
En Wieringa et al. (2009) se investigan enfoques para paralelizar la revisión de modelos acotados (BMC) dentro de entornos de memoria compartida, así como considerando agrupamiento de estaciones de trabajo.
Es de notar que existe una literatura limitada donde se muestre el uso mixto de métodos, esto es, utilizar combinaciones de algoritmos completos y heurísticos para resolver ISAT.

3.2. Aspectos teóricos sobre SAT e ISAT

Es común que las instancias de prueba para algoritmos que resuelven 3-SAT, se separen en tests-sets de instancias de prueba en casos satisfactibles y no satisfactibles Gent & Walsh (1999). En el caso de 3-SAT aleatorio, se muestra que la transición de fase entre instancias satisfactibles y no satisfactibles, se produce alrededor de $k = m/n = 4.26$

donde m = No. de cláusulas y n = No. de variables.
En Cheeseman et al. (1991) se construyen conjuntos de prueba con n=20, 50, 100 variables que se consideran como problemas pequeños. Para cada valor fijo de n, se generan 1000 instancias de prueba. Y para problemas grandes, se generan 100 instancias de prueba.
En Beckert et al. (2000) se muestran resultados de la complejidad para el problema 2-SAT, éstos indicaron que la complejidad del problema 2-SAT sobre fórmulas regulares en FNC depende en la forma de los signos que ocurren en la entrada. Cualquier información negativa, es decir, con variables con signos complementados, hace que el problema no tenga un método trivial de solución.
En Kullmann (1999) se introducen nuevos métodos para el análisis de instancias en 3-FNC, así como también nuevas técnicas algorítmicas tomando pruebas para la complejidad en tiempo de 3-SAT con el límite superior del caso más desfavorable de 1.5045 * n, donde n es el número de variables en la fórmula de entrada y se añaden nuevas cláusulas con 2 y 3 literales, llamadas cláusulas bloqueadas, generalizando así la regla de resolución extendida.
En Gent & Walsh (1999) se muestra que el comportamiento de transición de fase, similar a la observada en varios problemas NP-completos como es el 3-SAT aleatorio, se produce dentro de la jerarquía polinomial más alta, como es el caso del problema 2-QSAT aleatorio.
En Jian et al. (2014) la transición de fase se investiga durante la compilación de instancias k-satisfactibles sobre lenguajes manejables con métodos empíricos. A través de experimentos, se concluyó que la longitud implícita principal y la intercambiabilidad de la solución son impactadas de forma crucial sobre los tamaños de los resultados de la compilación.
En De Ita et al. (2016) se muestran algunos casos donde el problema ISAT se puede resolver en tiempo polinomial. Se propone un algoritmo para resolver el problema 2-ISAT y que permite analizar cuál es el límite superior de la complejidad en tiempo de ciertas instancias específicas.
En Audemard et al. (2007) se propone una nueva clase de instancias difíciles de SAT. Estas instancias se construyen utilizando un nuevo grafo basado en la representación de la fórmula booleana en Forma Normal Conjuntiva. Extiende el concepto de grafo de implicación de las cláusulas binarias (2-SAT) al caso general.
En De Ita et al. (2007) se presenta el diseño de una reducción polinomial para ciclos intersectados restringidos que se vuelven a dibujar como ciclos embebidos. El método resultante cuenta modelos para una 2-FNC y se puede utilizar para otros problemas de conteo tales como conjuntos independientes, conteo de coloreo de grafos y conteo de vértices cubiertos.

3.3. Algunas aplicaciones de SAT e ISAT

En cuanto a las aplicaciones de SAT e ISAT, estos se han utilizado para la verificación y optimización en la automatización del diseño electrónico, para la Satisfactibilidad en circuitos combinatorios. Un caso especial ha sido la aplicación de SAT en problemas de coloreo en Grafos.
En De Ita & Castillo (2013) se presentan aplicaciones de 2-SAT para reconocer el 3-coloreo de algunas clases de instancias grafos.
En Bart et al. (1992) se demuestra que el procedimiento GSAT puede usarse para resolver problemas difíciles generados aleatoriamente, también puede resolver los problemas de Satisfactibilidad estructurada rápidamente, resolviendo las codificaciones del problema de coloración de grafo tales como el problema de colocación de n-reinas usando inducción booleana, y probando el caso de 8 reinas sobre un algoritmo de satisfacción con restricciones.
En Prakash & Narendra (2011) se utiliza una técnica de codificación polinomial 3-SAT para el k-coloreo de un grafo, dando un enfoque de reducción del grafo 3-Colorable a la codificación 3-SAT, donde se utiliza el enfoque de restricción de vértice y el enfoque de restricción de aristas para codificar el grafo k-coloreable en expresiones 3-FNC.
En Prabhat & Mingsong (2009) se presenta una metodología para compartir el aprendizaje a través de múltiples propiedades mediante el desarrollo de técnicas eficientes para el agrupamiento de propiedades, la sustitución de nombres y el envío selectivo de cláusulas de conflicto.
En Larrabee (1992) se describe el método de satisfactibilidad booleana para generar patrones de prueba sobre fallas individuales detectadas en circuitos combinatorios.
En (Whittemore, Joonyoung y Karem, 2005) se propone un solucionador de satisfactibilidad que es adecuado para problemas de verificación y optimización en la automatización del diseño electrónico.
En Chandrasekar & Michael (2007) se trata de resolver el problema de generar una suite de pruebas completa para el retardo de trayectoria de falla (PDF). Se propone aplicar Satisfacción Incremental que aprende de las implicaciones de la lógica estática, de cláusulas específicas del segmento, y de núcleos de insatisfacción de cada PDF parcial no comprobable. Estas técnicas de aprendizaje improvisan la generación de prueba para el retardo de trayectoria de falla que tienen segmentos comunes comprobables y no comprobables, con el fin de lograr aceleraciones para la generación de pruebas PDF a circuitos grandes.
En Joonyoung (2001) se aplica el motor de satisfactibilidad incremental sobre una serie de aplicaciones en electrónica y automatización de diseño, realizando además el análisis de tiempo, pruebas de fallas de retardo y pruebas de fallas atascadas.
En Alexander et al. (2016) se introduce un algoritmo para la resolución incremental SAT bajo suposiciones, llamado última instancia Incremental (UI-SAT). UI-SAT utiliza además el sistema SatELite, y en el artículo se aplica sobre instancias con cláusulas unitarias de un conjunto disponible de 186 instancias generadas por un comprobador de modelos con límites incrementales. El algoritmo supera a los enfoques existentes sobre las ins-

tancias disponibles generadas por una aplicación industrial en la validación de hardware. En Benedetti & Bernardini (2011) se propone una técnica para mejorar los resultados en la realización de tareas de codificación y resolución con la revisión de modelos acotados de forma incremental (IBMC) implementado en NuSMV, el cual integra técnicas de comprobación de modelos basadas en diagrama de decisión binario (BDD). Su sistema para SAT está diseñado con una arquitectura abierta y donde se cambió el módulo codificador para realizar la codificación incremental, el módulo convertidor de una forma normal conjuntiva (FNC) se aplica para obtener el etiquetado adecuado de la fórmula procedente del codificador, y luego, el módulo decodificador puede reconstruir a partir de la tabla de símbolos y de un modelo testigo.
En Disch & Scholl (2007) se hace la comprobación de equivalencia de salida de circuitos basada en SAT, haciendo uso de técnicas de SAT incremental. Se presenta una aplicación de la revisión de modelos acotados (BMC) que se basa en un análisis de estructuras de circuitos compartidos. Se presentan heurísticas que tratan de maximizar el beneficio de la solución SAT incremental. En la aplicación se buscan pedidos en los que se comprueba la equivalencia de diferentes salidas de circuitos.

Parte II

Diseño metodológico y Resultados

Capítulo 4

Diseño metodológico

En este capítulo se presentan los diferentes pasos aplicados en la metodología de la investigación con el fin de dar cumplimiento a los objetivos planteados en el proyecto de tesis.

1. Una de las primeras tareas realizadas en el trabajo doctoral consistió en demostrar que el problema del 2-ISAT(satisfactibilidad incremental iniciando con una dos forma normal conjuntiva) es un problema de la clase de complejidad NP-Completo. Donde la fórmula lógica de entrada F se divide en dos partes; una subfórmula formada por una 2-FNC K, y la otra parte por ϕ. Lo anterior permite considerar el problema SAT(F) en dos fases.

2. Una vez que se ha dividido F en K y ϕ, se realiza una tarea de pre-procesamiento sobre F. El pre-procesamiento consiste en aplicar las reglas: regla de cláusula subsumida, resolución binaria y reducción por literales puros. El propósito de este pre-procesamiento es simplificar la fórmula ϕ. La ventaja de aplicar estas reducciones es el que ellas trabajan dentro de cotas de tiempo polinomial.

3. Al revisar SAT($K \wedge \phi$), en una primera fase, se revisa SAT(K), donde K es una 2-FNC. En esta primera fase se construyen estructuras combinatorias que serán usadas en la segunda fase del problema. En la segunda fase, se adiciona la fórmula ϕ, y se plantea determinar SAT($K \wedge \phi$).

4. Otra tarea consistió en diseñar propuestas algorítmicas basadas en el problema de satisfactibilidad incremental (2-ISAT) para modelar el problema del 3-coloreo sobre topologías específicas de grafos planares. Entre las topologías que encontramos adecuadas para modelar el 3-coloreo en base a 2-ISAT, se tienen: grafos outerplanars, grafos serial-paralelo y árboles poligonales.

5. Los modelos desarrollados en la tarea anterior motivaron el diseñar una propuesta algorítmica para resolver el problema de coloreo sobre grafos planares. Al atacar este problema de coloreo, se propuso un nuevo método lógico para la revisión de

usar 3 o 4 colores al colorear grafos planares. A este método se le llamó formación de cadenas triangulares.

6. El método de cadenas triangulares se basa en la formación de una forma normal conjuntiva F_G. F_G es formada por las restricciones de igualdades y desigualdades entre variables simbólicas que cubren los vértices de secuencias de caras triangulares en el grafo planar de entrada (ver anexo B.4).

Las topologías de grafos que consideramos en este trabajo, fueron:

Grafos outerplanars: (aquellos que no contienen a K_4 ni a $K_{2,3}$ como 'minors'). En nuestro caso, aplicamos cualquiera de los procedimientos de complejidad en tiempo lineal para re-dibujar un grafo outerplanar como un arreglo poligonal,(ver Anexo B.2).

Grafos serial-paralelo: Se pueden determinar nuevas restricciones que se van añadiendo a la fórmula inicial ($K \wedge \phi$), de forma que nos garantice el poder resolver la nueva instancia de 3-SAT de una forma incremental, y requiriendo un tiempo de cómputo acotado polinomialmente, (ver Anexo B.3) .

Árboles poligonales: Se realiza una caracterización tipo árbol para un árbol poligonal, esto nos ayuda a proponer un algoritmo eficiente para su 3-coloreo, (ver Anexo B.1).

Grafo Planar: Son grafos que pueden dibujarse en un plano sin que haya cruce de aristas salvo en sus puntos finales de incidencia. Los grafos planares también se describen como grafos que no aceptan a K_4 (grafo completo de 4 vértices) ni a $K_{3,3}$ (grafo completo bipartito con dos conjuntos de 3 vértices) como subgrafos del grafo planar.

Capítulo 5

Resultados sobre la Complejidad computacional del problema 2-ISAT

En este capítulo se analiza la complejidad computacional del problema de satisfactibilidad incremental (ISAT), iniciando con la versión más simple del mismo, que es cuando se parte de una fórmula inicial en 2-FNC, problema conocido como: 2-ISAT. Primero se verá que el problema 2-ISAT se puede comparar con el problema del 3-coloreo de un grafo, y que de hecho, existe una reducción polinomial en tiempo para transformar el problema del 3-coloreo a una instancia del problema 2-ISAT. También se presentan reglas de simplificación que trabajan sobre formas normales conjuntivas.

5.1. Reducción polinomial del 3-coloreo al problema 2-ISAT

Al proponer reducciones del 3-Coloreo al problema 2-ISAT debemos considerar la naturaleza dinámica del problema 2-ISAT. El problema 2-ISAT se compone de dos fases consecutivas. En la primera fase, la entrada es una 2-FNC K. El propósito de esta fase es determinar SAT(K), así como construir cualquier estructura computacional o lógica, que denotaremos mediante A_K. La estructura A_K se utilizará al procesar la fórmula de entrada de la segunda fase. La principal restricción sobre la construcción de A_K es invertir solo un tiempo polinomial sobre la longitud de la 2-FNC, esto es, sobre $|K|$, de modo que toda la primera fase sea ejecutada en un tiempo de cómputo acotado superiormente por un polinomio sobre la variable $|K|$.
En la segunda fase del problema 2-ISAT, la entrada es una 3-FNC ϕ donde todas las variables de la fórmula ϕ ya han aparecido en K, esto es, $v(\phi) \subseteq v(K)$. El propósito de esta segunda fase es determinar ahora SAT($K \wedge \phi$) . La solución de SAT(K) se puede usar, al igual que A_K, para acelerar la revisión de SAT($K \wedge \phi$).
Lema 5.1. El 3-Coloreo es polinomialmente reducible al problema SAT($K \wedge \phi$), con K una 2-FNC y ϕ una 3-FNC.

Prueba. Sea $G = (V, E)$ un grafo donde $n = |V|$, $m = |E|$. Definimos las variables lógicas $x_{v,c}$ para indicar que al vértice $v \in V$ se le ha asignado el color $c \in \{1, 2, 3\}$. Para cada vértice $v \in V$, tres variables lógicas $x_{v,1}$,$x_{v,2}$,$x_{v,3}$ son creadas. Por tanto habrá $3 * n$ variables booleanas en $v(K) \cup v(\phi)$. Definimos primero las restricciones que formarán a la fórmula K. Para cada arista $e = \{u, v\} \in E$, u y v deben tener un color diferente. Esta restricción se modela mediante las siguientes tres cláusulas binarias: $(\neg x_{u,1} \vee \neg x_{v,1}) \wedge (\neg x_{u,2} \vee \neg x_{v,2}) \wedge (\neg x_{u,3} \vee \neg x_{v,3})$. Existen $3 * |E|$ cláusulas binarias de esta clase.
Otra clase de restricciones binarias permitirá definir el hecho de que cada vértice no debe tener más de un color. Esta restricción se modela mediante las siguientes tres cláusulas binarias: $(\neg x_{v,1} \vee \neg x_{v,2}) \wedge (\neg x_{v,2} \vee \neg x_{v,3}) \wedge (\neg x_{v,3} \vee \neg x_{v,1})$, para cada vértice $v \in V$. Se tendrán $3 * |V|$ cláusulas binarias de esta clase. Ambos conjuntos de $3 * (|V| + |E|)$ cláusulas binarias conforman la fórmula K que estará en 2-FNC.
En este caso, SAT(K) no es suficiente para determinar un 3-coloreo de G, ya que aunque K fuera satisfactible, no hay un 3-coloreo de G deducible de SAT(K).
Para construir soluciones del 3-coloreo de G, una 3-FNC ϕ debe estar formada por las cláusulas que modelan la restricción de que a cada vértice se le debe asignar al menos un color. Entonces, para cada vértice $v \in V$ la siguiente cláusula es generada: $(x_{v,1} \vee x_{v,2} \vee x_{v,3})$. ϕ tiene $|V|$ 3-cláusulas. Además, cada uno de los $3 * n$ variables de $v(K)$ tiene solo una ocurrencia en ϕ.
Esta reducción se realiza en tiempo polinomial sobre el tamaño n y m, ya que consiste en crear $3 * (n + m)$ cláusulas binarias para K y (n) cláusulas ternarias para ϕ. Además, se cumple que G tiene un 3-Coloreo si y solo si $(K \wedge \phi)$ es satisfactible.

5.2. La NP-completitud del problema 2-ISAT

Una vez que se ha mostrado la existencia de la reducción polinomial del 3-coloreo al problema 2-ISAT, es directo el hecho de que 2-ISAT será un problema NP-Completo, dado que el 3-Coloreo lo es.
Teorema. 5.2. 2-ISAT es NP-Completo.

Prueba. La pertenencia de 2-ISAT a la clase de complejidad NP, proviene del hecho de que SAT está en la clase NP, ya que un algoritmo no determinista necesita proponer una asignación sobre $v(K)$, y verificar en tiempo polinomial si tal asignación satisface a ($K \wedge \phi$), ya que $v(\phi) \subseteq v(K)$. Esto sucede independientemente de las dos fases que componen a 2- ISAT.
Se ha visto (en el Lema 5.1) que el 3-Coloreo se reduce en tiempo polinomial a SAT($K \wedge \phi$), y puesto que el 3-Coloreo es un problema NP completo, entonces SAT($K \wedge \phi$) es también NP-completo. Si una estructura A_K permite determinar SAT($K \wedge \phi$) en un tiempo acotado superiormente por un polinomio de $|K \cup \phi|$, entonces el 3-Coloreo sería solucionable en tiempo polinomial. Así, si 2-ISAT es polinomialmente resoluble también el 3-Coloreo lo sería. Esto demuestra que 2-ISAT es NP completo.

Sin embargo, si se considera que la estructura A_K es suficientemente expresiva para contener, por ejemplo, el conjunto de modelos de K. Entonces revisar SAT(ϕ) en A_K, se puede resolver en tiempo polinomial ya que consiste en eliminar de A_K las asignaciones que hagan falsa a la fórmula ϕ. Por lo tanto, la complejidad total para revisar SAT($K \wedge \phi$) depende de la expresividad de A_K, con la limitación de que $|A_k|$ debe tener un tamaño acotado polinomialmente en $|K|$ y en $|Var(K)|$.

