UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO

FACULTAD DE INGENIERÍA
MAESTRÍA EN CIENCIAS DE LA INGENIERÍA

DISEÑO DE UN ALGORITMO PARA EL CONTEO DE MODELOS DE FÓRMULAS BOOLEANAS EN DOS FORMA NORMAL CONJUNTIVA

TESIS

QUE PARA OBTENER EL GRADO DE:
Maestro en Ciencias de la Ingeniería

PRESENTA:
Marco Antonio López Medina

TUTOR ACADÉMICO:
Dr. José Raymundo Marcial Romero

TUTORES ADJUNTOS:
Dra. Rosa María Valdovinos Rosas
Dr. Marcelo Romero Huertas

Toluca México, Octubre 2018

Resumen

El conteo de modelos sobre fórmulas booleanas es el problema conocido como #SAT, el cual forma parte de los problemas catalogados como #P-Completos.

#2SAT es una restricción a #SAT, donde las cláusulas que componen una fórmula contienen a lo más dos variables, uno de los métodos usados para resolver este problema es la *descomposición de la fórmula*, la cual puede realizarse por la asignación de valores de verdad a una variable o a una cláusula. La descomposición de una fórmula por variable requiere un procedimiento de selección para la variable o cláusula a eliminar, el método más utilizado es seleccionar la variable que tiene una mayor aparición en la fórmula, de esta forma se puede reducir el número de cláusulas dado el número de apariciones de la variable seleccionada.

Por otro lado, la descomposición de una fórmula por cláusula requiere un método para seleccionar la cláusula más viable a eliminar, este segundo método permite reducir la cantidad de procesos de descomposición de la fórmula con respecto a la descomposición por variable. Si se quieren eliminar dos variables por el primer procedimiento es necesario generar cuatro subfórmulas, sin embargo, con el segundo procedimiento es posible generar tres subfórmulas aprovechando la relación existente entre las dos variables que se encuentran dentro de la cláusula.

El objetivo de la descomposición de la fórmula es aplicar el método de forma iterativa hasta encontrar subfórmulas que puedan ser resueltas en tiempo polinomial. Una clase de fórmulas en las cuales existe un procedimiento polinomial para calcular #2SAT es aquella en la cual sus variables aparecen a lo más cuatro veces en la fórmula. En esta tesis se estudian clases de fórmulas cuyo conteo de modelos puede realizarse en tiempo polinomial, además se desarrollan métodos para resolverlas basados en la topología inducida por su gráfica signada. Las topologías para las cuales se presentan algoritmos polinomiales se conocen como cactus y outerplanar. Así mismo, demostramos experimentalmente que los algoritmos obtenidos son competitivos respecto a la herramienta más eficiente reportada en la literatura que es sharpSAT tanto en fórmulas cactus y outerplanar como en instancias generales.

Para cada uno de los algoritmos obtenidos se realizó un comparativo con sharpSAT, en cada caso se mejoró el desempeño de sharpSAT para los casos base presentados, se logró mejorar el tiempo de ejecución con respecto a sharpSAT para 2-FNC haciendo uso de los casos base de manera simultánea.

Agradecimientos

Agradezco a CONACYT por el apoyo brindado con No. 447783, sin ka cual habría sido difícil continuar con el desarrollo de la investigación, así como el apoyo brindado para una estancia nacional en la Benemérita Universidad Autónoma de Puebla durante la Convocatoria de Movilidad Nacional No. 291249.

A mis padres quienes a lo largo de mi vida han velado por mi bienestar y educación, y que confiaron en mis capacidades para enfrentar cualquier reto académico que se presentara.

A mis hermanos que creyeron en mi y me brindaron apoyo moral durante mi trayectoria académica.

A mis asesores que me permitieron llevar a cabo este proyecto de investigación, el Dr. José Raymundo Marcial Romero y el Dr. Guillermo De Ita Luna, gracias a su impulso e ideas se lograron obtener resultados satisfactorios en el transcurso del proyecto de investigación. Así como a los miembros de mi comité de tutores, la Dra. Rosa María Valdovinos Rosas y el Dr. Marcelo Romero Huertas.

A los miembros de mi comité de evaluaciones parciales de tesis en cada periodo académico, que me permitieron mejorar mis habilidades para dar a entender mi proyecto a personas ajenas al área.

Tabla de Contenido

Capítulo 1

Introducción

El problema $SAT(F)$, donde F es una fórmula booleana, consiste en decidir si F tiene un modelo, es decir, una asignación a las variables de F tal que al ser evaluada con respecto a la lógica booleana clásica devuelva un valor verdadero [14]. Si F esta en dos Forma Normal Conjuntiva (2-FNC) entonces $SAT(F)$ se puede resolver en tiempo polinomial, sin embargo, si F esta en $k-FNC, k > 2$, entonces $SAT(F)$ es un problema NP-Completo [8]. Por otro lado, existe el problema de conteo de modelos denotado como $\#SAT(F)$. Es decir $\#SAT(F)$ es un problema de conteo a diferencia de $SAT(F)$ que se puede considerar como un problema de decisión. $\#SAT(F)$ pertenece a la clase #P-Completo aún cuando F este en 2-FNC, este último denotado como #2SAT [2]. Aunque el problema #2SAT es #P-Completo, existen instancias que se pueden resolver en tiempo polinomial [3]. Por ejemplo, si la gráfica que representa la fórmula es acíclica, entonces el número de modelos se puede calcular en tiempo polinomial.

Actualmente, los algoritmos que se utilizan para resolver #2SAT para cualquier fórmula F en 2-FNC, descomponen F en subfórmulas hasta que se tienen casos denominados base en los que se puede contar en tiempo polinomial. Para el caso general el algoritmo con la menor complejidad en tiempo hasta el momento fue desarrollado por Wahlström [4], el cual esta dado por $O(1.2377^n)$ donde n representa el número de variables de la fórmula. El algoritmo de Wahlström utiliza el método de descomposición de variable cuyo criterio de elección está dado por el número de veces que aparece en la fórmula (sea la variable o su negación) y considera dos criterios de paro cuando $F = \emptyset$ o cuando $\emptyset \in F$.

Por otro lado, están las implementaciones para #SAT en donde el objetivo es buscar estrategias que permitan resolver el problema en un tiempo de cómputo razonable para instancias en donde el número de variables es considerable. Las herramientas que hasta la fecha se consideran las más eficientes son relsat [5] y sharpSAT [6]. La herramienta relsat esta basada en el algoritmo DPLL (Davis-Putnam algorithm) cuya ventaja es poder proce-

sar rápidamente fórmulas disjuntas, es decir cuyos conjuntos de variables no se intersectan. La herramienta sharpSAT es una mejora de relsat en donde se elige una literal heurísticamente, así mismo, utiliza descomposición de componentes y cache de componentes para agilizar el cálculo.

En esta tesis se desarrollan algoritmos para el conteo de modelos en fórmulas booleanas en 2-FNC utilizando la descomposición de la gráfica representativa de una fórmula en árbol-coárbol, además de distintos casos base, realizando un comparativo experimental con sharpSAT para determinar si se cumple con el objetivo de mejorar los tiempos de cómputo contra la herramienta más eficiente reportada en la literatura.

Organización de la tesis

Con base en los artículos 57, 59 y 60 del Reglamento de los Estudios Avanzados de la Universidad Autónoma del Estado de México, la presente tesis está desarrollada en la modalidad de Tesis por artículos especializados compuesta por los siguientes capítulos:

- Capítulo II. Protocolo de Tesis. Presenta una versión actualizada del protocolo de tesis registrado ante la Secretaría de Estudios Avanzados de la Universidad Autónoma del Estado de México.
- Capítulo III. Se presentan los artículos obtenidos durante la investigación.
  - Un Algoritmo de Tiempo Lineal para resolver #2SAT en Fórmulas Cactus. Consta de un artículo enviado al journal IEEE Transactions on Latin America que se encuentra en revisión.
  - Un método rápido y eficiente para #2SAT usando transformaciones en gráfica. Presenta un artículo enviado y aceptado en la conferencia: MICAI 2017. Publicado en Lecture Notes in Artificial Intelligence Vol 10632-10633.
  - Un algoritmo de tiempo lineal para el cómputo de #2SAT en fórmulas outerplanar en 2-FNC. Presenta un artículo enviado y aceptado en el MCPR 2018. Publicado en Lecture Notes in Computer Science Vol. 10880.
- Capítulo IV. Presenta las Conclusiones y trabajo futuro.

Capítulo 2

Protocolo de Investigación

En este capítulo se presenta el protocolo de tesis aprobado por el comité de tutores y registrado ante la coordinación del programa de estudios avanzados de la Facultad de Ingeniería.

2.1. Descripción del Proyecto

Este proyecto busca resolver el problema #2SAT utilizando la descomposición de la gráfica representativa de una fórmula en árbol-coárbol, además de distintos casos base que permitan encontrar una solución a cualquier instancia del problema, realizando un comparativo con la herramienta sharpSAT, para verificar de manera experimental que se mejora el tiempo de cómputo con respecto a esta herramienta que es la más eficiente reportada en la literatura.

2.2. Marco Teórico

En esta sección se presentan los conceptos necesarios utilizados para llevar a cabo el trabajo de investigación.

2.2.1. Lógica proposicional

Sintaxis. En esta lógica, el vocabulario es un conjunto de *variables* P . Una *fórmula* de la lógica proposicional sobre P se define como:

- Toda variable x_i de P es una fórmula.
- Si F y G son fórmulas, entonces $(F \vee G)$ y $(F \wedge G)$ son fórmulas.
- Si F es una fórmula, entonces $\bar{F}$ es una fórmula.

Interpretación. Una *interpretación* I sobre el vocabulario P es una función $I : P \to \{0, 1\}$, es decir, I es una función que, para cada variable asigna 1 (cierto) o 0 (falso).

Satisfacción. Sean I una interpretación y F una fórmula, ambas sobre el vocabulario P. La *evaluación* en I de F, denotada $eval_i(F)$, es una función que por cada fórmula da un valor 0 o 1. Se define $eval_i(F)$ para todos los casos posibles de F, usando *min, max* y -, que denotan respectivamente el mínimo, el máximo, y la resta (sobre números del conjunto {0,1}), donde , $min(0,0) = min(0,1) = min(1,0) = 0$ y $min(1,1) = 1, max(1,1) = max(0,1) = max(1,0) = 1$ y $max(0,0) = 0$:

- si F es un símbolo p de P entonces $eval_i(F) = I(p)$.
- $eval_i(F \vee G) = max(eval_i(F), eval_i(G))$
- $eval_i(F \wedge G) = min(eval_i(F), eval_i(G))$
  $eval_i(\bar{F}) = 1 - eval_i(F)$

Se define: si $eval_i(F) = 1$ entonces I satisface F, denotado $I \models F$. En este caso también se dice que F *es cierta en* I, o que I *es un modelo de* F.