Del teorema anterior, no se espera (a menos que P=NP) construir un algoritmo eficiente para resolver 2-ISAT, aunque en la primera fase, K ya ha sido procesada y se han construido algunas estructuras A_K, describiendo las dependencias lógicas entre las variables en K.
Notar que cada componente de SAT($K \wedge \phi$) es polinomialmente resoluble. Por ejemplo, SAT(K) se resuelve en tiempo lineal sobre el tamaño $|K|$, al ser K una 2-FNC. Similarmente, al ser ϕ una 3-FNC, donde cada una de sus variables ocurre una sola vez, entonces SAT(ϕ) es un problema trivial, ya que cada variable es pura en ϕ y por tanto, ϕ siempre es satisfactible. Sin embargo, la composición de ambos problemas, esto es, SAT($K \wedge \phi$) es un problema NP-completo.
Al ser SAT($K \wedge \phi$) un problema NP-completo, no se espera que exista una estructura A_K de longitud polinomial en $|K|$, que permita resolver SAT($K \wedge \phi$) de manera eficiente. Por ejemplo, si A_K es el conjunto de modelos de K, entonces la revisión de SAT($K \wedge \phi$) se puede realizar de manera eficiente, ya que consiste en eliminar de A_K todas las asignaciones que falsifican a alguna de ϕ. Sin embargo, el conjunto de modelos A_K no tiene siempre un tamaño polinomial en $|K|$. En el peor de los casos, la construcción del conjunto de modelos de K requeriría un tiempo exponencial sobre $|K|$.
Dado el conjunto de restricciones ($K \wedge \phi$), determinar SAT($K \wedge \phi$) requiere (hasta ahora) de un tiempo exponencial en el tamaño $|K \cup \phi|$, lo que a su vez, nos permite determinar un 3-coloreo de G. El carácter no determinista en el proceso de asignar un color a cada vértice de G se refleja al requerir un tiempo exponencial para resolver su respectiva instancia 3-SAT. Sin embargo, para topologías de grafos especiales, como es el caso de los grafos outerplanar, se pueden determinar nuevas restricciones que se van añadiendo a la fórmula ($K \wedge \phi$), de forma que nos garantiza el poder resolver la nueva instancia de 3-SAT de una forma incremental y requiriendo un tiempo total acotado polinomialmente. Por eso, en los capítulos siguientes abordamos el problema de buscar casos restrictivos en 2-ISAT que si puedan resolverse de manera eficiente.

Capítulo 6

Propuestas algorítmicas para el coloreo de grafos planares

En este capítulo se presentan diversas topologías de grafos no dirigidos, en donde instancias del problema 2-ISAT son usadas para modelar el 3-coloreo sobre clases especiales de topologías de grafos. Además, las instancias del problema 2-ISAT pueden resolverse en un tiempo polinomial de cómputo. Con esto, mostramos la utilidad de 2-ISAT para modelar y resolver de forma eficiente diversos casos del 3-coloreo de grafos.

6.1. El Grafo de las caras internas de un Grafo Planar

Un grafo planar G tiene un conjunto de regiones cerradas no intersectadas $F(G) = \{f_1, \dots f_k\}$ llamadas caras. Cada cara $f_i \in F(G)$ es representada por el conjunto de aristas, denotada por $E(f_i)$, que limitan su área interior. Toda arista $\{u, v\}$ en G que no es la frontera de alguna cara de G se llama arista acíclica.
Dos caras f_i, f_j en $F(G)$ son adyacentes si tienen aristas en común, esto es, $(E(f_i) \cap E(f_j)) \neq \emptyset$. De lo contrario, son caras independientes. Dos aristas acíclicas son adyacentes si comparten un vértice común. Una arista acíclica es adyacente a una región f_i si solo tienen un vértice común. Un conjunto de caras es independiente si cada par de ellas es independiente. Dado un grafo planar G se construye su grafo de caras internas $G_f = (X, E(G_f))$ de la siguiente manera

1. Cada cara f_i tiene asociado un vértice $x \in X$.
2. Hay una arista $\{u, v\} \in E(G_f)$ uniendo dos vértices adyacentes de X, cuando sus caras correspondientes (o aristas acíclicas) son adyacentes en G.
3. Las aristas acíclicas no serán representadas en el grafo dual.

A G_f se le llama el grafo de caras internas del grafo planar G. Notar que G_f no es el grafo dual de G, ya que en la construcción de G_f la cara externa no es considerada. El

grafo de caras internas G_f de G proporciona un mapeo de la relación entre las caras de G, y es útil en la búsqueda de los patrones de grafos 3-coloreables. Notar que G_f también es un grafo planar, con vértices que representan caras de G, como podemos ver en los grafos de las Figuras 6.1 y 6.2.

En la Figura 6.1. hay 5 ruedas poligonales cuyos centros están marcados por el único vértice centro de la rueda. También hay un subgrafo poliédrico cuyo centro es el polígono formado por el conjunto de vértices 6, 7, 8, 9, 10. Cualquiera de estas ruedas es una rueda impar si tiene un número impar de caras de ejes, de lo contrario, es una rueda par. En la Figura 6.2 se ve el grafo de caras internas del grafo de la Figura 6.1. Se puede notar como cada rueda de G forma un ciclo en G_f.

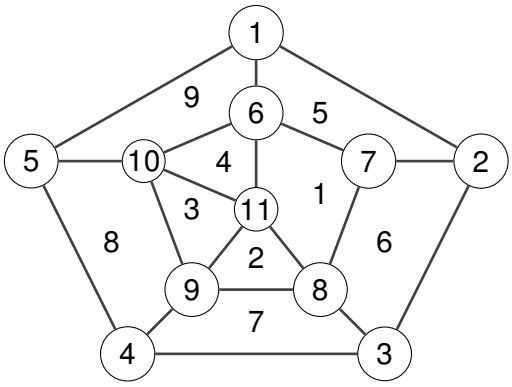


Figura 6.1: G con caras identificadas

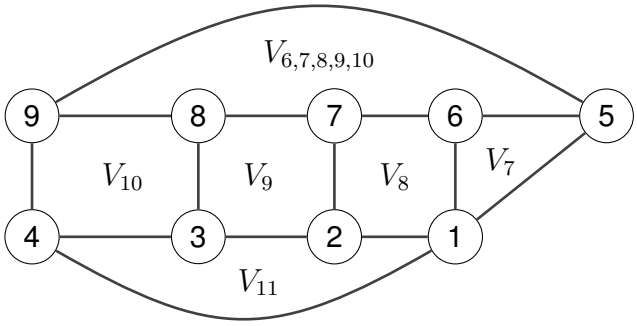


Figura 6.2: El grafo de cara-interna G_f

Al fijar un embebimiento de G, las aristas en G que inciden con la cara exterior se llamarán aristas externas (o aristas frontera) de G. Las caras de G que se delimitan por aristas frontera son llamadas caras frontera de G.
Dado un embebimiento de G, se particiona su conjunto de vértices en Ext(G) e Int(G). Ext(G) $\subseteq V$ es el conjunto de vértices que inciden con la única cara exterior de G. Mientras que los vértices en el embebimiento de G que quedán encerrados por aristas y por tal, no son incidentes con la cara externa, son llamados vértices internos y se denotan como Int(G).

Cuando un grafo planar G puede dibujarse de forma que todos sus vértices sean incidentes con la cara exterior, entonces G es llamado grafo outerplanar Oubiña & Zucchello (1984).
Un problema relevante a considerar en este trabajo de tesis, es el coloreo de grafos planares, y para este caso, las aristas acíclicas pueden eliminarse del grafo planar, ya que éstas no incrementan el número de colores obtenidos al colorear las caras de un grafo planar. De hecho, después de ser coloreadas las caras de un grafo planar, se pueden adicionar las aristas acíclicas y los vértices que se adicionan requieren a lo más dos colores para ser coloreados, ya que todo grafo acíclico es 2-coloreable.

Un patrón gráfico básico que aparece en grafos planares es a lo que llamamos *rueda*. Un subgrafo rueda se forma por un vértice distinguido (llamado vértice centro de la rueda) que es adyacente a los vértices que lo rodean formando un ciclo. Cada cara de la rueda (llamada un eje de la rueda) es un triángulo. Así, hay dos clases de vértices en la rueda; el vértice centro y los vértices que forman el ciclo.
Una rueda en un grafo planar G es representado mediante un ciclo en el grafo de caras internas G_f. Las ruedas con un número par de caras triangulares son 3-coloreables. Mientras que cualquier grafo planar conteniendo a K_4 o a ruedas con un número impar de caras triangulares requiere 4 colores para ser coloreado propiamente. Sin embargo, esas topologías no son los únicos casos de grafos planares 4-coloreables.
Introducimos un coloreo típico para una rueda, al asignar el primer color a su vértice centro. Los colores 2, 3 son asignados de manera alterna a los vértices de su ciclo. Este coloreo comienza en cualquier cara triangular de la rueda, y sigue en dirección opuesta de las manecillas del reloj y a través de los vértices del ciclo. Solo cuando el último vértice del ciclo es visitado se determina si un cuarto color es o no necesario.
Extendemos la clase de ruedas considerando cualquier polígono como una cara de eje de la rueda. Este tipo de rueda se llamará *rueda poliédrica*. En este tipo de ruedas hay vértices en el ciclo que rodea al vértice centro, que no son adyacentes con el centro. Diferenciamos los vértices del ciclo en una rueda poliédrica como vértices de eje si son adyacentes al vértice centro; y como extra-eje cuando no son adyacentes al centro. El vértice centro será un vértice incidente con todas las caras de la rueda poliédrica. Para las aristas en una rueda poliédrica, tenemos las aristas del ciclo y las aristas incidentes al centro, las cuales son llamadas aristas de radio.
Una rueda poliédrica de un grafo planar G se representa por un ciclo en su grafo de caras internas G_f. Sin embargo, un ciclo en G_f también puede codificar otro tipo de rueda de G. Por ejemplo, el caso de una rueda de G donde el centro es un polígono en lugar de un solo vértice, a lo que llamaremos un subgrafo poliédrico. Véase sección B.4 del apéndice B para observar ejemplos de ruedas poliédricas.

6.2. Arboles Poligonales

Dado un grafo planar G, si sus caras internas $G(F) = \{f_1, ..., f_k\}$ pueden arreglarse como una estructura de árbol donde en lugar de nodos tenemos polígonos, y donde dos polígonos consecutivos comparten exactamente una arista (como se ve en la Figura 6.3), entonces a este tipo de grafos se les llama un árbol poligonal.
El reconocimiento de estructuras repetitivas 'patrones' en los grafos es esencial para el diseño de algoritmos combinatorios eficientes. Por ejemplo, los patrones básicos de los grafos que se estudiarán aquí son polígonos que siguen una estructura de árbol.
Muchos problemas difíciles pueden resolverse de manera eficiente en grafos que podrían no ser árboles, pero en cierto sentido, tienen una topología suficientemente arborescente Stefan (2008). Por ejemplo, un árbol poligonal permite el diseño de un algoritmo eficiente para el 3-coloreo de sus vértices. Para esto, ampliamos la definición de árboles

poligonales introducido en De Ita et al. (2018) con la siguiente caracterización de un árbol poligonal $G_T = (V, E)$, (ver Figura 6.3).
Caracterización de árboles poligonales

i. Todo componente acíclico en G_T se deja fuera de las caras internas de los polígonos.
ii. Dos polígonos son adyacentes si comparten una arista común, un vértice común, o están unidos por una sola arista que une un vértice de cada polígono.
iii. Cualquier cadena de polígonos adyacentes no debe formar un ciclo de polígonos. Significa que el camino de los polígonos adyacentes sigue la estructura de un árbol donde cada polígono podría ser sustituido por un solo vértice para formar un árbol.

El hecho de que un árbol poligonal tenga una caracterización tipo árbol, nos ayuda a proponer un algoritmo eficiente para su 3-coloreo.

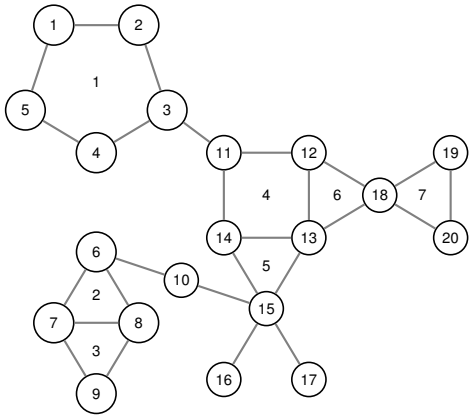


Figura 6.3: Un árbol poligonal

6.2.1. Coloreo de un árbol Poligonal

Se presenta la codificación del 3-coloreo de un grafo poligonal como un problema de satisfactibilidad incremental, considerando la clásica reducción polinomial de 3-coloreo a una instancia 3-Satisfactible.
Supongamos como entrada un grafo conectado $G = (V, E)$, con $n = |V|$ y $m = |E|$, así como un orden sobre los vértices que permite aplicar la bpf de manera determinista. Por ejemplo, comenzando la búsqueda con un nodo $v \in V$ de grado mínimo y visitando los nodos de grado más bajo cada vez que hay múltiples posibles nodos para visitar y con una etiqueta menor para el caso de varios vértices con mismo grado minimal. Sea $G' = bpf(G)$ el grafo generado por bpf. Sea T_G el árbol de expansión de G'. Note que $V(T_G) = V(G)$. Sea $C = \{C_1, C_2, \ldots, C_k\}$ el conjunto de ciclos fundamentales encontrados por $bpf(G)$. Si G es un grafo acíclico, entonces $T_G = G'$ y $C = \emptyset$.
Lema 6.2.1. Si $G' = bpf(G)$ es acíclico o G' es bipartito, entonces G' es 2-coloreable.

Prueba. Si G' es acíclico o contiene solo ciclos fundamentales de longitud par, entonces

G es bipartito y por tanto es 2-coloreable, ya que los vértices en G pueden ser coloreados por niveles, es decir, todos los vértices en el mismo nivel tienen el mismo color y los nodos de dos niveles consecutivos estarán coloreados con los dos colores de forma alternada.

Consideremos ahora un árbol poligonal G_p cuya especificación para cualquier 3 coloreo ha sido codificado por $\phi_0 = (K \wedge \phi)$, una 3-FNC. Presentamos un orden para realizar el 3-coloreo de los vértices de G_p. Esto es estratégico para encontrar un 3 coloreo válido. Sea F_r el conjunto de vértices estratégicos para ser coloreados primero. F_r se forma de la siguiente manera:

1. Todos los puntos finales de las aristas comunes entre polígonos adyacentes deben estar en F_r.
2. Además, los vértices comunes entre polígonos adyacentes unidos por un vértice se agregan a F_r.
3. Los puntos finales que son los puentes entre dos polígonos adyacentes se agregan a F_r.
4. Finalmente, todos los elementos en F_r se ordenan en caminos que formen trayectorias maximales entre vértices en F_r

Sea $x_{v,c}$ la variable lógica que denota que al vértice v se le asigna el color $c \in \{1, 2, 3\}$. Sea $Tres = \{1, 2, 3\}$ el conjunto que contiene tres colores posibles. Para cada vértice $v \in V(G_p)$ un conjunto $Tabu(v)$ es asociado. $Tabu(v)$ indica los colores prohibidos para el vértice v. De hecho, $Tabu(v)$ contiene los colores asociados a los vértices vecinos de v. Notar que $|Tabu(v)| < 2, \forall v \in V$, porque cuando $|Tabu(v)| = 2$, la cláusula $(x_{v,1} \vee x_{v,2} \vee x_{v,3})$ asigna un color de forma inmediata al vértice v.
Cuando G_f es un árbol, decimos que G (su correspondiente grafo planar) es un árbol poligonal López et al. (2018). Esto significa que, aunque G tiene ciclos, todos esos ciclos pueden estar arreglados como un árbol, cuyos nodos son polígonos en lugar de solo vértices de G. En este caso, un orden para visitar las caras del grafo planar proporciona un procedimiento eficiente para el 3-coloreo de G, como se afirma en el siguiente teorema.
Teorema 6.3.1 Si el grafo de caras internas de un grafo planar G tiene una topología de árbol, entonces G es 3 coloreable.

Prueba. Sea G_f el grafo de caras internas de un grafo planar G. Cada cara de G es representada por un nodo $x \in V(G_f)$. Toda cara es 3-coloreable ya que es en sí, un ciclo simple. Además, toda arista acíclica de G, representada por un nodo de G_f , también es 3-coloreable, ya que todo grafo acíclico es 2-coloreable. Proponemos ahora un procedimiento para asignar un 3-coloreo a G al ir visitando en pre-orden los nodos de G_f. Se 3-colorea primero la cara del nodo padre de G_f y después, las caras de sus hijos. En cada nivel actual, se consideran las dos caras adyacentes (padre e hijo en G). Ambas regiones tienen dos vértices comunes, los puntos extremales x,y que conforman su arista común. Estos vértices comunes se colorean primero, y luego, los vértices restantes de las caras padre-hijo. Así, estos vértices restantes de las caras tendrán a lo

más dos colores prohibidos. Nótese que no hay un par de vértices adyacentes u y v en cualquiera de las dos caras, tal que $\{u, v\} \subseteq (N(x) \cap N(y))$, porque entonces $\{x, y, u, v\}$ formaría K_4 y este subgrafo no puede ser parte de ningún árbol poligonal. Por lo tanto, para todos los vértices restantes en ambas caras, está disponible al menos un color de los tres posibles en $Tres$. El proceso del 3-coloreo finaliza cuando todos los nodos del árbol de G_f se han visitado en pre-orden.

Por ejemplo, al procesar el grafo de la Figura6.3, se construye el siguiente conjunto F_r ={{14,13},{12,13},{7,8},{18},{3,11}}.
Se ordenan los vértices en F_r formándose caminos maximales:13-18-12-13-14-11-3,7-8.
A partir de considerar los valores en F_r se agregarán nuevas cláusulas unitarias a $(K \wedge \phi)$. Estas cláusulas unitarias fijan un color para los vértices de G_p. Se procesa cada camino maximal en F_r, para ir formando un coloreo sobre sus vértices:
$\{13, 18\} \rightarrow (X_{13,1}), (X_{18,2}) \rightarrow (\neg X_{12,1}), (\neg X_{12,2}), (\neg X_{14,1}), (\neg X_{15,1}) \rightarrow (X_{12,3}) \rightarrow (\neg X_{11,3})$.
$\{18, 12\} \rightarrow \emptyset$.
$\{12, 13\} \rightarrow \emptyset$.
$\{13, 14\} \rightarrow (X_{14,2}) \rightarrow (\neg X_{11,2}), (\neg X_{15,2}) \rightarrow (X_{11,1}), (X_{15,3}) \rightarrow (\neg X_{3,1})$.
$\{14, 11\} \rightarrow \emptyset$.
$\{7, 8\} \rightarrow (X_{7,1}), (X_{8,2}) \rightarrow (\neg X_{6,1}), (\neg X_{6,2}) \rightarrow (X_{6,3}), (X_{9,3})$.

El proceso para el 3-coloreo de árboles poligonales se muestra en el algoritmo 2 y fue publicado en López et al. (2018). En el Apéndice B Anexos B.1, B.2 y B.3 se muestra el proceso para asignar un color al elemento actual de F_r.

Algoritmo 2 3-$coloreo(T_G)$

Eliminar cualquier vértice de grado máximo dos, así como cualquier subgrafo acíclico de G {Estos subgrafos pueden ser coloreados al final del proceso}
while $((Fr \neq \emptyset)$**and**$(\Phi_0 \neq \emptyset))$ **do**
Sea $e = push(Fr)$;
$e = u$ OR $e = \{u, v\}$
if u (and v when $e == \{u, v\}$) ya ha sido coloreado **then**
Continuar {considerar el siguiente elemento de Fr}
else
if $e = \{u\}$ **then**
for $v \in N(x)$ and $a \in Tabu(v) - Tabu(u)$ **do**
$s = \{(x_{u,a})\}$
end for
else
if u ya ha sido coloreado **then**
$b = min\{Tres - Tabu(v)\}$
end if
else
if v ya ha sido coloreado **then**
$a = min\{Tres - Tabu(u)\}$
end if
else
if $(Tabu(v) == Tabu(u)) == \emptyset$ **then**
$a = 1; b = 2;$
end if
else
if $Tabu(u) \neq \emptyset$ **then**
$a = min\{Tres - Tabu(u)\}; Tabu(v) = Tabu(v) \cup \{a\};$
end if
else
if $Tabu(v) \neq \emptyset$ **then**
$b = min\{Tres - Tabu(v)\}; Tabu(u) = Tabu(u) \cup \{b\}; a = min\{Tres - Tabu(u)\}$
end if
end if
end if
Sea $s = \{(x_{u,a}), (x_{v,b})\}$ una asignación que determina el color para u, v
Aplicar $\Phi_0 = UP(\Phi_0, s)$
end while
Regresar los vértices eliminados en el paso 1;
Aplicar 2-coloreo asignándoles un color diferente al de sus vecinos

6.3. Grafos Outerplanar

Un grafo outerplanar G_o puede ser dibujado en una forma planar y donde cualquier par de ciclos básicos son independientes, o uno de ellos está embebido dentro de otro ciclo. Por lo tanto, G_o es planar y se puede dibujar de forma que sus vértices sean incidentes a la cara externa del grafo, como puede observarse en el grafo siguiente.

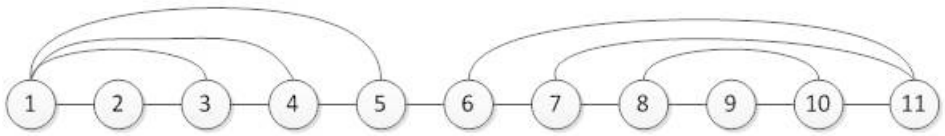


Figura 6.4: Grafo outerplanar.

El generar un dibujo planar de un grafo puede considerarse como un problema por sí solo, en parte porque los algoritmos para dibujar grafos tienden a crear un grafo planar embebido como primer paso, y en parte, porque dibujar puede ser una aplicación dependiente al problema a tratar. En nuestro caso, aplicamos cualquiera de los procedimientos de complejidad en tiempo lineal para re-dibujar un grafo outerplanar como un arreglo poligonal. Por ejemplo, aplicando el algoritmo en Boyer & Myrvold (2004).
El tener representado un grafo outerplanar G_o como un arreglo poligonal P_G, nos permite definir un orden sobre las aristas frond que aparecen en P_G. En este caso, una arista frond de P_G es la arista común entre dos polígonos adyacentes en P_G. En la figura siguiente se hace la representación poligonal del grafo outerplanar de la Figura 6.4.

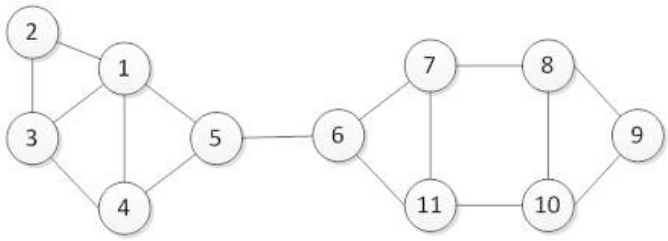


Figura 6.5: Grafo poligonal.

Consideremos ahora un grafo outerplanar G_o que ha sido codificado por una 3-FNC $\phi_0 = (K \wedge \phi)$. Sea F_r el conjunto de aristas (frond) de G_o ordenadas en post-orden. Las aristas en F_r permitirán definir nuevas cláusulas unitarias que se irán agregando a la cláusula inicial $(K \wedge \phi)$. Estas nuevas cláusulas unitarias permitirán determinar un color para cada vértice que conforma una arista en F_r.
De acuerdo con el orden dado a las aristas en F_r, cada arista $\{u, v\}$ se toma de F_r y se le asigna un color a los vértices finales: u y v, en caso de que alguno de ellos no haya sido ya coloreado.

El proceso para asignar color a los vértices u, v para una arista $\{u, v\} \in F_r$ se muestra en la sección B.2 del apéndice B.
El algoritmo 3 muestra los pasos a seguir para realizar un 3-coloreo sobre grafos outerplanars, este algoritmo fue publicado en De Ita et al. (2018).

Algoritmo 3 3-coloreo sobre grafos outerplanar

Elimine cualquier vértice de grado dos como máximo, así como cualquier arista acíclica de G {estos elementos pueden ser coloreadas al final del proceso}
while ($F_r \neq \emptyset$) y ($K \neq \emptyset$) **do**
  if u y v han sido coloreados **then**
    Continuar {considerar el siguiente frond}
  else
    if u ha sido coloreado **then**
      $b = min\{Tres - Tabu(v)\}$
    end if
  else
    if v ha sido coloreado **then**
      $a = min\{Tres - Tabu(u)\}$
    end if
  else
    if $((Tabu(v) == Tabu(u)) == \emptyset)$ **then**
      $\{a = 1; b = 2\}$
    end if
  else
    if $(Tabu(u) \neq \emptyset)$ **then**
      $a = min\{Tres - Tabu(u)\}; Tabu(v) = Tabu(v) \cup \{a\};$
    end if
  else
    if $(Tabu(v) \neq \emptyset)$ **then**
      $b = min\{Tres - Tabu(v)\}; Tabu(u) = Tabu(u) \cup \{b\}; a = min\{Tres - Tabu(u)\}$
    end if
  end if
  Sea $s = \{(x_{u,a}), (x_{v,b})\}$ una asignación que determine el color de u, v
  Aplicar $K = UP(K, s)$
end while
Regresa a los vértices eliminados en el paso 2.
Se aplica un procedimiento de 2-coloreo para elementos restantes

En la sección B.2 del apéndice B se ilustra la aplicación de este algoritmo 3 sobre un grafo outerplanar de entrada.

6.4. Propuesta algorítmica para grafos serial-paralelo

Un grafo serial-paralelo de dos terminales (TTSPG) es un grafo que puede ser construido por una secuencia de composiciones: serial y/o paralelo, iniciando a partir de un conjunto de copias del grafo de una sola arista K_2, identificando sus dos vértices terminales, como puede observarse en la Figura B.18.
Un grafo TTSPG permite representar de forma natural circuitos (eléctricos, electrónicos, de comunicación, etc.) que combinan composiciones seriales y paralelas.

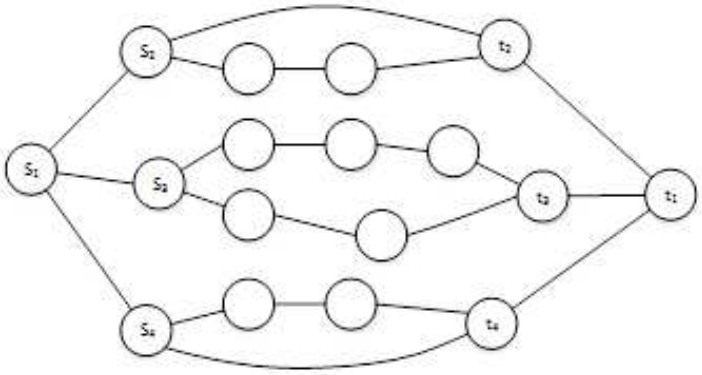


Figura 6.6: Grafo serial- paralelo de entrada.

Nuestra propuesta algorítmica para construir un 3-coloreo de un grafo serial-paralelo se basa en transformar el grafo de entrada a una fórmula en 3-FNC del tipo $\varphi_0 = (K \wedge \phi)$, definiendo así la primer fórmula φ_0 para un proceso de satisfactibilidad incremental. Posteriormente, se irán agregando nuevas cláusulas para conformar una serie de fórmulas $\varphi_i \; i = 1, \ldots, k$. Los modelos para $((K \wedge \varphi_0) \wedge \varphi_1 \wedge \cdots \varphi_k)$ determinarán un 3-coloreo válido del grafo serial-paralelo de entrada.
Nuestro algoritmo se basa en la reducción de un 3-coloreo de un grafo a una fórmula $(K \wedge \phi)$ justificado por el lema 5.1, que nos propone una forma de codificar el 3-coloreo de un grafo a través de la fórmula: $(K \wedge \phi)$, con K una 2-FNC y ϕ una 3-FNC.
El carácter no determinista en el proceso de asignar un color a cada vértice de G en la reducción presentada (lema 5.1), se refleja al requerir un tiempo exponencial para resolver su respectiva instancia 3-SAT. Sin embargo, para topologías de grafos especiales, como es el caso de los grafos serial-paralelo, se pueden determinar nuevas restricciones que se van añadiendo a la fórmula inicial $(K \wedge \phi)$, de forma que nos garantice el poder resolver la nueva instancia de 3-SAT de una forma incremental, y requiriendo un tiempo de cómputo acotado polinomialmente.
Recordemos que a todo vértice v en G se asocia un conjunto de colores prohibidos $Tabu(v)$. Así, cuando a un vértice v se le asigna un color: $Color(v)$ entonces a todo vértice vecino u de v, se coloca en $Tabu(u)$ el color asignado a v, ya que este color será ahora prohibido para u.

Sea G un TTSPG (grafo serial-paralelo) , y sea $\varphi_0 = (K \wedge \phi)$ la codificación de su 3-coloreo que se infiere de la aplicación del lema 5.1. En esta clase de topologías, los vértices claves para definir un 3-coloreo sobre TTSPG, son los vértices que aparecen en las parejas (s_i, t_i) =(fuente, objeto) de cada componente serial-paralelo.
Cuando en cada camino de s a t no hay otras componentes paralelas, decimos que la componente (s, t) es simple, es decir, cada camino de s a t es un camino simple, y cada vértice del camino tendrá grado 2 (excepto para s y t). En una componente serial-paralela, cuando alguno de los vértices s o t es eliminado, entonces la componente se convierte en un árbol (un subgrafo acíclico), con raíz en s si t es eliminado, o viceversa.
Lema 6.4.1 Cada componente serial de un grafo serial-paralelo es 2-coloreable.

Prueba. Las partes seriales en un grafo serial-paralelo tienen vértices con grado 2 o menos, y por lo tanto, las partes seriales en un TTG son 2 coloreables (por lema 6.2.1), y entonces, pueden ser coloreados usando 2 colores de manera alternada desde s a t.
Nuestra propuesta algorítmica forma el conjunto $Fr = \{(s_{0,3}), (s_{1,3} \vee t_{1,3}),$ $(s_{2,3} \vee t_{2,3})), \ldots, (s_{k,3} \vee t_{k,3})\}$ conteniendo la cláusula $(s_{i,3} \vee t_{i,3})$ asociada con los dos vértices terminales de cada componente serial-paralelo: (s_i, t_i) de G, ordenadas por su aparición en G, de las componentes más externas hacia las más internas.

Regla de asignación del color 3. El algoritmo asigna a cada uno de los componentes serial-paralelo (s, t) de un grafo, el color 3 al vértice s, o al vértice t, o a ambos. Nuestra propuesta inicia asignando color 3 al primer vértice fuente s_0, y esto se hace al adicionar la cláusula $(s_{0,3})$ a φ_0.

Lema 6.4.2. Si $Color(s_i) = 3$ entonces $3 \notin Tabu(t_{i+1})$.

Prueba. (Por contradicción) Supongamos que se asigna $Color(s_i) = 3$, y que por tanto $3 \in tabu(t_{i+1})$, pero esto significa que s_i sería adyacente a t_{i+1}. Pero el vértice s_{i+1} está entre s_i y t_{i+1} y por tal, s_i y t_{i+1} no son adyacentes.

Lema 6.4.3. Dado un subgrafo G conteniendo una sola dos-terminal (s, t), si se cumple $(s_{i,3} \vee t_{i,3})$ entonces el grafo G es 3-coloreable.

Prueba. Asuma que $\{s, t\} \notin E(G)$, asignando $f(s) = f(t) = 3$, y formando ahora $G' = G - \{s, t\}$ se forma un grafo acíclico el cual es 2-coloreable (usando sólo los colores 1 y 2), y entonces G es 3-coloreable. Si $\{s, t\} \in E(G)$, sea $f(s) = 3$ y $G' = G - \{s\}$ es un árbol con raíz en el nodo t. G' puede colorearse por niveles iniciando con el color 1 para t y usando los colores 2 y 1 por niveles hasta llegar a todas las hojas de G', formándose así, un 3-coloreo para G.

Teorema de Correctés. Si para toda dos-componente (s_i, t_i) de un grafo TTSPG G se cumple la cláusula: $(s_{i,3} \vee t_{i,3})$ entonces el grafo G es 3-coloreable.

Prueba. Por inducción sobre el número de dos componentes en el grafo G.

i) Para una sola dos-componente (s_1, t_1), este caso se demostró anteriormente (Lema 6.4.3).
ii) Se supone que la propiedad se cumple en todo grafo con n dos-componentes (Hipótesis inductiva)
iii) Sea G un grafo con $n+1$ dos-componentes. Y sea (s_0, t_0) la componente más externa en G. Nuestro algoritmo asigna $f(s_0) = 3$, y $\forall y \in N(s_0)$ se marca $Tabu(y) = Tabu(y) \cup \{3\}$, formándose así, $G' = G - \{s_0\}$. G' contiene n dos-componentes, cumpliéndose la hipótesis inductiva para G'. Además, no puede darse el caso de que exista una dos-componente $(s_i, t_i) \in G'$ tal que $Tabu(s_i) = Tabu(t_i) = 3$, ya que esto implicaría que se tienen las aristas $\{s_0, s_i\}$ y $\{s_0, t_i\}$ en G, lo que contradice que (s_0, t_0) y (s_i, t_i) son dos-componentes distintas en G. Así que o bien se cumple $(s_{i,3})$ o bien $(t_{i,3})$, y por tanto también se cumple: $(s_{i,3} \vee t_{i,3})$. Resulta entonces que G es 3-coloreable.

Como toda cláusula $(s_{i,3} \vee t_{i,3})$, $i = 1, \ldots, k$ se cumple, así como: $(s_{o,3})$, entonces SAT$((K \wedge \varphi_0) \wedge F_r)$ determina un 3-coloreo para cualquier grafo serial-paralelo G. Nuestro algoritmo inicia ejecutando $K' = UP((K \wedge \varphi_0), (s_{0,3}))$, lo que permite determinar de forma automática los colores válidos, dentro del conjunto $\{1, 2, 3\}$, para algunos de los vértices de G. Si durante la ejecución de $K' = UP((K \wedge \varphi_0), (s_{0,3}))$ no se asigna color a alguna dos-componente (s_i, t_i), entonces de forma incremental y de acuerdo al orden definido en F_r, se adiciona una nueva cláusula unitaria (u), ejecutando ahora: $K' = UP(K', (u))$, donde (u) será $(s_{i,3})$, o bien $(t_{i,3})$, según los conjuntos tabús de las variables: s_i ,t_i. Así, nuestra propuesta algorítmica encuentra de forma automática y basada en el teorema anterior, un 3-coloreo para G López & De Ita (2019). Presentamos el pseudo-código de esta propuesta algorítmica para grafos serial-paralelo.