Una asignación s es un conjunto de literales tomadas de una fórmula F, con la condición de que sólo una de las literales x_i o $\bar{x_i}$ pertenece a s. Puede también considerarse como un conjunto de literales no complementarias. Si $x_i \in s$, siendo s una asignación, entonces s convierte x_i en verdadero y $\bar{x_i}$ en falso. Por otro lado, si $\bar{x_i} \in s$ entonces s convierte a $\bar{x_i}$ en verdadero y x_i en falso. Considerando una cláusula c y una asignación s como un conjunto de literales, se dice que c se satisface por s sí y solo si $c \cap s \neq \emptyset$ y si para toda $x_i \in c, \bar{x_i} \in s$ entonces s falsifica a c. Sea F una fórmula booleana en FNC, se dice que F se satisface por la asignación s si cada cláusula en F se satisface por s. Por otro lado, se dice que F se contradice por s si al menos una cláusula de F se falsifica por s. Un modelo de F es una asignación para $v(F)$ tal que satisface F. Se denota como $SAT(F)$ al conjunto de modelos de la fórmula F.

2.2.2. Forma Normal Conjuntiva (FNC)

Una expresión lógica, la cual contiene variables x_i y los conectivos lógicos $\wedge, \vee$ y $\neg$ (AND, OR y NOT), es una fórmula. Se usa la notación $\bar{x_i}$ para denotar la negación $\bar{x_i}$, y el término literal para referirse a una x_i o a una $\bar{x_i}$. El que la expresión esté en *forma normal conjuntiva* significa que es una conjunción:

$$C_1 \wedge C_2 \wedge \ldots \wedge C_c$$

de subexpresiones C_i, cada una de las cuales es una disyunción de literales

$$(x_1 \vee x_3 \vee x_4) \wedge (\bar{x_1} \vee x_3) \wedge (\bar{x_1} \vee x_4 \vee \bar{x_2}) \wedge \bar{x_3} \wedge (x_2 \vee \bar{x_4}) \tag{2.1}$$

La ecuación 2.1 es una expresión en Forma Normal Conjuntiva con cinco *elementos conjuntivos*. En general una variable podría aparecer más de una vez en un elemento conjuntivo, y este último incluso podría duplicarse.

El problema de satisfacción-FNC (o FNC-Sat) es el siguiente: Dada una expresión en forma normal conjuntiva, ¿existe una asignación de verdad que la satisfaga?.

El problema de conteo de modelos, mejor conocido por $\#SAT$ consiste en calcular el número de modelos de una fórmula booleana dada, es decir, el número de asignaciones de verdad para las cuales la fórmula se evalúa como verdadera. #k-SAT es el problema de calcular el número de modelos de una fórmula en FNC donde las cláusulas tienen a lo más k variables.

En este caso el problema que se aborda directamente es $\#2SAT$ que es el conteo de modelos sobre una fórmula booleana en Forma Normal Conjuntiva donde cada cláusula contiene a lo más dos variables. El problema $\#2SAT$ pertenece a la clase #P completo [8], por tanto no existe algoritmo eficiente que resuelva el problema general en tiempo polinomial.

Las instancias en 2-Forma Normal Conjuntiva (2-FNC) pueden ser representadas mediante una gráfica como se verá en la sección 2.2.3.

Árbol de eliminación

Dada una fórmula F en k-FNC, $V(F)$ denota las variables que aparecen en F. Sea $b : V(F) \to \{0, 1\}$ una asignación parcial de valores de verdad a las variables en F. La fórmula F_b se obtiene de F aplicando las asignaciones de acuerdo a b.

Un árbol de eliminación para una k-FNC F se define como sigue, se propone un *orden de eliminación* $(y_1, ..., y_n)$ de las variables, donde $y_i = x_j, 1 \leq i \leq n, 1 \leq j \leq n$ como una correspondencia 1 a 1, es decir no existe $y_i = x_j$ y $y_k = x_j, 1 \leq k \leq n, k \neq i$. Todo nodo f del árbol corresponde a una fórmula booleana, la raíz del árbol (nivel 0) corresponde a F. Todo nodo f en los niveles $0 \leq i < n$ tiene dos hijos: Un hijo corresponde a $f_{y_i=0}$ y el otro a $f_{y_i=1}$. Entonces un camino de la raíz a una hoja corresponde a una asignación a cada una de las variables, y una fórmula en una hoja puede ser 0 o 1. $\#SAT(F)$ es el número de hojas marcadas con 1.