Algoritmo 4 3-coloreo para grafos serial-paralelo

Identificar vértices s(fuente) y t(objeto) por niveles de anidamiento, formándose el conjunto F_r
Remover todo vértice de grado < 3
Aplicar $K' = UP((K \wedge \varphi_0), (s_{0,3}))$
while $(Fr \neq \emptyset)$ y $(K' \neq \emptyset)$ **do**
  Selecciona componente $(s_i, t_i) \in F_r$ del nivel más externo
  if $Tabu(s_i) = 3$ **then**
    $\{Color(t_i) = 3; v = t_i\ ; G = G - t_i\}$
  else
    $\{Color(s_i) = 3; v = s_i\ ; G = G - s_i\}$
  end if
  $\forall y \in N(v) : Tabu(y) = Tabu(y) \cup \{3\}$
  Aplicar $K = UP(K, s)$
end while
Aplica 2-coloreo sobre vértices de grado < 3

En la sección B.3 del apéndice B se ilustra la aplicación de este algoritmo 4, para el coloreo de un grafo serial - paralelo.

6.5. Coloreo de Grafos Planares

En términos del coloreo de grafos planares, si se considera un grafo planar desconectado G, se cumple que la unión del coloreo de sus componentes conectadas es también un coloreo válido para G. Por lo anterior, consideraremos a partir de ahora como entrada para nuestra propuesta algorítmica, un grafo planar G conectado y sin aristas acíclicas. En esta sección, empezamos nuestro análisis para buscar patrones básicos en un grafo planar para realizar su coloreo. Como las ruedas son un patrón básico general al formar grafos planares, empezamos considerando el coloreo de ruedas simples.
Lema 6.5.1. La unión de ruedas simples de longitud par, donde sus vértices centros son independientes una con otra, es 3-coloreable.

Prueba. El coloreo típico sobre las ruedas puede extenderse considerando la unión de las ruedas asignando el primer color a todos los vértices centro de las ruedas, ya que estos forman un conjunto independiente en el grafo. Aristas comunes entre las ruedas solo están dadas por las aristas del ciclo de las ruedas. Si los vértices centro son removidos del grafo, ya que fueron coloreados, el subgrafo restante es bipartito porque solo tiene las aristas de ciclos de longitud par, y por tanto, el subgrafo remanente es 2-coloreable. Así, obtenemos un 3-coloreo para este tipo de grafos usando diferentes colores entre los vértices centro y los vértices del ciclo.

El siguiente lema propone un método para el 3-coloreo de componentes acíclicas, cuyos vértices tienen a lo más un color prohibido.
Lema 6.5.2. Un componente acíclico, donde sus vértices tienen como restricción a lo más un color, es 3-coloreable.

Prueba. Consideremos la componente acíclica como un árbol enraizado en v_r. Un coloreo en pre-orden inicia a partir de v_r, donde $Color(v_r) = MIN\{Three - Tabu(v_r)\}$. Si avanzamos en pre-orden para cada nuevo nivel a colorear, todo vértice x en el nuevo nivel tendrá a lo más 2 colores restringidos, el color de su nodo padre y el color que podría existir en $Tabu(x)$. Así, siempre está disponible un color de los 3 posibles en $Three$. El proceso del 3-coloreo finaliza, cuando todos los nodos del árbol han sido visitados en pre-orden.

Si un grafo planar G no tiene una topología de árbol poligonal, significa que hay ciclos en G_f, y por lo tanto, ruedas poliédricas en G. Para este tipo de grafos planares, es posible reconocer patrones gráficos 3- coloreables.

Lema 6.5.3 Toda rueda poliédrica es 3-coloreable.

Prueba. Si r_x es una rueda poliédrica, entonces hay una cara del eje que no es triangular. Por lo tanto, hay al menos un vértice $v_e \in V(r_x)$ que no es adyacente al vértice centro v_x de la rueda, de lo contrario todas las caras serían triangulares. El grafo $R_{v_e} = (r_x - v_e)$ es un arreglo poligonal, y entones es 3-coloreable por el Lema 6.5.2. y Teorema 6.3.1. Cualquier 3-coloreo típico para R_{v_e} puede extenderse a un 3-coloreo para r_x si v_e tiene el mismo color que v_x, porque en un coloreo típico el color del vértice centro no se usa en los vértices del ciclo. Por lo tanto, el color del centro no se ha utilizado para los vértices en $N(v_e)$.

La unión de ruedas 3-coloreables no es necesariamente 3-coloreable. Por ejemplo, cada rueda individual en el grafo de la Figura B.22 del apéndice B es 3-coloreable, cuando la cara común se asume como parte de cada rueda, y por el Lema 6.5.3, cada una de ellas es 3-coloreable. Sin embargo, el grafo final (unión de las ruedas) es 4-coloreable como veremos en la siguiente sección.

6.6. Método para la 3 o 4 colorabilidad de secuencia de ruedas

El proceso de identificación de vértices y el uso de cadenas Kempe han sido dos de las herramientas más usadas en los algoritmos de coloreo sobre grafos planares.
Definición. Una identificación $< x, y >$ de vértices x y y en un grafo G significa una operación en G que elimina x y y y agrega un nuevo vértice adyacente a esos vértices a los que x y y fueron adyacentes. Una contracción de una arista $\{x, y\}$ es un caso especial de la identificación $< x, y >$.
El propósito en una cadena Kempe es encontrar un camino maximal en el grafo que pueda ser 2-coloreable.
Definición. Una cadena Kempe es definida como: Si v es un vértice con color a, entonces la cadena Kempe$-(a, b)$ de G que contiene al vértice v es el subconjunto maximal conectado de $V(G)$ que contiene a v, y cuyos vértices están todos coloreados, ya sea con el color a o con el color b.
Una cadena Kempe puede verse como una técnica de re-coloreo, considerando una parte conectada maximal C_r que fue ya coloreada e intercambiar dos de esos colores para formar un nuevo coloreo C_{r+1}.
Las cadenas Kempe fueron utilizadas para realizar la prueba del teorema de los 4 colores en grafos planares, pero resultó ser una técnica incompleta. Sin embargo, el uso de cadenas Kempe fueron útiles en la prueba computacional asistida de Appel & Haken (1977). En base a un proceso de identificación de dos vértices se puede dar una demostración sucinta de que todo grafo planar es 5-coloreable.
El argumento de Kempe demostró que cinco colores son suficientes para colorear mapas planos y que un contra ejemplo mínimo de la conjetura de cuatro colores (mínimo con

respecto al número p de polígonos en el mapa) no podría contener cualesquiera polígono de dos lados, triángulos o cuadriláteros Whitney & Tutte (1992).

6.6.1. Cadenas Triangulares

Definición Un conjunto S de caras triangulares define una secuencia triangular cuando cualquier par f_1, f_2 de caras triangulares son adyacentes, o hay un subconjunto de caras triangulares adyacentes en S, conectando a f_1 con f_2.

Definición Una secuencia triangular maximal S es una secuencia triangular donde no se pueden agregar más caras triangulares a S sin perder la adyacencia entre caras.

Definición Una cadena triangular t se forma de una tripleta de variables simbólicas $t =< x, y, z >$ que se usan para etiquetar los vértices de una secuencia triangular maximal.

Una cadena triangular t no asigna colores absolutos a los vértices; en su lugar, solo se utilizan tres colores simbólicos diferentes para etiquetar los vértices. Cuando se han etiquetado todos los vértices en una secuencia triangular maximal, es posible determinar si la cadena está coloreada solo por tres colores. De esta manera, generalizamos las cadenas de Kempe usando cadenas triangulares.
Cuando un vértice $u \in V(G)$ se etiqueta por una variable x de una cadena triangular t, tal variable es agregada a los conjuntos $Tabu$ de su vecindad $N(u)$, ya que u ha reservado el color x, y ese color es ahora prohibido para los vecinos de u.
Definición Una cadena triangular es inconsistente cuando dos vértices adyacentes de la secuencia triangular son etiquetados con la misma variable simbólica; en otro caso, la cadena es consistente(satisfactible).

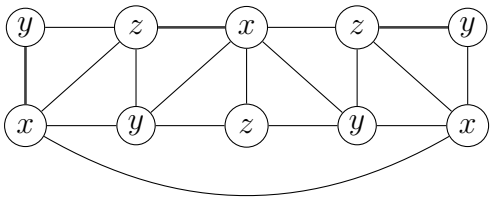


Figura 6.7: Cadena triangular inconsistente

El propósito es cubrir toda la secuencia triangular maximal por cadenas consistentes. Por ejemplo, la cadena triangular en el grafo de la Figura 6.7 es inconsistente, ya que dos vértices adyacentes tienen asignado la misma variable (x). Notar que no importa el orden para asignar variables a los vértices en G. Cualquier cadena triangular para G será inconsistente, ya que G no es 3-coloreable.
Una cadena triangular t cubre todos los vértices en una secuencia triangular maximal

en tiempo lineal sobre el número de los vértices en la secuencia, ya que consiste en asignar una variable a cada vértice en la secuencia y seleccionar en cada iteracción (pero no para las dos primeras iteraciones) al vértice cuyo conjunto $Tabu$ tiene dos variables simbólicas.
Similarmente, la detección de una cadena triangular inconsistente se realiza mientras t está etiquetando los vértices, ya que consiste en encontrar si algún vértice en la secuencia tiene tres variables simbólicas en su conjunto $Tabu$, y en este caso, el vértice se marca como un punto inconsistente en la cadena, para evitar propagar la inconsistencia en los vértices restantes del grafo.
El etiquetado de todas las secuencias triangulares en un grafo se puede hacer en tiempo lineal. Una cadena triangular inconsistente nos permite detectar un patrón gráfico 4 coloreable que se reconoce en tiempo lineal en la longitud de la secuencia triangular.
La recuperación de la inconsistencia de una cadena se puede hacer asignando el cuarto color (en este caso el color 1) a todos los vértices etiquetados por la misma variable cuyo número de vértices libres asociados es el máximo en la cadena.
Este proceso de formación de cadenas triangulares y verificación de sus consistencia, itera mientras haya cadenas inconsistentes a quienes recuperar su consistencia y hasta no obtener más cadenas triangulares inconsistentes. Un problema posterior, es revisar la consistencia de la conjunción de cadenas triangulares, cada una de ellas consistente. Ya que la unión de cadenas consistentes puede formar un sistema insatisfactible según las restricciones que definan las variables involucradas. Si la conjunción de cadenas triangulares consistentes forma un sistema satisfactible, entonces el grafo es 3 coloreable.
Por ejemplo, se puede recuperar la consistencia del grafo de la Figura 6.7 al asignar el color 1 a cada uno de los vértices etiquetados con la etiqueta y, puesto que esta etiqueta tiene ocurrencia máxima dentro de los vértices del grafo. Y después de remover los vértices etiquetados con y del grafo, no es dificil hallar un 3-coloreo sobre el grafo remanente.

6.6.2. Aplicando cadenas triangulares para colorear ruedas poliédricas

En esta sección, presentamos un método lógico para identificar si la conjunción de cadenas triangulares forma un sistema satisfactible. Para presentar nuestra propuesta, consideremos como ejemplo el subgrafo expresado como $(r_x \cup_f r_y)$ formado por dos ruedas r_x y r_y compartiendo una cara común $f = (F(r_x) \cap F(r_y))$ (ver grafo de la Figura 6.8).
Una cadena triangular $t_1 =< x, y, z >$ es asociada a $V(r_x)$, mientras la cadena triangular $t_2 =< a, b, c >$ es asociada a $V(r_y)$ de la siguiente manera. La variable x es asociada al centro de r_x, mientras que la variable a es asociada al centro de r_y. La cadena triangular t_1 se usa para cubrir los vértices de las caras triangulares de r_x, mientras que t_2 cubre los vértices de las caras triangulares de r_y. Se cosntruye una fórmula proposicional con

equivalencias F_G de la siguiente manera.

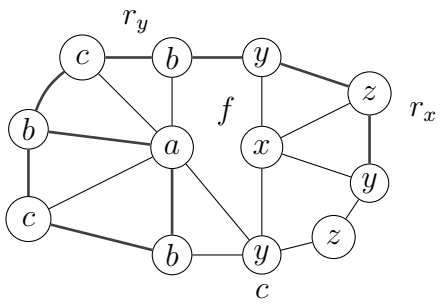


Figura 6.8: Unión de 2 ruedas 3-coloreables

I. Los vértices $V(f)$, que son vértices comunes entre $V(r_x) \cap V(r_y)$, determinan restricciones de igualdad entre sus variables correspondientes: $(y \oplus z) = (b \oplus c)$, donde $(\oplus)$ denota el operador lógico xor. Esto significa que los vértices en $V(r_x) \cap V(r_y)$ definen colores iguales (variables con mismo color).

II. Una arista $e = \{u, v\} \in E(G)$, con puntos finales en $V(r_x)$ y $V(r_y)$, define una restricción de desigualdad entre sus variables correspondientes: $(u \neq v)$. Esto significa que vértices adyacentes entre dos ruedas diferentes define diferentes colores (variables con diferente color).

III. F_G también considera que cualquier par de vértices adyacentes debe tener colores diferentes. Esto se codifica como: $(x \neq y) \wedge (x \neq z) \wedge (z \neq y) \wedge (a \neq b) \wedge (b \neq c) \wedge (c \neq a)$.

IV. En F_G también se agregan las restricciones que definen que cada uno de los vértices en $V(r_x) \cup V(r_y)$ tiene uno de los tres posibles colores, lo que se codifica como: $((x = a) \vee (x = b) \vee (x = c)) \wedge ((y = a) \vee (y = b) \vee (y = c)) \wedge (((z = a) \vee (z = b) \vee (z = c))$.

Notar que cualquier desigualdad puede considerarse como la negación de una restricción de igualdad. Además, las restricciones tipo I-III son cláusulas unitarias. Un par de cláusulas unitarias contradictorias implica la insatisfactibilidad de F_G. Mientras que la satisfactibilidad de F_G determina la existencia de un 3 coloreo válido para G. Cuando F_G es satisfactible, entonces existe una función biyectiva $f_R : \{x, y, z\} \rightarrow \{a, b, c\}$ determinando el 3 coloreo de G.

Para el grafo de la Figura 6.8, las restricciones tipo II define que $(b \neq y) \wedge (a \neq y)$, y entonces $(c = y)$ se infiere de las restricciones tipo IV, que es validado por la única restricción tipo I. Como no hay más restricciones de tipo II, entonces la fórmula F_G puede ser satisfecha por las asignaciones $((c = y) \wedge (a = x) \wedge (b = z))$, o $((c = y) \wedge (a = z) \wedge (b = x))$. Por lo tanto, el grafo es 3 coloreable, donde $\{x, y, z\}$ representa cualquier permutación

de los valores {1,2,3}.

Definición Dos cadenas triangulares son adyacentes, cuando hay aristas con puntos finales etiquetados por variables de ambas cadenas, o cuando hay un vértice etiquetado por variables de ambas cadenas. De lo contrario, las cadenas triangulares son independientes.
Cuando existen caras triangulares con vértices que no son asociados con variables, pero la tripleta actual no puede etiquetarlas, entonces se usa una nueva cadena triangular para etiquetarlas. Este proceso de cubrir vértices de caras triangulares por variables continúa hasta que todos los vértices estén etiquetados por variables.
Definición Un sistema de cadenas triangulares adyacentes es compatible si la fórmula lógica definida por las restricciones (I - IV), formado por las variables etiquetando los vértices de la secuencia triangular, es satisfactible.
Consideremos ahora como ejemplo el grafo en la Figura 6.9. La tripleta $< x_1, y_1, z_1 >$ está asociado a las caras triangulares 1, 2 y 3, de la siguiente manera: $x_1 \rightarrow \{a\}, y_1 \rightarrow \{b, f\}, z_1 \rightarrow \{c, e\}$. Mientras, la tripleta $< x_2, y_2, z_2 >$ está asociado a las caras triangulares 5 y 6 de la siguiente manera: $x_2 \rightarrow \{i\}, y_2 \rightarrow \{d, g\}, z_2 \rightarrow \{h\}$. Siguiendo las restricciones tipo II, se tiene que $(y_1 \neq y_2)$ ni $(z_1 \neq y_2)$. Por lo tanto, $(y_2 = x_1)$ es inferido por la regla IV. Como no hay más restricciones tipo I o II, entonces F_G es satisfactible.

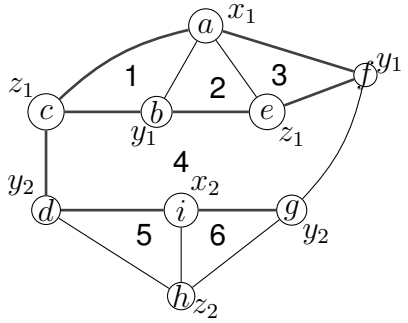


Figura 6.9: $\forall x \in V(G), Tabu(x) = 1$

Uno de los modelos para F_G es $(x_1 = y_2) \wedge (y_1 = x_2) \wedge (z_1 = z_2)$. De esta manera, nuestro método determina que el grafo es 3 coloreable de manera determinista. Esto es contrario al uso de cadenas Kempe que no proporciona un procedimiento determinista.
Cuando F es satisfactible, entonces las variables simbólicas restantes en G se pueden dividir en 3 grupos, como a_1 a_2 y a_3. Si $\forall a_i(\exists v$ etiquetado por $a_i) : Tabu(v) = \{1\}, i = 1, 2, 3$, entonces los colores 2, 3 y 4 son usados para éstos 3 grupos. En otro caso, para cualquier grupo $a_i, i = 1, 2, 3$ formado solo por vértices libres, entonces se puede asignar el color 1 para a_i. Al eliminar los vértices en a_i, el grafo restante será 2 coloreable.
Dado un grafo planar G como entrada, nuestro método lógico trabaja de la siguiente manera. Todas las secuencias triangulares en G tienen que estar cubiertas por cadenas triangulares. Además, una fórmula lógica F_G está formada por la conjunción de las restricciones (I - IV) dadas por las variables etiquetando los vértices en las secuencias triangulares. Después, la propagación unitaria se aplica utilizando las cláusulas unitarias

del tipo de restricciones (I-III) con las cláusulas restantes, hasta determinar si la fórmula resultante es satisfactible De-Ita & López-Ramírez (2019).
Puede notarse que la revisión de la satisfactibilidad de F_G se realiza mediante la propagación del proceso de resolución unitaria en las restricciones tipo I-III contra todas las restricciones en F_G. Así como la aplicación de la transitividad de la relación de igualdad. Por lo tanto, revisar la satisfactibilidad de F_G se realiza en un tiempo lineal sobre el tamaño de la fórmula F_G.