Este concepto de árbol de eliminación donde se elige un orden de eliminación para las variables puede ser aplicado a las cláusulas, dada una fórmula en k-FNC puede tomarse una cláusula c de tamaño k, donde se puede generar un árbol de eliminación en el cual los

nodos tienen $2^k - 1$-hijos, si se cumple que no existan la variable v y $\bar{v}$ en la cláusula, debido a que existen $2^k - 1$ asignaciones que pueden satisfacer a la cláusula c, con este cambio ahora el número de niveles que puede tener el árbol es n/k, aunque cada nodo tenga $2^k - 1$ hijos. El uso de eliminación por cláusula permite solo tomar en cuenta aquellas asignaciones que pueden satisfacer la fórmula, por ejemplo en fórmulas en 2-FNC al elegir una cláusula se generan 3 hijos por nodo en el árbol eliminando 2 variables en un nivel, mientras que en la versión de eliminación por variable, para eliminar 2 variables es necesario generar 2 nodos para la primer variable y para la segunda genera otros 2 para cada uno de los anteriores.

2.2.3. Gráfica

Definición 1 *Una gráfica G es un par ordenado $(V(G), E(G))$ donde $V(G)$ es el conjunto de vértices y $E(G)$ es el conjunto de aristas, unidas con una función de incidencia φ_G que asocia cada arista de G con un par no ordenado de (no necesariamente distintos) vértices de G. Si e es una arista y u y v son vértices tal que $\varphi_G(e) = \{u, v\}$ entonces se dice que e une u y v, y los vértices u y v son llamados extremos de e. Para denotar el número de vértices y aristas en G se utiliza $v(G)$ y $e(G)$ llamados orden y tamaño de G, respectivamente [1].*

Se dice que si un vértice está conectado con otro mediante una arista éstos son adyacentes, y uno incide con el otro, además si los vértices unidos son distintos se dice que son vecinos. Tomando el concepto de gráfica la representación gráfica de una arista con vértices u y v es:

$$u \text{ — } v$$

Una arista con extremos idénticos es llamado un ciclo simple o loop, dos o más aristas con el mismo par de extremos son llamadas aristas paralelas, la Figura 2.1 muestra un ejemplo. Una gráfica simple es aquella que no tiene loops o aristas paralelas.

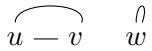


Figura 2.1: De izquierda a derecha se muestra un par de aristas paralelas y un ciclo simple

Un *camino* es una gráfica simple [1] cuyos vértices pueden ser ordenados en una secuencia lineal de tal forma que dos vértices son adyacentes si son consecutivos en la secuencia, o no adyacentes si no lo están. Un ciclo en tres o más vértices es una gráfica simple cuyos vértices pueden ser ordenados en una secuencia cíclica de manera que dos de ellos son adyacentes si son consecutivos en la secuencia, o no adyacentes si no lo están; un ciclo en un vértice consiste en un vértice con un loop, y un ciclo en dos vértices consiste de dos vértices unidos

por un par de aristas paralelas. Como se muestra en la Figura 2.2, la longitud de un camino o un ciclo es el número de aristas que incluye.

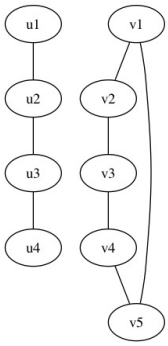


Figura 2.2: De izquierda a derecha un camino de longitud cuatro, y un ciclo de longitud cinco

Una gráfica es conectada si, para cada partición de sus vértices dentro de dos conjuntos no vacíos X y Y, existe una arista con un extremo en X y un extremo en Y, de otra forma se considera una gráfica no conectada. En otras palabras, una gráfica es desconectada si su conjunto de vértices puede ser dividido en dos conjuntos no vacíos X y Y y que ninguna arista tenga un extremo en X y otro en Y (Figura 2.3) [1].

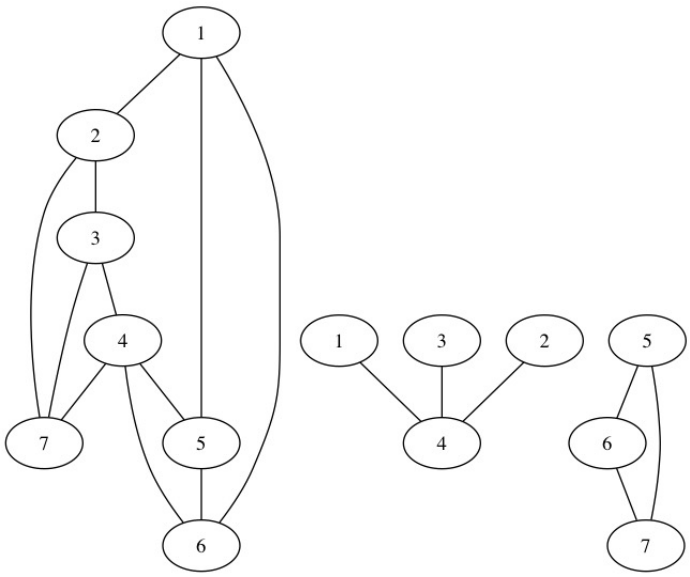


Figura 2.3: (a) Una gráfica conectada, y (b) una gráfica no conectada

Subgráficas

Dada una gráfica G, existen dos caminos naturales para derivar en gráficas más pequeñas de G. Si e es una arista de G y m el número de aristas en G, se puede obtener una gráfica con $m-1$ aristas eliminando a e de G pero dejando los vértices y las aristas restantes intactas. La gráfica resultante se denota por $G\backslash e$. De manera similar, si v es un vértice de G y G tiene n vértices, se puede obtener una gráfica con $n-1$ vértices eliminando de G el vértice v junto con las aristas incidentes en él. La gráfica resultante se denota por $G-v$ (Figura 2.4) [1].