La propuesta de revisión basada en cadenas triangulares puede ser combinado con otros procesos de coloración para construir algoritmos robustos. Por ejemplo, podemos revisar la existencia de ruedas clásicas para asignar el color 1 a los centros de estas ruedas, eliminando los centros de G y marcando a sus vecinos. Al procesar primero las ruedas clásicas, quedarán en el grafo actual sólo ruedas poliédricas, a quienes les aplicamos nuestro método basado en cadenas triangulares.

El teorema T4C para grafos planares garantiza que efectivamente 4 colores son suficientes para colorear los grafos planares Robertson et al. (1997). Nuestra propuesta algorítmica busca vértices críticos que requieran el cuarto color (color 1) y cuyos vecindad sea 3 coloreable. Al realizar este proceso de forma iterativa, nuestro método elimina los vértices críticos que requieren el cuarto color hasta obtener un subgrafo 3 coloreable.

Parte III

Conclusiones

Capítulo 7

Conclusiones

El problema de satisfactibilidad incremental (ISAT) consiste en comprobar si la satisfactibilidad (SAT) de una fórmula lógica se mantiene cuando se añaden nuevas cláusulas, es decir, se considera una generalización del problema SAT al permitir cambios en el tiempo sobre la fórmula de entrada.
En particular, el problema de satisfactibilidad incremental que parte de una 2-FNC (2-ISAT) consiste de dos fases; en la primer fase se considera el problema de determinar SAT(K), siendo K una 2-FNC. La idea es que durante la resolución de SAT(K), se vayan construyendo estructuras combinatorias que serán usadas en la segunda fase del problema. En la segunda fase se adiciona una fórmula ϕ que esta en 3-FNC y que se expresa en base a las variables de K. Y ahora se cuestiona por el valor lógico de SAT($K \wedge \phi$).
Se sabe que SAT(K) se resuelve en tiempo polinomial cuando K es una 2-FNC, en tanto que SAT($K \wedge \phi$) es un caso especial del problema 3-SAT, el cual es un problema NP-Completo.
El primer objetivo de este trabajo de tesis, fue la determinación formal de la complejidad computacional de 2-ISAT.
Para la demostración de la complejidad computacional de SAT($K \wedge \phi$) primero se propuso un algoritmo eficiente para resolver SAT(K), al mismo tiempo que se analizaron que estructuras combinatorias formadas durante la resolución de SAT(K) se podrían re-utilizar al querer determinar SAT($K \wedge \phi$).
Después se analizaron las condiciones en las que SAT($K \wedge \phi$) podría resolverse polinomialmente. Concluyendo a través de una demostración formal, que en el caso general, y al considerar las 2 fases del problema 2-ISAT, éste es un problema NP-completo.
Con esto se dió respuesta a la hipótesis 1 de este trabajo de tesis: Cuando K es una 2-FNC y ϕ es una 3-FNC y no hay ningún tipo de restricciones sobre K y ϕ, entonces SAT($K \wedge \phi$) es un problema NP-completo.
Sin embargo, también se pudo comprobar que hay casos especiales para ϕ donde SAT($K \wedge \phi$) se resuelve eficientemente. Casos simples para la resolución eficiente de SAT($K \wedge \phi$) es cuando K es una 2-FNC y ϕ en también una 2-FNC o bien, cuando las

cláusulas de ϕ se forman sólo por literales puras.
En este trabajo se demostraron casos prácticos donde 2-ISAT se resuelve en tiempo polinomial, pero además, la aplicación de 2-ISAT permite modelar y resolver otros problemas de interés combinatorio. En este trabajo mostramos como 2-ISAT permite modelar y resolver en tiempo polinomial, el 3-coloreo de grafos planares sobre las siguientes topologías: arreglos poligonales, grafos outerplanar y los grafos serial - paralelo.
En este trabajo de tesis se revisó primero los casos donde el coloreo de grafos planares se resuelve eficientemente (en tiempo polinomial de cómputo). Entre estos caso, se tiene que el reconocimiento de un grafo de entrada 2-coloreable se resuelve en tiempo lineal, ya que implica el reconocimiento de que solo ocurran ciclos de longitud par en el grafo.
Similarmente, se revisó el teorema de Grötzsch que indica que cualquier grafo planar libre de triángulos es 3-coloreable. Sin embargo, el reconocimiento del 3-coloreo sobre un grafo planar es un problema clásico NP-Completo. En general, es difícil reconocer entre si se deben usar 3 o 4 colores para colorear de forma propia un grafo planar cuando este contiene caras triangulares, ya que no se conoce al momento, una condición suficiente para reconocer la 3-colorabilidad de un grafo planar.
Con respecto a la hipótesis 2: se relacionó al problema 2-ISAT con el coloreo de ciertas topologías de grafo planares. En este trabajo se mostró que el problema 2-ISAT es adecuado para modelar problemas de coloreo sobre los siguientes grafos planares: outerplanar, serial-paralelo y árboles poligonales. Para todas estas topologías de grafos planares se diseñaron propuestas algorítmicas eficientes, de complejidad lineal sobre la longitud del grafo de entrada, para determinar un 3-coloreo sobre este tipo de grafos.
Los algoritmos resultantes se inspiraron en la construcción del 3-coloreo en dos fases del problema, tal y como nos indicaba el problema 2-ISAT. En la primer fase del problema se modela el 3-coloreo del grafo usando sólo cláusulas binarias, y durante la segunda fase, se van adicionando cláusulas unitaria y/o binarias que utilizan un tercer color para colorear los vértices críticos encerrados por vecindades 2-coloreables.
Para el caso de la topología de árboles poligonales se realiza una caracterización de este tipo de grafos, redibujados como un árbol de ciclos, lo que permite aplicar una búsqueda en post-orden para diseñar un algoritmo eficiente que constuya un 3-coloreo del grafo.
Para el caso de grafos 'outerplanars', aplicamos el procedimiento de re-dibujar un grafo outerplanar como un arreglo poligonal, que sería una topología de grafo donde previamente desarrollamos un algoritmo en tiempo lineal para su 3-coloreo.
En el caso de un grafo serial-paralelo, se determinan nuevas restricciones formadas al asignar un tercer color sobre cláusulas del tipo (fuente $\vee$ objeto) de toda dos-componente paralela del grafo. El colorear los vértices fuente u objeto de las componentes paralelas, permite descomponer el grafo inicial en subáboles acíclcicos. Esta descomposición garantiza el resolver el 3-coloreo de instancias de grafos serial-paralelo de una forma incremental y en tiempos de cómputo acotados polinomialmente.
Con respecto a la hipótesis 2 de este trabajo de tesis: Si la satisfactibilidad incremental podría adaptarse para modelar el coloreo de grafos en topologías especiales, tal hipótesis se acepta totalmente, dado que se construyeron propuestas algorítmicas

eficientes para el 3-coloreo de topologías especiales de grafos planares.
Avanzando en el análisis de topologías de grafos y en el diseño de algoritmos para grafos planares, se concluyó este trabajo de investigación al proponer un nuevo algoritmo para el coloreo de grafos planares.
Para ésta última tarea, se desarrolló un nuevo método de coloreo para grafos planares basado en la construcción de una forma normal conjuntiva F_G. La fórmula F_G es formada por las restricciones de igualdad y desigualdad entre variables simbólicas que cubren los vértices de secuencias de caras triangulares.
Este nuevo método generaliza la técnica de formar cadenas Kempe (cadenas de dos colores), al formar en su lugar, cadenas triangulares de 3-colores representadas por tripletas de variables simbólicas. Estas cadenas triangulares son efectivas para hallar topologías de grafos planares que requieran de un cuarto color para su coloreo.
El método de cadenas triangulares puede combinarse con otras técnicas clásicas para el coloreo de grafos, proporcionando algoritmos robustos para el coloreo de grafos planares.
Con respecto a la hipótesis 3: Si al extender SAT con restricciones lógicas entre variables simbólicas se podría modelar el coloreo de grafos planares, la hipótesis se acepta totalmente, dado que en base a esta extensión de SAT, se ha propuesto un nuevo método para determinar cuando se requiere un cuarto color para el coloreo de grafos planares.

Apéndice A

Artículos publicados

ISSN: 1793-8201

IJCTE 2017 Vol.9(6): 412-416 ISSN: 1793-8201
DOI: 10.7763/IJCTE.2017.V9.1177

A Note for the Two Incremental Satisfia

Cristina López R., Guillermo De Ita L., and F

Abstract—Let K be a two conjunctive normal form and φ a three conjunctive n the same set of variables. It is well known that SAT(K) is in the complexity clas problem. We consider the computational complexity of determining SAT(K□φ) ISAT). We show that this problem is NP-complete even if the number of occurre propose a method to review SAT(K□φ). Our proposal is adequate to solve 2 recognize tractable instances of 2-ISAT.

Index Terms—3-Coloring, incremental satisfiability roblem, 2-ISAT, NP-Complete

Cristina, López R. is with the Faculty of Computer Sciences in Benemérita U México (mail: cristyna2001@hotmail.com).
Guillermo, De Ita. L. is with Faculty of Computer Sciences in Benemérita U deita@ccc.inaoep.mx).
Pedro. Bello L. is with Faculty of Computer Sciences in Benemérita Universi (mail: pb5pbello@gmail.com).

[PDF]

Cite:Cristina López R., Guillermo De Ita L., and Pedro Bello L., "A Note for t *International Journal of Computer Theory and Engineering* vol. 9, no. 6, pp. 412-

PREVIOUS PAPER
Indicators to Measure a Smart Building: An Indonesian Perspective
NEXT PAPER
Profiling and Evaluation of Implicit and Explicit Storm Surge Models

What's New

Jun 03, 2019 News!Vol.9, No.5-Vol.10, No.3 have been indexed by EI (Inspec). [Click]

Oct 16, 2019 News!Vol.11, No.6 has been published with online version. [Click]

Sep 10, 2019 News!Vol.11, No.1-Vol. 11, No. 5 have been indexed by Crossref.

General Information

ISSN: 1793-8201 (Print)
Abbreviated Title: Int. J. Comput. Theory Eng.
Frequency: Bimonthly
DOI: 10.7763/IJCTE
Editor-in-Chief: Prof. Wael Badawy
Executive Editor: Ms. Mia Hu
Abstracting/Indexing: Index Copernicus, Electronic Journals Library, Google Scholar, Ulrich's Periodicals Directory, Crossref, ProQuest, WorldCat, EBSCO, and EI (INSPEC, IET), Cabell's Directories.
E-mail: ijcte@iacsitp.com

International Journal of Computer Theory and Engineering, Vol. 9, No. 6, December 2017

A Note for the Two Incremental Satisfiability Problem

Cristina López R., Guillermo De Ita L., and Pedro Bello L.

***Abstract*—Let *K* be a two conjunctive normal form and φ a three conjunctive normal form, both formulas are defined over the same set of variables. It is well known that *SAT(K)* is in the complexity class P, while *SAT(φ)* is a classic NP-Complete problem. We consider the computational complexity of determining SAT($K \wedge \varphi$) as an incremental satisfiability problem (2-ISAT). We show that this problem is NP-complete even if the number of occurrences of each variable in φ is one. Also, we propose a method to review SAT($K \wedge \varphi$). Our proposal is adequate to solve 2-ISAT problem. Our algorithm allows us to recognize tractable instances of 2-ISAT.**

***Index Terms*—3-Coloring, incremental satisfiability problem, 2-ISAT, NP-Complete problem, SAT problem.**

I. Introduction

One of the fundamental problems in automatic reasoning is the satisfiability problem (SAT) that tries to determine whether a logical propositional formula *F* is (or not) satisfiable. Considering *F* as a conjunctive normal form (CNF) without restriction on the number of literals by clause, SAT(F) is a classic NP-complete problem, even if each clause has at least 3 literals.

The 2-SAT case, that determines the satisfiability of two conjunctive normal forms (2-CNF), is an important tractable case of SAT. Variations of the 2-SAT problem, e.g. in the optimization and counting area, have been essential for establishing frontiers between tractable and intractable problems.

The incremental satisfiability problem (ISAT) consists in verifying whether satisfiability is maintained when new clauses are added to an initial satisfiable formula. ISAT is considered a generalization of the SAT problem since it allows changes of the input formula over time.

We consider ISAT as a dynamic incremental set of clauses: $F_0, F_1, \ldots, F_i$ starting with an initial satisfiable formula F_0. Each F_i results from a change in the preceding formula F_{i-1}, imposed by the 'outside world'.

Even though, the changes can be a restriction (add clauses) or a relaxation (remove clauses), we focus in the restriction case. In our proposal the incremental process is finished when F_i is unsatisfiable or there are no more clauses to be added. ISAT can be used in a large variety of applications that need to be processed in an evolutive environment. This could be the case of applications such as reactive scheduling and planning, dynamic combinatorial optimization, review faults in combinatorial circuits, dynamic constraint satisfaction, and machine learning in a dynamic environment [1].

One idea used on ISAT methods, is to preserve the structures formed when previous formulas were processed, allowing the recognition of common subformulas that they were previously considered. More importantly, it allows the solver to reuse information across several related consecutive problems. The resulting performance improvements make ISAT a crucial feature for modern SAT solvers in real-life applications [2].

Rather than solving related formulas separately, modern solvers attempt to solve them incrementally, since many practical applications require solving a sequence of related SAT formulas [3], [4]. In this article, we consider ISAT as an incremental problem that starts with an initial satisfiable formula $K=F_0$ in 2-CNF. In a second phase, a new formula φ in 3-CNF is added to *K*, both formulas; *K* and φ are defined on the same set of variables. We denote this version of ISAT as the 2-ISAT problem.

As a generalization of SAT, ISAT has been considered as an NP Problem, although until now, the authors have not seen complexity theory studies about the complexity-time differences between SAT and ISAT. For example, it is known that 2-SAT is in the complexity class P. However, it is not known the computational complexity of 2-ISAT.

In [5], we have proposed a method to review the satisfiability of ($K \wedge \varphi$), and we have analysed instances of *K* and φ that allows the existence of polynomial-time procedures. We present here, a study about the threshold for the 2-ISAT problem that could be helpful to understand the border between P and NP complexity classes for instances of 2-ISAT. In general, we show that 2-ISAT is NP-Complete, even if each variable in φ has only one occurrence.

II. The Graph Vertex Coloring Problem

The graph vertex coloring problem consists of coloring the vertices of the graph with the smallest possible number of colors so that no two adjacent nodes receive the same color. If such a coloring with *k* colors exists, the graph is k-colorable. The chromatic number of a graph *G*, denoted as $\chi(G)$, represents the minimum number of colors for proper coloring *G*. The *k*-colorability problem consists of determining whether

Manuscript received July 7, 2017; revised November 23, 2017. This work was supported by the Benemérita Universidad Autónoma de Puebla. Department of Computer Sciences, Puebla, México.

Cristina, López R. is with the Faculty of Computer Sciences in Benemérita Universidad Autónoma de Puebla, Puebla, Mexico (mail: cristyna2001@hotmail.com).

Guillermo, De Ita. L. is with Faculty of Computer Sciences in Benemérita Universidad Autónoma de Puebla, Mexico (e-mail: deita@ccc.inaoep.mx).

Pedro. Bello L. is with Faculty of Computer Sciences in Benemérita Universidad Autónoma de Puebla, Puebla, Mexico (mail: pb5pbello@gmail.com).

DOI: 10.7763/IJCTE.2017.V9.1177

Modelling 3-Coloring of Outerplanar Graphs via Incremental Satisfiability

Citation Data: Electronic Notes in Discrete Mathematics, ISSN: 1571-0653, Vol: 69, Page: 101-108
Publication Year: 2018

Metrics Details

CAPTURES 2

Readers 2
Mendeley ↗ 2

Article Description

A novel method to model the problem of 3-coloring on outerplanar graphs is presented. The proposal is based on the specification of the logical constraints for the 3-coloring of an outerplanar graph in a dinaymic way, resulting in a polynomial-time instance of the incremental satisfiability problem. This proposal can be extended to consider other polynomial-time instances of the 3-coloring problem.

Bibliographic Details

DOI: 10.1016/j.endm.2018.07.014 ↗

AUTHOR(S):
Guillermo De Ita Luna; Cristina López Ramírez; Meliza González Contreras

PUBLISHER(S):
Elsevier BV

Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in DISCRETE MATHEMATICS

Electronic Notes in Discrete Mathematics 69 (2018) 101–108 www.elsevier.com/locate/endm

Modelling 3-Coloring of Outerplanar Graphs via Incremental Satisfiability

Guillermo De Ita Luna, [1,2] Cristina López Ramírez, [1,2]
Meliza González Contreras [2]

Fac. Cs. de la Computación, BUAP
Puebla, México

Abstract

A novel method to model the problem of 3-coloring on outerplanar graphs is presented. The proposal is based on the specification of the logical constraints for the 3-coloring of an outerplanar graph in a dinaymic way, resulting in a polynomial-time instance of the incremental satisfiability problem. This proposal can be extended to consider other polynomial-time instances of the 3-coloring problem.

Keywords: Incremental Satisfiability Problem, Graph Coloring, Outerplanar Graphs, 3-Coloring

1 Introduction

The graph vertex coloring problem consists of coloring the vertices of the graph with the smallest possible number of colors, so that two adjacent vertices can not receive the same color. If such a coloring with k colors exists, the graph

[1] Thanks to SNI and Conacyt.
[2] Email:deita@cs.buap.mx,cris2001@hotmail.com,mcontreras@cs.buap.mx

https://doi.org/10.1016/j.endm.2018.07.014

Bibliographic information

- DOI https://doi.org/10.1007/978-3-319-92198-3

- Publisher Name Springer, Cham
- eBook Packages Computer Science
- Print ISBN 978-3-319-92197-6
- Online ISBN 978-3-319-92198-3
- Series Print ISSN 0302-9743
- Series Online ISSN 1611-3349
- Buy this book on publisher's site

Published in cooperation with

The International Association for Pattern Recognition

SPRINGER NATURE

Modelling 3-Coloring of Polygonal Trees via Incremental Satisfiability

Cristina López-Ramírez(✉), Guillermo De Ita, and Alfredo Neri

Facultad de Ciencias de la Computación, BUAP, Puebla, Mexico
cristyna2001@hotmail.com, anerjas@hotmail.com, deita@cs.buap.mx

Abstract. A novel method to model the 3-coloring on polygonal tree graphs is presented. This proposal is based on the logical specification of the constraints generated for a valid 3-coloring on polygonal graphs. In order to maintain a polynomial time procedure, the logical constraints are formed in a dinaymic way. At the same time, the graph is traversing in postorder, resulting in a polynomial time instance of the incremental satisfiability problem. This proposal can be extended for considering other polynomial time instances of the 3-coloring problem.

AQ1

AQ2

Keywords: Incremental satisfiability problem · Graph coloring
Polygonal tree graphs · 3-coloring

1 Introduction

The graph vertex coloring problem consists of coloring the vertices of the graph with the smallest possible number of colors, so that two adjacent vertices can not receive the same color. If such a coloring with k colors exists, the graph is k-colorable. The chromatic number of a graph G, denoted as $\chi(G)$, represents the minimum number of colors for proper coloring G. The k-colorability problem consists of determining whether an input graph is k-colorable.

The inherent computational complexity, associated with solving NP-hard problems, has motivated the search for alternative methods, which allow in polynomial time the solution of special instances of NP-hard problems. For example, in the case of the vertex coloring problem, 2-coloring is solvable in polynomial time. Also, in polynomial time has been solved the 3-colorability for some graph's topologies, such as: AT-free graphs and perfect graphs, as well as to determine $\chi(G)$ for some classes of graphs such as: interval graphs, chordal graphs, and comparability graphs [7]. In all those cases, special structures (patterns) have been found to characterize the classes of graphs that are colorable in polynomial time complexity.

Graph vertex coloring is an active field of research with many interesting subproblems. The graph coloring problem has many applications in areas such as: scheduling problems, frequency allocation, planning, etc. [1,4,5].

In particular, hexagonal chains are the graph representations of an important subclass of benzenoid molecules, unbranched catacondensed benzenoid

J. F. Martínez-Trinidad et al. (Eds.): MCPR 2018, LNCS 10880, pp. 1–10, 2018.
https://doi.org/10.1007/978-3-319-92198-3_10

IEEE

Revista del IEEE América Latina

ESPAÑOL ENGLISH PORTUGUES

Patrocinador:

CTEEP

Home

Bienvenidos a la Revista IEEE América Latina, una revista electrónica que está siendo publicada en español y portugués por la Región 9 del IEEE.

El lanzamiento del proyecto ha sido celebrado mundialmente tanto dentro como fuera del IEEE y se están presentando muchos trabajos. Esta revista no pretende competir con las publicaciones tradicionales en inglés del IEEE. Asume que hay trabajos de primer nivel en español y portugués que no están siendo publicados y que pueden compartirse a través de esta nueva revista.

Estamos interesados en publicar las mejores contribuciones, de modo que prestigiados ingenieros están colaborando en comités editoriales especializados. Han trabajado para seleccionar los trabajos que se ofrecen a través de la revista.

Agradecemos públicamente al Dr. Antonio Jardini y a su equipo por el gran trabajo voluntario que está asegurando el éxito de esta iniciativa.

Le invitamos a continuar enviando sus contribuciones. La Revista ha sido incluida en IEEEXplore http://ieeexplore.ieee.org/ ISSN: 1548-0992

IEEE LATIN AMERICA TRANSACTIONS, VOL. 17, NO. 4, APRIL 2019 607

Modelling the 3-Coloring of Serial-Parallel Graphs Via Incremental Satisfiability

C. López-Ramírez, and G. De Ita

***Abstract*—A novel algorithm is presented for the 3-coloring on parallel-serial graphs. Our proposal is based on the logical specifications (using conjunctive normal forms) of the constraints for a valid 3-coloring of a serial-parallel graph, and afterward, to apply incremental satisfiability in order to build efficiently, a valid 3-coloring. Our proposal builds an initial satisfiable conjunctive formula φ, where SAT(φ) requests an exponential time. However, when new clauses are added to φ, then the colors of the vertices of each serial-parallel component of the input graph are determined in automatic and incremental way. Our procedure is deterministic and it finds a valid 3-coloring. It has a linear-time complexity on the size of the graph. We also show the correctness of our algorithm.**

***Index Terms*—SAT Problem, Incremental Satisfiability, Serial-Parallel Graphs, 3-Coloring.**

I. Introducción

Uno de los problemas fundamentales en el razonamiento automático es el problema de Satisfactibilidad (SAT), que revisa si una fórmula lógica F es (o no) satisfactible. SAT es también una tarea relevante en otros problemas como: estimar el grado de creencia, revisar o actualizar creencias, en la explicación abductiva, en el diagnóstico lógico y en otros procesos propios de la Inteligencia Artificial (IA) [1].

SAT es un problema teórico importante, y fue el primer problema reconocido en la clase de complejidad NP-Completo. SAT continúa siendo un problema fundamental para aclarar la frontera entre problemas de las clases de complejidad P y NP-Completo. En particular, el caso 2-SAT, que determina la satisfactibilidad de una dos Forma Normal Conjuntiva (2-FNC), es un caso importante del problema SAT por resolverse en tiempo polinomial.

A pesar de la dificultad teórica del problema SAT, los procedimientos actuales de decisión, conocidos como solucionadores SAT, han sido sorprendentemente eficientes.

Los solucionadores SAT se han usado en diversas aplicaciones industriales. Tales aplicaciones rara vez se limitan a resolver sólo un problema de decisión, ya que en general, una sola aplicación requiere que se resuelva una secuencia de problemas relacionados, por lo que manejan una secuencia de problemas relacionados como una instancia del problema de satisfactibilidad incremental (ISAT).

Cristina López-Ramírez - estudiante del doctorado LKE de la Facultad de Cs. de la Computación - BUAP. cristyna2001@hotmail.com.

Guillermo De Ita Luna - profesor de la Facultad de Cs. de la Computación, Universidad Autónoma de Puebla. deitaluna63@gmail.com.

Las mejoras en el rendimiento de los algoritmos para ISAT han resultado ser una característica crucial para los solucionadores SAT [2].

El problema ISAT considera como entrada una secuencia de cláusulas: F_0, F_1,…, F_n, empezando con una fórmula inicial satisfactible F_0. Cada F_i resulta de un cambio en la fórmula anterior F_{i-1} impuesto por el 'mundo exterior'.

Aunque el cambio puede ser una restricción (añadir cláusulas) o una relajación (eliminar cláusulas), nos centraremos en el caso de restricción, al considerar solo adiciones de nuevas cláusulas. El proceso de agregar cláusulas se termina cuando se llega a la insatisfactibilidad, o no hay más cláusulas que añadir.

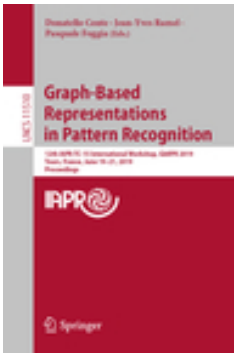
Una estrategia que se aplica al diseñar algoritmos para ISAT es preservar las estructuras computacionales formadas cuando se procesaron fórmulas anteriores, lo que permite por ejemplo, reconocer subfórmulas comunes que van apareciendo en las fórmulas de entrada, lo que permite la reutilización de la misma información a través de la secuencia de fórmulas.

Diferentes métodos se han aplicado para resolver ISAT, entre ellos, variaciones del procedimiento de ramificación y límites, denotado como Métodos IDPL, que generalmente se basan en el método clásico de Davis-Putnam-Loveland (DPL)[3]. En un procedimiento IDPL, al agregar nuevas cláusulas, se trata de mantener el árbol de búsqueda generado previamente. Se ha mostrado que IDPL se ejecuta más rápido que DPL para un gran conjunto de problemas SAT.

Whittemore et al. [4] definió ISAT como la resolución de cada subfórmula dentro de una secuencia finita de fórmulas. Wieringa [2] presentó una variedad de métodos secuenciales y paralelos para resolver ISAT, así como algunas de sus aplicaciones. Nadel [5] presenta una variación a ISAT bajo la hipótesis de cláusulas que fueron modificadas por variables de decisión; y de todas las cláusulas inferidas que dependen de algunos de los supuestos que incluyen su negación.

ISAT es de interés a una gran variedad de aplicaciones que necesitan ser procesadas en un entorno evolutivo. Este es el caso de aplicaciones como: planificación y programación reactiva, optimización combinatoria dinámica, revisión de fallas en circuitos combinatorios, satisfacción de restricciones dinámicas y aprendizaje en entornos dinámicos [6].

Por otro lado, el coloreo de grafos es quizás, uno de los problemas combinatorios más populares en la teoría de grafos [7]. En particular, la 3-colorabilidad sobre grafos planos es un conocido problema NP-completo. Una línea de investigación al considerar instancias de grafos planos es hallar cotas superiores en el número de colores a emplear.



Graph-Based Representations in Pattern Recognition

Conte, D. (et al.) (Eds.) (2019)

Bibliographic Information

Bibliographic Information

Bibliographic Information

Book Title

Pattern Recognition

Book Subtitle

11th Mexican Conference, MCPR 2019, Querétaro, Mexico, June 26–29, 2019, Proceedings

Editors

Jesús Ariel Carrasco-Ochoa

José Francisco Martínez-Trinidad

José Arturo Olvera-López

Joaquin Salas

Series Title

Image Processing, Computer Vision, Pattern Recognition, and Graphics

Series Volume

11524

Copyright

2019

Publisher

Springer International Publishing

Copyright Holder

Springer Nature Switzerland AG

eBook ISBN

978-3-030-21077-9

DOI
10.1007/978-3-030-21077-9
Softcover ISBN
978-3-030-21076-2
Edition Number
1
Number of Pages
XV, 444
Number of Illustrations
66 b/w illustrations, 96 illustrations in colour
Topics
Pattern Recognition

Recognizing 3-colorable Basic Patterns on Planar Graphs

Guillermo De Ita Luna(✉) and Cristina López-Ramírez

Fac. Cs. de la Computación, Universidad Autónoma de Puebla, Puebla, Mexico
deita@cs.buap.mx, cristyna2001@hotmail.com

Abstract. We recognize the wheel graphs with different kinds of centers or axle faces as the basic pattern forming a planar graph. We focus on the analysis of the vertex-coloring for these graphic patterns, and identify cases for the 3 or 4 colorability of the wheels. We also consider different compositions among wheels and analyze its colorability process.

If a valid 3-coloring exists for the union of wheels G, then our proposal determines the constraints that a set of symbolic variables must hold. These constraints are expressed by a conjunctive normal form F_G. We show that the satisfiability of F_G implies the existence of a valid 3-coloring for G. Otherwise, it is necessary to use 4 colors in order to properly color G. The revision of the satisfiability of F_G can be done in polynomial time by applying unit resolution and general properties from equalities and inequalities between symbolic variables.

AQ1

AQ2

Keywords: Wheel graphs · Polyhedral wheel graphs · Planar graphs · Vertex coloring

1 Introduction

By a proper coloring (o just a coloring) of a graph G, we refer to an assignment of colors (elements of a set) to the vertices of G, one color to each vertex, such that adjacent vertices are colored differently. The smallest number of colors in any coloring of G is called the chromatic number of G and is denoted by $\chi(G)$. When it is possible to color G from a set of k colors, then G is said to be k-colorable, while such coloring is called a k-coloring. If $\chi(G) = k$, then G is said to be k-chromatic, and every k-coloring is a minimum coloring of G.

The computation of the chromatic number $\chi(G)$ is polynomial computable if G is k-colorable with $k \leq 2$, but in other case the problem becomes NP-complete [4]. As a consequence, there are many unanswered questions related to the colouring of a graph.

Graph vertex colouring problem is an active field of research with many interesting subproblems and applications in areas like frequency allocation,planning, computer vision,scheduling, image processing, etc [2,7]. In this context, planar graphs play an important role in the graph theory area and complexity

J. A. Carrasco-Ochoa et al. (Eds.): MCPR 2019, LNCS 11524, pp. 1–11, 2019.
https://doi.org/10.1007/978-3-030-21077-9_31

Apéndice B

Ejemplos de la aplicación de las propuestas algorítmicas

B.1. Ejemplo de la construcción de un 3 coloreo sobre un arreglo poligonal

Consideramos que se aplica el algoritmo del 3-coloreo sobre arreglos poligonales sobre el grafo de la Figura B.1, obteniéndose el siguiente conjunto de fronds:
F_r ={{14,13},{12,13},{7,8},{18},{3,11}}.
Se ordenan los vértices en F_r formándose caminos maximales:13-18-12-13-14-11-3,7-8.
El primer paso del algoritmo es eliminar cualquier vértice de grado como máximo dos y después se van coloreando los vértices involucrados en los polígonos que tienen aristas en los caminos maximales de F_r. La Figura B.2 ilustra un 3-coloreo. Después se regresan los vértices de grado máximo 2, y se aplica el 2-coloreo asignándoles un color diferente al de sus vecinos (ver Figura B.3).
Se procesa cada camino maximal en F_r, para ir formando cláusulas unitarias que determinan un color para cada uno de los vértices del camino.
$\{13, 18\} \rightarrow (X_{13,1}), (X_{18,2}) \rightarrow (\neg X_{12,1}), (\neg X_{12,2}), (\neg X_{14,1}), (\neg X_{15,1}) \rightarrow (X_{12,3}) \rightarrow (\neg X_{11,3})$.
$\{18, 12\} \rightarrow \emptyset$
$\{12, 13\} \rightarrow \emptyset$
$\{13, 14\} \rightarrow (X_{14,2}) \rightarrow (\neg X_{11,2}), (\neg X_{15,2}) \rightarrow (X_{11,1}), (X_{15,3}) \rightarrow (\neg X_{3,1})$.
$\{14, 11\} \rightarrow \emptyset$
$\{7, 8\} \rightarrow (X_{7,1}), (X_{8,2}) \rightarrow (\neg X_{6,1}), (\neg X_{6,2}) \rightarrow (X_{6,3}), (X_{9,3})$

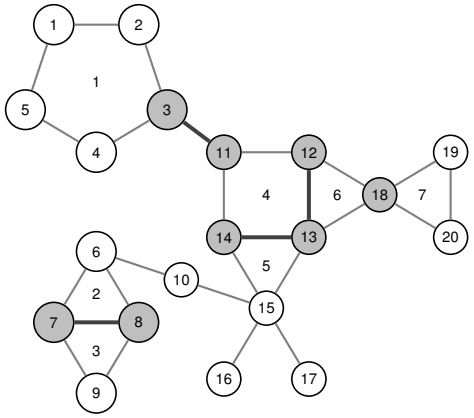


Figura B.1: Los elementos maximales de Fr

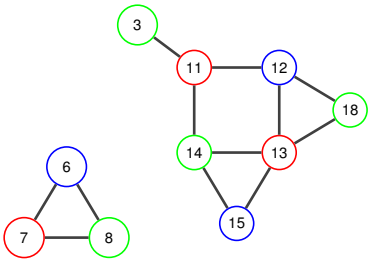


Figura B.2: Coloreo F_r.

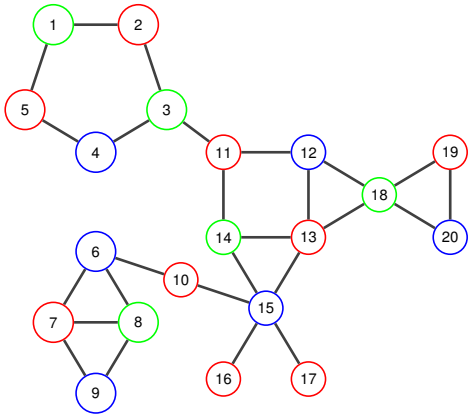


Figura B.3: 3-coloreo final

B.2. Ejemplo de la construcción de un 3 coloreo sobre un Grafo outerplanar

Ahora mostramos la aplicación de nuestra propuesta algorítmica para el 3-coloreo sobre grafos outerplanars.

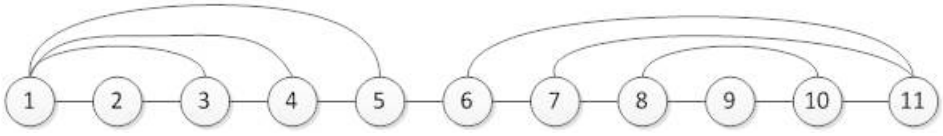


Figura B.4: Grafo outerplanar.

Representación poligonal del grafo

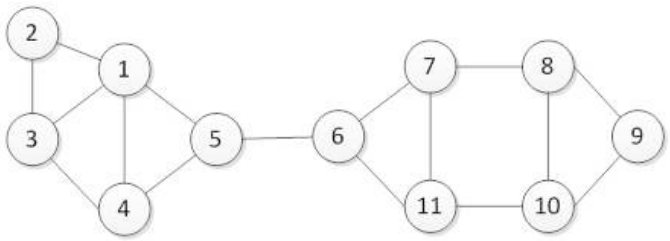


Figura B.5: Grafo poligonal.