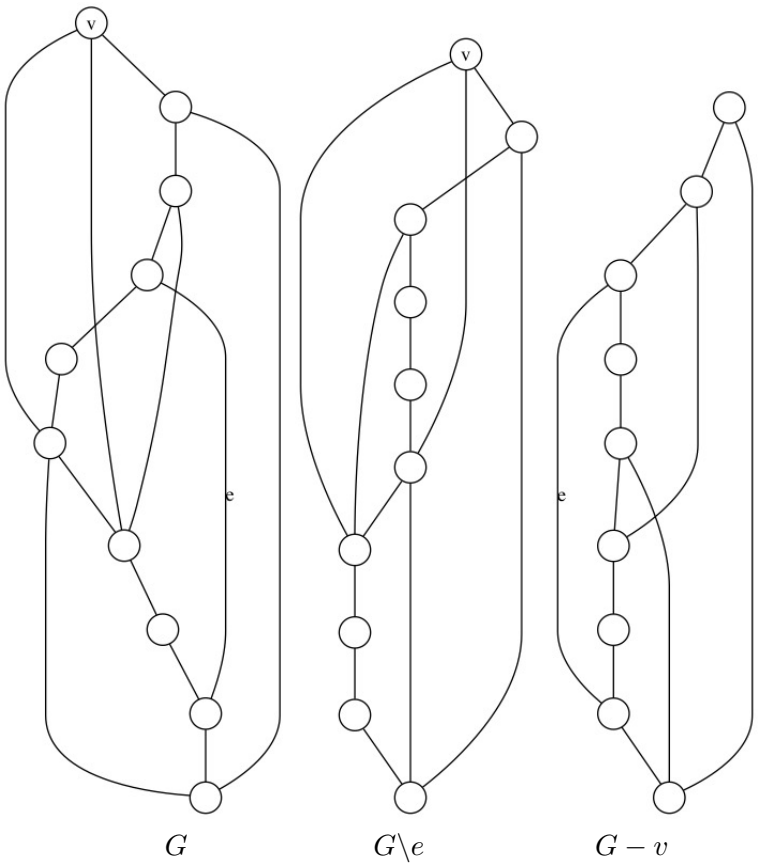


Figura 2.4: Subgráficas resultantes de eliminar la arista e y el vértice v respectivamente

Una gráfica es acíclica si no contiene un ciclo. Esto quiere decir que una gráfica acíclica debe contener un vértice de grado menor a dos.

Una arista de corte es aquella que al ser eliminada de una gráfica conectada, la convierte

en una gráfica desconectada [1].

Una subgráfica de expansión de una gráfica G es una subgráfica obtenida por medio de eliminación de aristas solamente, en otras palabras, es una subgráfica cuyo conjunto de vértices es el conjunto de vértices completo de G. Si S es el conjunto de aristas eliminadas, la subgráfica de G se denota como $G\backslash e$ [1](Figura 2.4).

Una gráfica acíclica conectada es llamada un árbol, en una gráfica acíclica desconectada cada componente es acíclico, por esta razón son llamados bosques.

Si G es una gráfica conectada y no es un árbol, y e es una arista de ciclo de G, entonces $G\backslash e$ es una subgráfica de expansión de G, ya que e no es una arista de corte de G. Repitiendo este proceso de eliminación de aristas en ciclos hasta que cada arista que se mantenga sea una arista de corte, se obtiene un árbol de expansión de G (Figura 2.4).

El complemento $E\backslash T$ de un árbol de expansión T es llamado coárbol, y se denota $\bar{T}$. Para cada arista $e = xy$ en el coárbol $\bar{T}$ de una gráfica G, existe un camino xy único en T que conecta a los extremos, llamado $P = xTy$. Entonces $T + e$ contienen un ciclo único, este ciclo es llamado ciclo fundamental de G con respecto a T y e.

Árbol de expansión

Un árbol de expansión A de una gráfica conectada G, es el conjunto de aristas que conectan a los vértices de G de tal forma que solo existe un camino entre dos pares de vértices cualesquiera, entonces $E(A) \subseteq E(G)$ y el número de aristas es $|E(A)| = |V(G)| - 1$.

Gráficas cactus

Este tipo de gráficas se caracterizan porque sus ciclos cumplen con las siguientes condiciones:

1. Cada arista de la gráfica pertenece a lo más a un ciclo.
2. Cualquier par de ciclos comparten a lo más un vértice.

Gráficas outerplanar

Este tipo de gráficas son cíclicas y no contienen una subdivisión de las gráficas k_4, que es la gráfica completa de 4 vértices, y la gráfica $k_{2,3}$, que es la gráfica bipartita completa con partes de tamaño 2 y 3.

Este tipo de gráficas puede ser representada por medio de cadenas poligonales, en las cuales estos poligonos comparten a lo más un lado con otro polígono, es decir dos poligonos no pueden estar unidos por dos lados, esto daría como resultado que contengan $k_{2,3}$ como un menor de la gráfica.

Estas gráficas se pueden representar también como un conjunto de ciclos sobre un camino en el cual no existe intersección entre cualquier par de ciclos.

2.2.4. Construcción de una gráfica a partir de una fórmula en 2-FNC

Existen algunas representaciones gráficas de una Forma Normal Conjuntiva, una es la gráfica primal signada o también conocida como gráfica restringida.

Sea F una 2-FNC, su gráfica restringida se denota por $G_F = (V(F), E(F))$ con $V(F) = v(F)$ y $E(F) = \{\{v(x), v(y)\} : \{x \vee y\} \in F\}$, esto es, los vértices de G_F son las variables de F, y para cada cláusula $\{x \vee y\}$ en F existe una arista $\{v(x), v(y)\} \in E(F)$. Para $x \in V(F), \delta(x)$ denota su grado, es decir el número de aristas incidentes en x. Cada arista $c = \{v(x), v(y)\} \in E(F)$ se asocia con un par (s_1, s_2) de signos, que se asignan como etiquetas de la arista que conecta las variables de la cláusula. Los signos s_1 y s_2 pertenecen a las variables x y y respectivamente. Por ejemplo la cláusula x^0, y^1 determina la arista con etiqueta "$x \overset{-+}{—} y$", que es equivalente a la arista "$y \overset{+-}{—} x$".

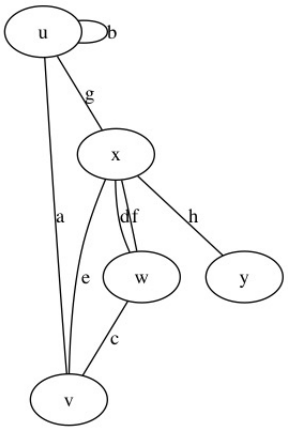
Sea $S = \{+, -\}$ un conjunto de signos. Una gráfica con aristas etiquetadas en S es el par (G, ψ), donde $G = (V, E)$ es una gráfica restringida, y ψ es una función con dominio E y rango S. $\psi(e)$ se denomina a la etiqueta de la arista $e \in E$. Sea $G = (V, E, \psi)$ una gráfica restringida con aristas etiquetadas en $S \times S$ y "x" y "y" nodos en V, si $e = \{x, y\}$ es una arista y $\psi(e) = (s, s')$, entonces $s(s')$ es el signo adyacente a $x(y)$.

Sea G una gráfica conectada de n vértices, un árbol de expansión de G es un subconjunto de $n - 1$ aristas tal que forman un árbol de G. Se denomina co-árbol al subconjunto de aristas que son el complemento de un árbol.

Búsqueda primero en profundidad

Este tipo de búsqueda es utilizado para construir el árbol de expansión de una gráfica y que al construirlo las aristas de retroceso, o ciclos, que se encuentren son el coárbol($\bar{T}$)

La matriz de adyacencia de G es de tamaño $n \times n$, denotada como $A_G := (a_{uv})$, donde a_{uv} es el número de aristas que unen los vértices u y v, cada loop cuenta como dos aristas en la matriz (Figura 2.5).



G

	u	v	w	x	y
u	2	1	0	1	0
v	1	0	1	1	0
w	0	1	0	2	0
x	1	1	2	0	1
y	0	0	0	1	0

A

Figura 2.5: Gráfica G y matriz de adyacencia A

La búsqueda primero en profundidad está basada en el concepto de árbol, en la cual el vértice añadido al árbol T en cada etapa es aquel que es vecino de la adición más reciente a T, en otras palabras, se debe realizar una búsqueda en la lista de adyacencia para el vértice más recientemente añadido x para un vecino que no está en T. Si existe ese vecino, es añadido a T, si existe uno y éste ya fue agregado a T se agrega una arista de retroceso en T y se agrega esta arista en $\bar{T}$, si no existe ese vecino se retrocede al vértice añadido a T justo antes de x y se examinan sus vecinos, hasta que todos los vértices son añadidos a T y las aristas son agregadas a T o $\bar{T}$.

2.3. Estado del arte

El problema $SAT(F)$, donde F es una fórmula booleana, consiste en decidir si F tiene un modelo, es decir una asignación a las variables de F tal que al ser evaluadas con respecto a la lógica booleana clásica devuelva un valor verdadero. Si F esta en dos Forma Normal Conjuntiva (2-FNC) entonces $SAT(F)$ se puede resolver en tiempo polinomial, sin embargo, si F esta en k-FNC, $k > 2$, entonces $SAT(F)$ es un problema NP-Completo. Por otro lado, existe el problema de conteo, denotado como $\#SAT(F)$, el cual consiste en contar el número de modelos de F. $\#SAT(F)$ es un problema de conteo a diferencia de $SAT(F)$ que se puede considerar como un problema de decisión. $\#SAT(F)$ pertenece a la clase #P-Completo aún cuando F este en 2-FNC, este último denotado como #2SAT [2].

Paturi et al. [10] propone un algoritmo para SAT basado en aplicar *s-resolución*. Resolución consiste en encontrar un par de cláusulas C_1 y C_2 tal que $v \in C_1$ y $\neg v \in C_2$. Para ese par, el resolvente, denotado por $R(C_1, C_2)$ es la cláusula $C = D_1 \vee D_2$ donde D_1 y D_2 son obtenidas eliminando v y $\neg v$ de C_1 y C_2 respectivamente. La s-resolución significa que dado un valor s entero, $|v(C_1)|, |v(C_2)| \leq s$ y $R(C_1, C_2) \leq s$, la complejidad en tiempo de este algoritmo está dada para $k = 3, O(1.3067^n)$ y $k = 4, O(1.4768^n)$.