Se quitan las partes acíclicas y los nodos con grado menor o igual 2.

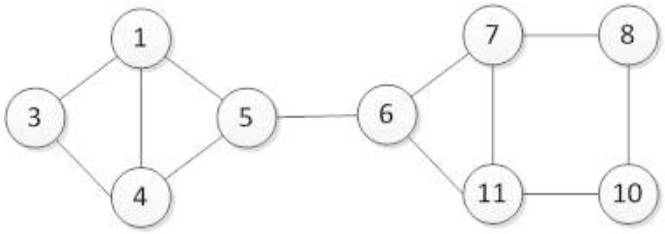


Figura B.6: Grafo sin vértices de grado 2.

Se forma una fórmula F que contenga a todas las cláusulas que modelan un 3-coloreo. Primero se consideran las cláusulas binarias que indican que un vértice no debe tener dos colores diferentes:
$(\neg X_{1,1} \vee \neg X_{1,2}) \wedge (\neg X_{1,1} \vee \neg X_{1,3}) \wedge (\neg X_{1,2} \vee \neg X_{1,3})$, $(\neg X_{3,1} \vee \neg X_{3,2}) \wedge (\neg X_{3,1} \vee \neg X_{3,3}) \wedge (\neg X_{3,2} \vee \neg X_{3,3})$, $(\neg X_{4,1} \vee \neg X_{4,2}) \wedge (\neg X_{4,1} \vee \neg X_{4,3}) \wedge (\neg X_{4,2} \vee \neg X_{4,3})$, $(\neg X_{5,1} \vee \neg X_{5,2}) \wedge (\neg X_{5,1} \vee \neg X_{5,3}) \wedge (\neg X_{5,2} \vee \neg X_{5,3}) \ldots$
Posteriormente, las cláusulas binarias que restringen que una arista tenga mismo color en sus vértices finales:
$(\neg X_{1,1} \vee \neg X_{3,1}) \wedge (\neg X_{1,1} \vee \neg X_{4,1}) \wedge (\neg X_{1,1} \vee \neg X_{5,1})$, $(\neg X_{1,2} \vee \neg X_{3,2}) \wedge (\neg X_{1,2} \vee \neg X_{4,2}) \wedge (\neg X_{1,2} \vee \neg X_{5,2})$, $(\neg X_{1,3} \vee \neg X_{3,3}) \wedge (\neg X_{1,3} \vee \neg X_{4,3}) \wedge (\neg X_{1,3} \vee \neg X_{5,3})$, $(\neg X_{3,1} \vee \neg X_{1,1}) \wedge (\neg X_{3,1} \vee \neg X_{4,1})$, $(\neg X_{3,2} \vee \neg X_{1,2}) \wedge (\neg X_{3,2} \vee \neg X_{4,2})$, $(\neg X_{3,3} \vee \neg X_{1,3}) \wedge (\neg X_{3,3} \vee \neg X_{4,3})$ $\ldots$
Por último, se codifican las cláusulas ternarias que indican que todo vértice debe tener un color:
$(X_{1,1} \vee X_{1,2} \vee X_{1,3}) \wedge (X_{3,1} \vee X_{3,2} \vee X_{3,3}) \wedge (X_{4,1} \vee X_{4,2} \vee X_{4,3}) \wedge (X_{5,1} \vee X_{5,2} \vee X_{5,3}) \ldots$
Lista de fronds $F_r = \{(7, 11), (5, 6), (1, 4)\}$

Se van coloreando los nodos siguiendo los fronds en las figuras.

$UP((X_{7,1}), F).\, UP((X_{11,2}), F)$

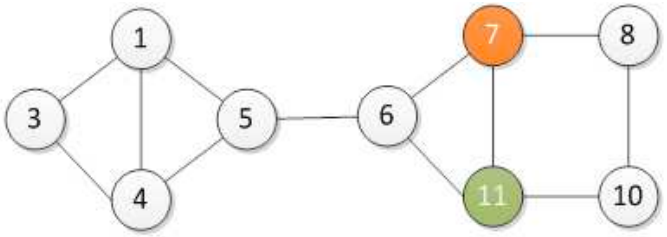


Figura B.7: Coloreo de vértices en el frond {7,11}.

Por cláusula ternaria $\neg X_{6,1}, \neg X_{6,2} \Rightarrow X_{6,3}$

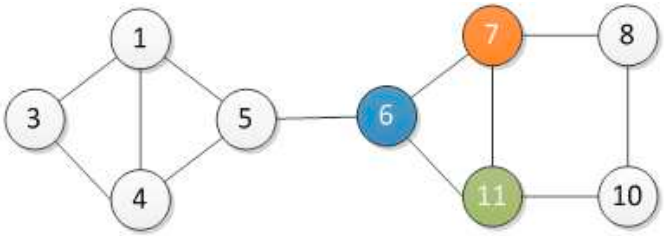


Figura B.8: Coloreo de vértice 6.

$UP((X_{5,1}), F).\, UP((X_{6,3}), F)$

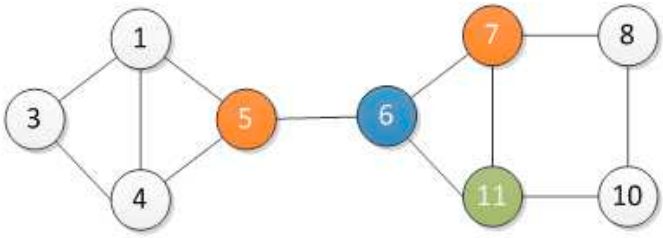


Figura B.9: Coloreo de vértices del siguiente frond.

$UP((X_{1,2}), F).\ UP((X_{4,3}), F)$

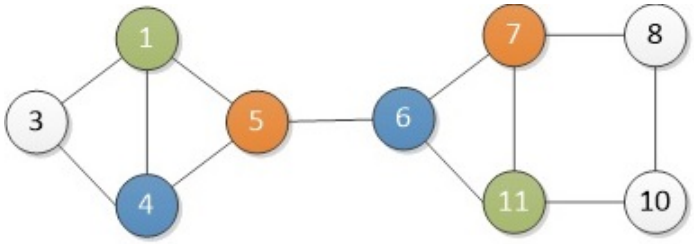


Figura B.10: Coloreo de vértices del siguiente frond.

Por cláusula ternaria $\neg X_{3,2}$, $\neg X_{3,3} \Rightarrow X_{3,1}$

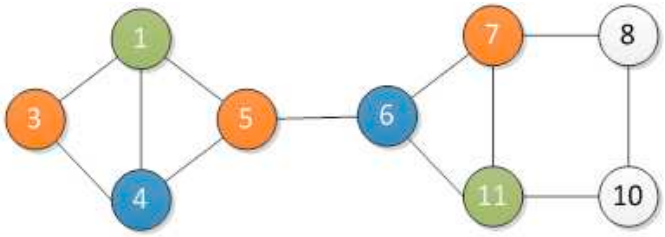


Figura B.11: Coloreo de vértice del último frond.

Se asigna color a los nodos restantes $X_{8,2}$, $X_{10,1}$

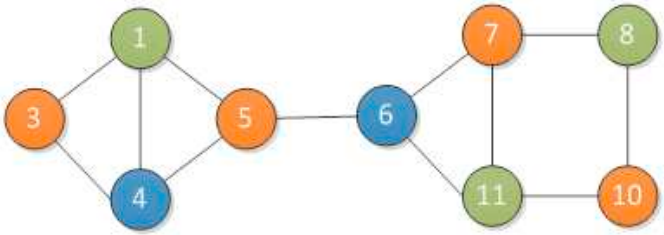


Figura B.12: Coloreo de los vértices restantes.

Se regresan los vértices eliminados y se les asigna color.

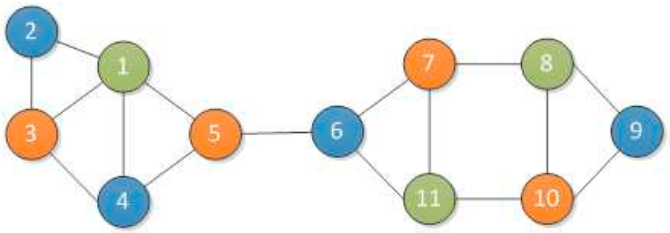


Figura B.13: 3-coloreo final del grafo poligonal.

B.3. Ejemplo para construir un 3 coloreo sobre un grafo serial paralelo

Ilustremos la aplicación de nuestra propuesta algorítmica para el 3-coloreo de un grafo serial-paralelo.
Consideremos el grafo serial-paralelo G de la Figura B.14.

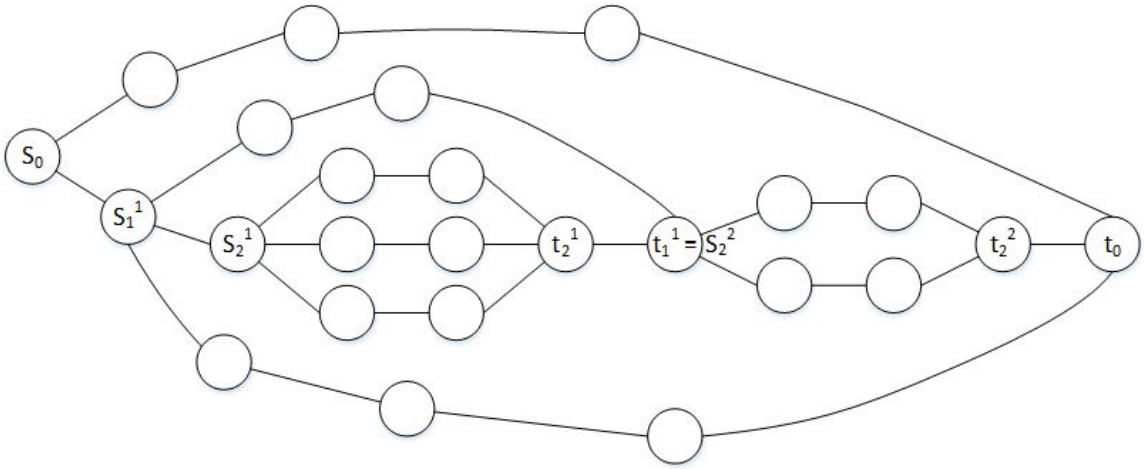


Figura B.14: Grafo serial-paralelo G.

Paso inicial: Reconocer los pares (s_i, t_i), removiendo los vértices de grado < 3, ya que son 2-coloreables como se ve en Figura B.15.

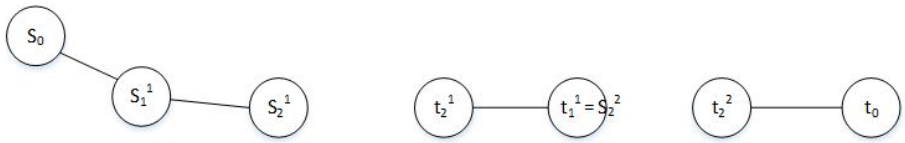


Figura B.15: Subgrafo con fuentes-objetos del grafo G.

Se empieza asignando color $f(s_0) = 3$, el cual se representa con el color azul, después se continúa asignando color 3 a cada componente paralela, como se muestra en la Figura B.16.

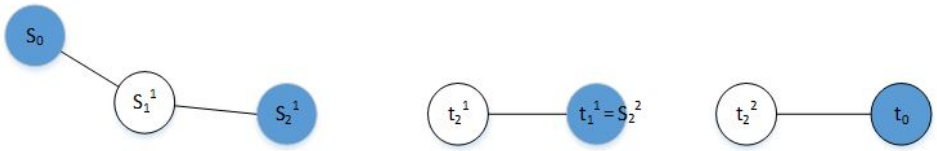


Figura B.16: Primera parte coloreada del grafo G.

Se colorean ahora los vértices de grado < 3, ya que son 2-coloreables como se ve en la Figura B.17.

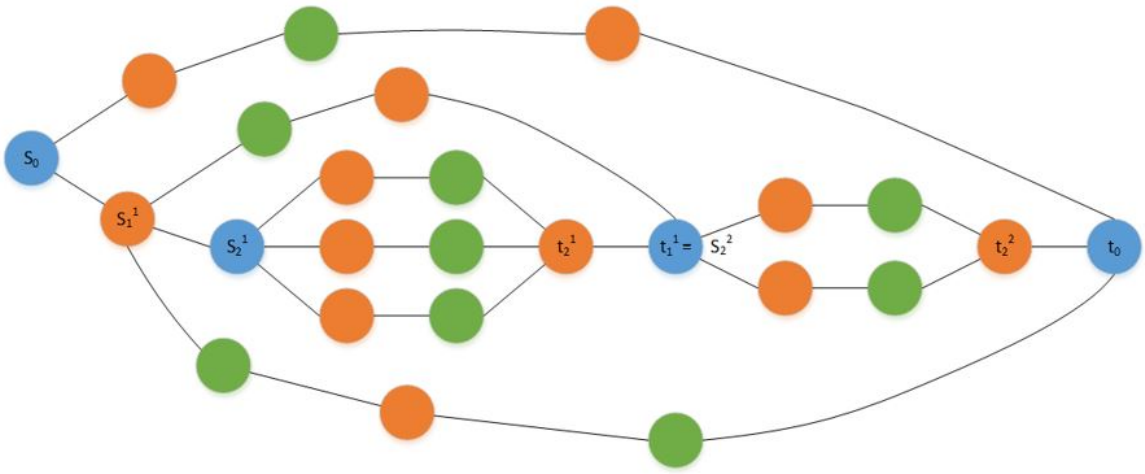


Figura B.17: Grafo G con un 3-coloreo.

Ejemplo 2. Se reconoce toda pareja (s_i, t_i) de las componentes serial-paralelo del grafo de entrada (Figura B.18). Se remueven los vértices de grado <3 y se asigna el color 3 a la primera fuente (s_1), como se muestra en la Figura B.19.

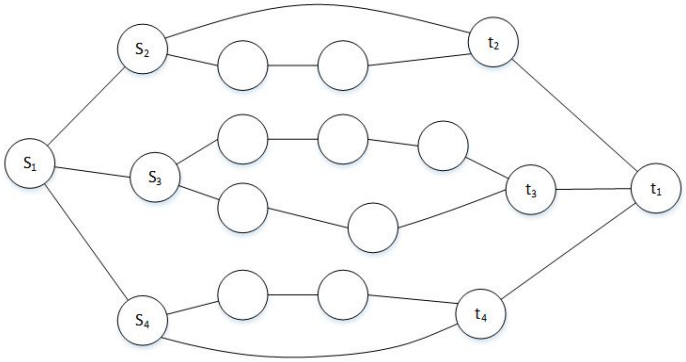


Figura B.18: Grafo serial paralelo de entrada G.

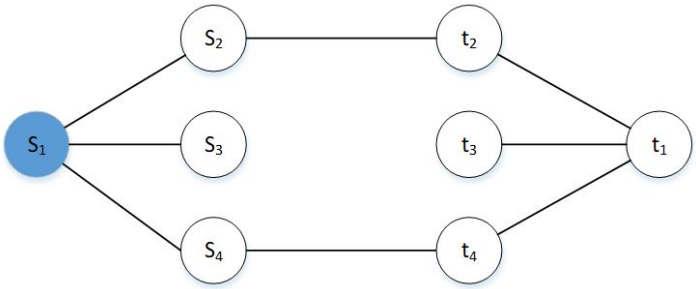


Figura B.19: G con primera fuente coloreada y vértices removidos.

Cómo ningún vértice fuente de las demás dos componentes pueden tomar color 3, entonces los vértices objetivo tomarán el color 3. Cuando se recuperan los vértices de grado 2, se aplica un 2-coloreo sobre estos vértices y G queda coloreado como se ve en la Figura B.20.

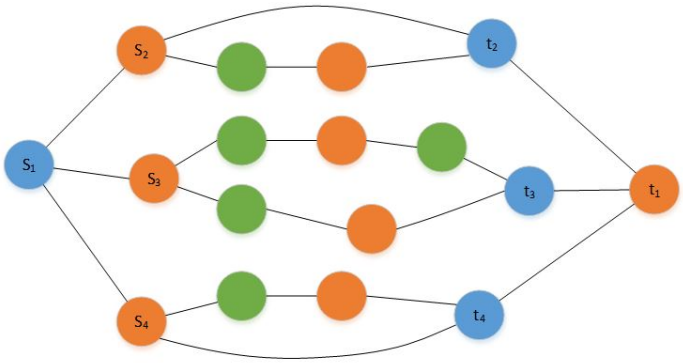


Figura B.20: Un 3-coloreo de G.

B.4. Algoritmo para determinar entre la 3 o 4 coloreabilidad de grafos planares

Sea F_G la fórmula lógica formada por las restricciones (I - IV), que son inducidas por las cadenas triangulares utilizadas para etiquetar las secuencias triangulares en un grafo planar. Cuando F_G es insatisfactible, entonces es necesario identificar las variables que tienen que estar asociadas con el cuarto color. Para esta tarea, se selecciona una variable simbólica x involucrada en un par de cláusulas unitarias contradictorias en F_G. x debe estar asociado solo con vértices libres; de lo contrario, se selecciona otra variable simbólica de las cláusulas contradictorias de F_G.
Cuando la variable x mantiene ambas condiciones, entonces los vértices etiquetados con x tienen asignado el color 1, se eliminan de G y se marcan sus vecinos. De esta manera, se forma una nueva fórmula lógica por las restricciones restantes del grafo actual. Este proceso de formación de fórmulas lógicas, verificando su satisfactibilidad, y asignando color 1 a sus vértices libres se itera hasta obtener una fórmula satisfactible.
Por ejemplo, consideremos el grafo de la Figura B.22. en este caso, la cadena triangular $t_1 =< x_1, y_1, z_1 >$ se usa para etiquetar todas las caras triangulares, hasta que no se puedan etiquetar más vértices. Como t_1 no es suficiente para cubrir todas las caras triangulares, entonces una nueva cadena triangular $t_2 =< x_2, y_2, z_2 >$ se usa para etiquetar los vértices de las caras triangulares restantes.
Para este grafo la igualdad $(y_2 = y_1)$ se forma. Mientras tanto, las restricciones de tipo II determinan la desigualdad $(y_1 \neq y_2)$. Así F_G es insatisfactible, entonces G no es 3-coloreable; por lo tanto es necesario usar un cuarto color para colorear adecuadamente la secuencia de caras triangulares.
Como la variable y_1 aparece en un par de clásulas contradictorias, y como todo vértice etiquetado con y_1 es libre, entonces $c(y_1) = 1$.
Esto también significa que todos los vértices etiquetados por y_1 se eliminan de G y sus vecinos están marcados.
En nuestro ejemplo, el grafo que resulta de eliminar vértices etiquetados con y_1 mantendrá una sola cara triangular etiquetada por la cadena: $y_2 - z_2 - x_2$ y cuyos vértices asociados no son libres. Entonces, se necesitan tres colores diferentes a 1 para colorear la cara triangular y cualquier camino acíclico. Por lo tanto, el grafo es 4-coloreable.

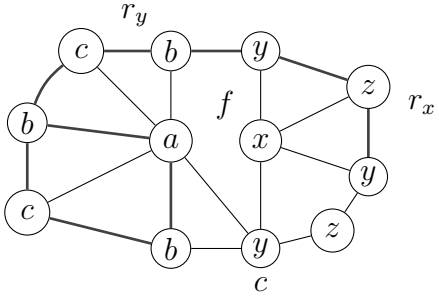


Figura B.21: Ruedas 3-colorables

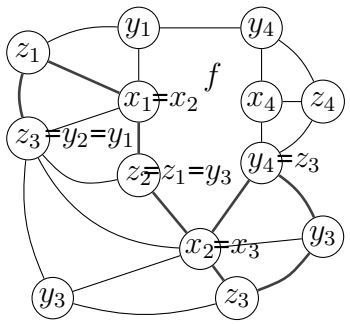


Figura B.22: Ruedas 4-coloreables

Se utilizan las cadenas triangulares para determinar entre la 3 o 4 colorabilidad del grafo actual. Para ilustrar esta propuesta, consideremos como instancia de prueba el grafo G usado por Heawood en su demostración de que las cadenas Kempe no es un método completo para 4CT. Tal grafo fue ilustrado en la Figura B.23 y su grafo de caras internas en la Figura B.24.

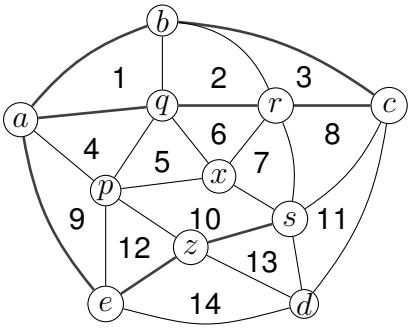


Figura B.23: Grafo G con caras

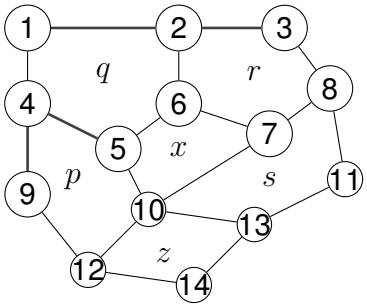


Figura B.24: G con caras-internas G_f

G tiene dos ruedas clásicas adyacentes centradas en q y r. Supongamos que $c(q) = 1$, ya que de hecho, ambas ruedas son simétricas, formándose el grafo G' de la Figura B.25. Como ya no hay más ruedas clásicas, aplicamos cadenas triangulares en G'.
Sea $t_1 =< x_1, y_1, z_1 >$ una cadena triangular que forma los siguientes grupos para $V(G) : x_1 \rightarrow \{e, s, b\}, y_1 \rightarrow \{p, d, r\}, yz_1 \rightarrow \{a, z, x, c\}$.

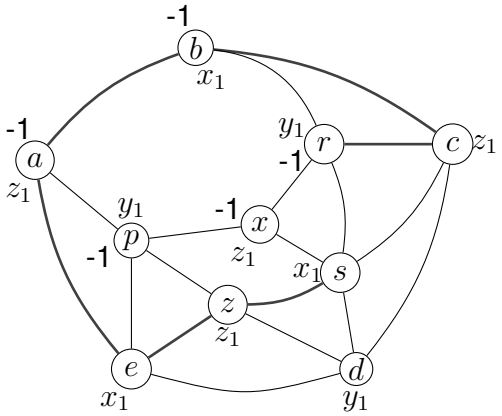


Figura B.25: The graph $G' = (V(G) - \{q\})$

Las restricciones I-IV forman un sistema satisfactible. Por lo tanto, F_G es satisfactible. Sin embargo, en cada uno de los tres grupos formados por las variables: x_1, y_1, z_1 hay al menos un vértice no libre, entonces no se puede usar el color 1 en ninguno de estos grupos. De donde, sólo los colores 2,3 y 4 puede asignarse a la cadena t_1 un 4 coloreo para G.

Referencias

Aho, V., Hopcroft, J. E., & Ullman, J. D. (1988). Estructura de datos y algoritmos. In *Addison-Wesley Iberoamericana*.

Alban, G. & Anbu, A. (2008). Incremental diagnosis of des by satisfiability. *Artificial Intelligence*, 787–788.

Alexander, N., Vadim, R., & Ofer, S. (2016). Ultimately incremental sat. *Lógica/ Lenguajes, Algoritmos y Nuevos Métodos de Razonamiento.*, *8561*, 206–218.

Andrzej, P. & Maciej, S. (1986). Efficient vertex- and edge-coloring of outerplanar graphs. *SIAM Journal on Algebraic and Discrete Methods.*, *7*, 131–136.

Ansótegui, C. & Manyà, F. (2003). An introduction to satisfiability algorithms. *Iberoamericana de Inteligencia Artificial*, *20*, 43–56.

Appel, K. & Haken, W. (1977). Every planar map is four colourable, part i: discharging. *Illinois J. Mathematics*, 429–490.

Armin, B., Keijo, H., Tommi, J., Timo, L., & Viktor, S. (2006). Linear encodings of bounded ltl model checking. *ACM (Association for Computing Machinery) Logical Methods in Computer Science*, *2*, 1–64.

Audemard, G., Jabbour, S., & Lakhdar, S. (2007). Using graph-based representation to derive hard sat instances. *Journals of Algorithms.*

Auderman, G. & Laurent, S. (2003). Predicting learnt clauses quality in modern sat solvers. *IJCAI*, 399–404.

Bart, S., Hector, L., & David, M. (1992). A new method for solving hard satisfiability problems. *AAAI-92*, 440–446.

Beckert, B., Hahnle, R., & Manyá, F. (2000). The 2-sat problem of regular signed cnf formulas. *IEEE International Symposium on*, *17*, 59–80.

Bekos, M. A., Kaufmann, M., & Raftopoulou, C. N. (2017). On optimal 2- and 3-planar graphs. In *Symposium on Computational Geometry*.

Belov, A., Janota, M., & Marques-S, J. (2013). Formula preprocessing in mus extraction. *Tools and Algorithms for the Construction and Analysis of Systems. (TACAS)*, *7795*, 108–123.

Benedetti, M. & Bernardini, S. (2011). Incremental compilation-to-sat procedures. In Hoos H.H., M. D. (Ed.), *Theory and Applications of Satisfiability Testing*, volume 3542, (pp. 46–58).

Boyer, J. & Myrvold, W. (2004). On the cutting edge: Simplied o(n) planarity by edge addition. *Jour. of Graph Algorithms and Applications*, *8*, 241–273.

Bubeck, U. & Kleine, H. B. (2009). A new 3-cnf transformation by parallel-serial graphs. *Inf. Process. Lett.*, *109*, 376–379.

Byskov, J. (2005). *Exact Algorithms for graph colouring and exact satisability*. PhD thesis, Denmark.

Chandrasekar, K. & Michael, S. H. (2007). Integration of learning techniques into incremental satisfiability for efficient path-delay fault test generation. *Electronic Design Automation (EDA)*, *2*, 1002–1007.

Chartrand, G. & Harary, F. (1967). Planar permutation graphs. *Annales de l'I.H.P. Probabilités et statistiques*, *3*(4), 433 – 438.

Chávez, O. & Pozos, P. (2014). Modelando la toma de decisiones mediante fusión de creencias. *Electrónica de Comunicaciones y Trabajos de ASEPUMA*, *15*, 141–158.

Cheeseman, P., Kanefsky, B., & Taylor, W. (1991). Where the really hard problems are. *Proceedings of the 12th international joint conference on Artificial intelligence*, *1*, 331–337.

Cook, S. (1971). The complexity of theorem proving procedures. *Proceedings of the 3rd. ACM Symposium on Theory of Computing*, 151–158.

Davis, M., Logemann, G., & Loveland, D. W. (1962). A machine program for theorem-proving. *Commun. ACM*, *5*, 394–397.

Davis, M. & Putman, H. A. (1960). A computing procedure for quantification theory. *J. ACM*, *7*, 201–215.

De Ita, L. G., Bautista, C., & Altamirano, L. (2011). Solving 3-colouring via 2sat. In Martínez-Trinidad, J., Carrasco-Ochoa, J., Ben-Youssef, B. C., & Hancock, E. (Eds.), *Pattern Recognition. MCPR 2011*, volume 6718.

De Ita, L. G. & Castillo, J. (2013). Recognizing 3-colorings cycle-patterns on graphs. *Pattern Recognition Letters*, *34*, 433–438.

De Ita, L. G., Cristina, L. R., & Meliza, G. C. (2018). Modelling 3-coloring of outerplanar graphs via incremental satisfiability. *Electronic Notes in Discrete Mathematics*, *69*, 101 – 108.

De Ita, L. G., López, B. P., & Contreras, G. M. (2007). Efficient counting of models for boolean formulas represented by embedded cycles. In *Latin-American Workshop on Non-Monotonic Reasoning, Proceedings ofthe LANMR07 Workshop, Benemérita Universidad Autónoma de Puebla, Puebla, Pue., México, 17th - 19th September 2007*.

De-Ita, L. G. & López-Ramírez, C. (2019). Recognizing 3-colorable basic patterns on planar graphs. In *MCPR*.

De Ita, L. G., Marcial-Romero, J., & Hernández, S. J. A. (2016). The incremental satisfiability problem for a two conjunctive normal form. *Electr. Notes Theor. Comput. Sci.*, *328*, 31–45.

De Ita, L. G., Marcial-Romero, J., López, P., & Contreras, G. M. (2018). Linear-time algorithms for computing the merrield-simmons index on polygonal trees. *MATCH Comm. in Math. and in Comp. Chemistry*, *79*, 55–78.

Disch, S. & Scholl, C. (2007). Combinational equivalence checking using incremental sat solving, output ordering, and resets. *2007 Asia and South Pacific Design Automation Conference*, 938–943.

Drias, H. (1998). A monte carlo algorithm for the satisfiability problem. In Mira, J., del Pobil, A., & Ali, M. (Eds.), *Methodology and Tools in Knowledge-Based Systems. IEA/AIE 1998.*, volume 1415, (pp. 159–168).

Dvorák, Z., Krá, l. D., & Thomas, R. (2009). Three-coloring triangle-free graphs on surfaces. *Proc. of 20th ACM-SIAM Symp. on Discrete Algorithms*, 120–129.

Fermé, E. (2007). Revisión de creencias. *Iberoamericana de Inteligencia Artificial*, *34*, 17–39.

Florian, L. & Marques-Silva, J. (2008). Improvements to hybrid incremental sat algorithms. In Hans, K. B. & Xishun, Z. (Eds.), *Theory and Applications of Satisfiability Testing SAT 2008*, volume 4996, (pp. 168–181).

Galliere, J. H. (1988). *Logic for Computer Science: Foundation of Automatic Theorem Proving*. Wiley.

Garey, M. R. & David, S. J. (1978). Computers and intractability: A guide to the theory of np-completeness.

Gent, I. P. & Walsh, T. (1999). Beyond np: the qsat phase transition. *American Association for Artificial Intelligence (AAAI).*, 648–653.

Grötzsch, H. (1959). Ein dreifarbensatz für dreikreisfreie netze auf der kugel. *Wiss. Z. Martin Luther Univ. Halle-Wittenberg, Math. Nat. Reihe*, *8*, 109–120.

Hartmanis, J. & Stearns, R. E. (1965). On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, *117*, 285–306.

Hooker, J. N. (1993). Solving the incremental satisfiability problem. *J. Log. Program.*, *15*, 177–186.

HoonSang, J. & Fabio, S. (2005). An incremental algorithm to check satisfiability for bounded model checking. *Electr. Notes Theor. Comput. Sci.*, *119*, 51–65.

Jian, G., Jianan, W., & Minghao, Y. (2014). Experimental analyses on phase transitions in compiling satisfiability problems. *Science China Information Sciences*, *58*, 1–11.

João, A. & Monteiro, J. C. (2017). Analysis of short-circuit conditions in logic circuits. *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2017*, 824–829.

Joonyoung, K. (2001). *Incremental Boolean satisfiability and its application to electronic design automation.* PhD thesis.

Kullmann, O. (1999). New methods for 3-sat decision and worst-case analysis. *Theor. Comput. Sci.*, *223*, 1–72.

Larrabee, T. (1992). Test pattern generation using boolean satisfiability. *IEEE Trans. on CAD of Integrated Circuits and Systems*, *11*, 4–15.

Li, C. M. & Anbulagan, A. (1997). Heuristics based on unit propagation for satisfiability problems. *Artifical Intelligence*, *1*, 366–371.

López, R. C. & De Ita, L. G. (2019). Modelling the 3-coloring of serial-parallel graphs via incremental satisfiability. *IEEE Latin America Transactions*, *17*, 607–614.

López, R. C., De Ita, L. G., & Neri, A. (2018). Modelling 3-coloring of polygonal trees via incremental satisfiability. In Martínez-Trinidad, J., Carrasco-Ochoa, J., Olvera-López, J., & Sarkar, S. (Eds.), *Pattern Recognition. MCPR 2018*, volume 10880, (pp. 93–102).

Marques-Silva, J. & Sakallah, K. A. (1999). Grasp: A search algorithm for propositional satisfiability. *IEEE Trans. Computers*, *48*, 506–521.

McMillan, K. (2003). Interpolation and sat-based model checking. In Hunt, W. & Somenzi, F. (Eds.), *Computer Aided Verification*, volume 2725, (pp. 1–13).

Mertzios, G. B. & Spirakis, P. G. (2014). Algorithms and almost tight results for 3-colorability of small diameter graphs. *Algorithmica*, *74*, 385–414.

Mohamed-El, B. M. (2004). An evolutionary local search method for incremental satisfiability. In Buchberger, B. & Campbell, J. (Eds.), *Artificial Intelligence and Symbolic Computation*, volume 3249, (pp. 143–156).

Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., & Malik, S. (2001). Chaff: Engineering an efficient sat solver. In *Proceedings of the 38th Annual Design Automation Conference*, New York, NY, USA. ACM.

Mouhoub, M. & Sadaoui, S. (2007). Solving incremental satisfiability. *International Journal on Artificial Intelligence Tools*, *16*, 139–147.

Nadel, A., Ryvchin, V., & Strichman, O. (2012). Preprocessing in incremental sat. In Cimatti, A. & Sebastiani, R. (Eds.), *Theory and Applications of Satisfiability Testing – SAT 2012. SAT 2012.*, volume 7317, (pp. 256–269).

Niklas, E. & Niklas, S. (2003). Temporal induction by incremental sat solving. *Electr. Notes Theor. Comput. Sci.*, *89*, 543–560.

Oubiña, L. & Zucchello, R. (1984). A generalization of outerplanar graphs. *Discrete Mathematics*, *51*(3), 243 – 249.

Patrignani, M. (2004). Planarity testing and embedding. In *Handbook of Graph Drawing and Visualization*.

Prabhat, M. & Mingsong, C. (2009). Efficient techniques for directed test generation using incremental satisfiability. *22nd International Conference on VLSI Design*, 65–70.

Prakash, C. S. & Narendra, S. C. (2011). Polynomial 3-sat encoding for k-colorability of graph. *IJCA Special Issue on Evolution in Networks and Computer Communications*, *1*, 19–24.

Robertson, N., Sanders, D., Seymour, P., & Thomas, R. (1997). The four color theorem. *Journal of Combinatorial Theory, Series B*, *70*, 2–44.

Silvio, F., Giorgio, P., Maksim, S., Pierfrancesco, U., & Francesco, Z. (2017). Universality of the sat-unsat (jamming) threshold in non-convex continuous constraint satisfaction problems.

Stefan, S. (2008). Not so easy problems for tree decomposable graphs. *ArXiv*, *abs/1107.1177*.

Whitney, H. & Tutte, W. (1992). Kempe chains and the four colour problem. In Eells J., T. D. (Ed.), *Hassler Whitney Collected Papers*, volume 1, (pp. 185–225).

Whittemore, J., Joonyoung, K., & Karem, A. S. (2001). Satire: A new incremental satisfiability engine. *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*, 542–545.

Wieringa, S. (2011). On incremental satisfiability and bounded model checking. In *DIFTS@FMCAD*.

Wieringa, S., Niemenmaa, M., & Heljanko, K. (2009). Tarmo: A framework for parallelized bounded model checking. In *PDMC*.

Zhang, H. (1997). Sato: An eficient propositional prover. In McCune, W. (Ed.), *Automated Deduction—CADE-14*, volume 1249, (pp. 272–275).