Hertli et al. [11] propone una mejora al algoritmo presentado por Paturi en el cual aplica *s-implicación*. Se dice que v o $\neg v$ es *s-implicada* si existen al menos s número de cláusulas que impliquen el valor de v o $\neg v$ para todas las posibles soluciones, la complejidad en tiempo de este algoritmo está dada para $k = 3, O(1.30704^n)$ y $k = 4, O(1.46899^n)$.

Thurley [6] presenta un algoritmo para resolver *#k-SAT* de manera aproximada dividiendo el problema de acuerdo a la fórmula de entrada. Si la fórmula tiene pocas soluciones entonces es factible enumerar cada una de ellas utilizando un algoritmo basado en un árbol de eliminación. Por otro lado, si el número de soluciones es mayor, utiliza una ecuación que con probabilidad de $\frac{3}{4}$ da el número de modelos de manera correcta, esta probabilidad se establece ya que una cláusula en 2-Forma Normal Conjuntiva tiene cuatro posibles asignaciones de las cuales solo tres de ellas la satisfacen, la complejidad en tiempo de este algoritmo está dada para $k = 3, O(1.5366^n)$ y para $k = 4, O(1.6155^n)$.

Schmitt et al. [9] presenta una mejora al algoritmo de Thurley, modificando los árboles de eliminación, considerando cláusulas en lugar de variables como forma de eliminación, reduciendo así los niveles del árbol, la complejidad en tiempo de éste algoritmo está dada para $k = 3, O(1.51426^n)$ y para $k = 4, O(1.60816^n)$.

Burchard et al. [7] presenta un algoritmo exacto para calcular el número de modelos basado en el número de ramas que se satisfacen en el árbol de decisión. Aplica técnicas como: descomposición en subfórmulas, cache de fórmulas, preprocesado, deducción rápida y conflicto de cláusulas. La mejora que propone es que, al realizar decisiones dentro del árbol y utilizar hilos en cada decisión pueden surgir errores en los valores del cache y por lo tanto se pueden aplicar reglas para accesar al cache lo que soluciona este problema y permite resolverlo de manera paralela.

Marcial et al. [13] muestra un algoritmo exacto para calcular el número de modelos mediante la representación de la fórmula en una gráfica, en este algoritmo los ciclos que se encuentran en la gráfica determinan la dificultad para resolver el problema. Al generar un árbol con el método de búsqueda primero en profundidad si no se encuentran elementos en el co-árbol (aristas que denotan ciclos) puede calcular el número de modelos en tiempo polinomial.

López et al. [12] muestra un algoritmo para calcular el número de modelos utilizando también la representación en gráfica de la fórmula. El algoritmo divide la fórmula de entrada en particiones, es decir subfórmulas que contienen ciclos intersectados, por lo que el número

de modelos se puede calcular de forma independiente en cada partición. La complejidad del algoritmo se mejora respecto a [13].

Aunque el problema #2SAT es #P-Completo, existen instancias que se pueden resolver en tiempo polinomial [3]. Por ejemplo, si la gráfica que representa la fórmula es acíclica, entonces el número de modelos se puede calcular en tiempo polinomial. Actualmente, los algoritmos que se utilzan para resolver el problema para cualquier fórmula F en 2-FNC, descomponen F en subfórmulas hasta que se tienen casos base en los que se puede contar de forma eficiente. El algoritmo con la menor complejidad en tiempo hasta el momento fue desarrollado por Wahlström [4] el cual esta dado por $O(1.2377^n)$ donde n representa el número de variables de la fórmula. El algoritmo de Wahlström utiliza como criterio de elección de una variable, el número de veces que aparece en la fórmula (sea la variable o su negación). Los dos criterios de paro del algoritmo son cuando $F = \emptyset$ o cuando $\emptyset \in F$.

Por otro lado, están las implementaciones para #SAT (SAT solvers) en donde el objetivo es buscar estrategias que permitan resolver el problema de forma eficiente para instancias en donde el número de variables es considerable. Las herramientas reportadas en la literatura que hasta la fecha se consideran eficientes son relsat [5] y sharpSAT [6] ambas bajo el paradigma secuencial y count Antom [7] utilizando paralelismo.

La herramienta relsat esta basada en el algoritmo DPLL (Davis-Putnam algorithm) cuya ventaja es poder procesar rápidamente fórmulas disjuntas, es decir cuyos conjuntos de variables no se intersectan.

La herramienta sharpSAT es una mejora de relsat en donde se elige una literal heurísticamente, así mismo, utiliza descomposición de componentes y cache de componentes para agilizar el cálculo.

La herramienta Count Antom es una implementación paralela basada en sharpSAT cuyos resultados muestran que se mejora el tiempo en las instancias de prueba con respecto a sharpSAT.

En esta tesis se presentan algoritmos desarrollados para el conteo de modelos en fórmulas booleanas en 2-FNC para fórmulas que presentan gráficas de tipo cactus, así como fórmulas de manera general, y se comparan los resultados con sharpSAT.

2.4. Planteamiento del problema

La lógica proposicional es una rama de la lógica clásica que estudia las variables proposicionales o sentencias lógicas, sus posibles implicaciones, evaluaciones de verdad y en algunos casos su nivel absoluto de verdad.

Sea $X = \{x_1, \ldots, x_n\}$ un conjunto de n variables booleanas (es decir que pueden tomar únicamente dos valores de verdad) [19]. Una literal es una variable x_i o la variable negada $\bar{x_i}$. Una cláusula es una disyunción de literales distintas. Una fórmula booleana en forma normal conjuntiva (FNC) F es una conjunción de cláusulas. Sea $v(Y)$ el conjunto de variables involucradas en el objeto Y, donde Y puede ser una literal, una cláusula o una fórmula booleana. Por ejemplo, para la cláusula $c = \{x_1 \vee \bar{x_2}\}, v(c) = \{x_1, x_2\}$ y $Lit(c) = \{x_1, x_2, \bar{x_1}, \bar{x_2}\}$ es el conjunto de literales que aparecen en c.

Una asignación s es un conjunto de literales tomadas de una fórmula F, con la condición de que sólo una de las literales x_i o $\bar{x_i}$ pertenece a s. Puede también considerarse como un conjunto de pares de literales no complementario. Si $x_i \in s$, siendo s una asignación, entonces s convierte x_i en verdadero y $\bar{x_i}$ en falso. Por otro lado si $\bar{x_i} \in s$ entonces s convierte a $\bar{x_i}$ en verdadero y x_i en falso. Considerando una cláusula c y una asignación s como un conjunto de literales, se dice que c se satisface por s sí y solo si $c \cap s \neq \emptyset$ y si para toda $x_i \in c, \bar{x_i} \in s$ entonces s falsifica a c. Sea F una fórmula booleana en FNC, se dice que F se satisface por la asignación s si cada cláusula en F se satisface por s. Por otro lado, se dice que F se contradice por s si al menos una cláusula de F se falsifica por s. Un modelo de F es una asignación para $v(F)$ tal que satisface F. Se denota como $SAT(F)$ al conjunto de modelos de la fórmula F. Dada una fórmula F en FNC, SAT consiste en determinar si F tiene un modelo, mientras que #SAT consiste en contar el número de modelos que tiene F sobre $v(F)$. Por otro lado, #2SAT denota #SAT para fórmulas en 2-FNC.

2.5. Justificación

La complejidad computacional estudia la "dificultad" inherente de problemas de importancia teórica y/o práctica. El esfuerzo necesario para resolver un problema de forma eficiente puede variar, ésto depende de cómo se piensa abordar el problema y qué parte de él se pretende resolver.

Un problema se denomina #P si no existe un algoritmo eficiente para encontrar una solución óptima. Probar que un problema es #P es importante puesto que permite encontrar un algoritmo para la solución óptima y centrarse en objetivos realizables (encontrar algoritmos para obtener soluciones a una parte del problema.

El desarrollo de algoritmos para resolver problemas #P, en este caso #2SAT tiene como aplicaciones estimar el grado de veracidad en teorías proposicionales [17], generación de explicaciones a preguntas proposicionales, reparación de bases de datos inconsistentes, inferencia bayesiana [14, 15, 16, 17, 19, 18], entre otras.

2.6. Hipótesis

Existe un algoritmo que mejore el tiempo de conteo de modelos de fórmulas booleanas en 2-FNC con respecto a sharpSAT.

2.7. Objetivo general

Desarrollar un algoritmo para el conteo de modelos de fórmulas booleanas en 2-forma normal conjuntiva, basándose en su representación mediante gráfica y su descomposición en árbol-coárbol.

Objetivos Particulares

1. Desarrollar un algoritmo para el conteo de modelos de fórmulas booleanas en 2-FNC, cuya representación en gráficas sea cactus.
2. Desarrollar un algoritmo para el conteo de modelos de fórmulas booleanas en 2-FNC, cuya representación en gráficas sea outerplanar.
3. Desarrollar un algoritmo para el conteo de modelos en cualquier fórmula en 2-FNC utilizando particiones.
4. Desarrollar un algoritmo para el conteo de modelos en fórmulas en 2-FNC utilizando reducción múltiple.

2.8. Metodología propuesta

Los pasos que se llevarán a cabo durante ésta investigación son los siguientes:

1. Representar fórmulas booleanas en 2-FNC mediante gráficas. En esta etapa se busca crear la representación de una fórmula booleana mediante una gráfica que se descompondrá en árbol y coárbol.
2. Desarrollar un método para la descomposición de la fórmula de entrada así como la aplicación de reducción múltiple.
3. Desarrollar un método para el conteo de modelos en gráficas acíclicas. En esta etapa se realizará el conteo de modelos para una fórmula que en su representación de gráfica árbol-coárbol su coárbol sea vacio.

4. Desarrollar un método para el conteo de modelos sobre gráficas con aristas paralelas. En esta etapa se resolverán los casos en los que un par de variables aparezcan más de una vez en una misma cláusula.

5. Desarrollar un método para el conteo de modelos sobre gráficas con ciclos simples no intersectados, gráficas cactus. En esta etapa se propondrá un algoritmo para cualquier fórmula de entrada sin importar el número de veces que aparezca una variable o el número de relaciones que tenga, mientras no se contengan aristas paralelas y cualquier par de ciclos no contengan alguna arista en común.

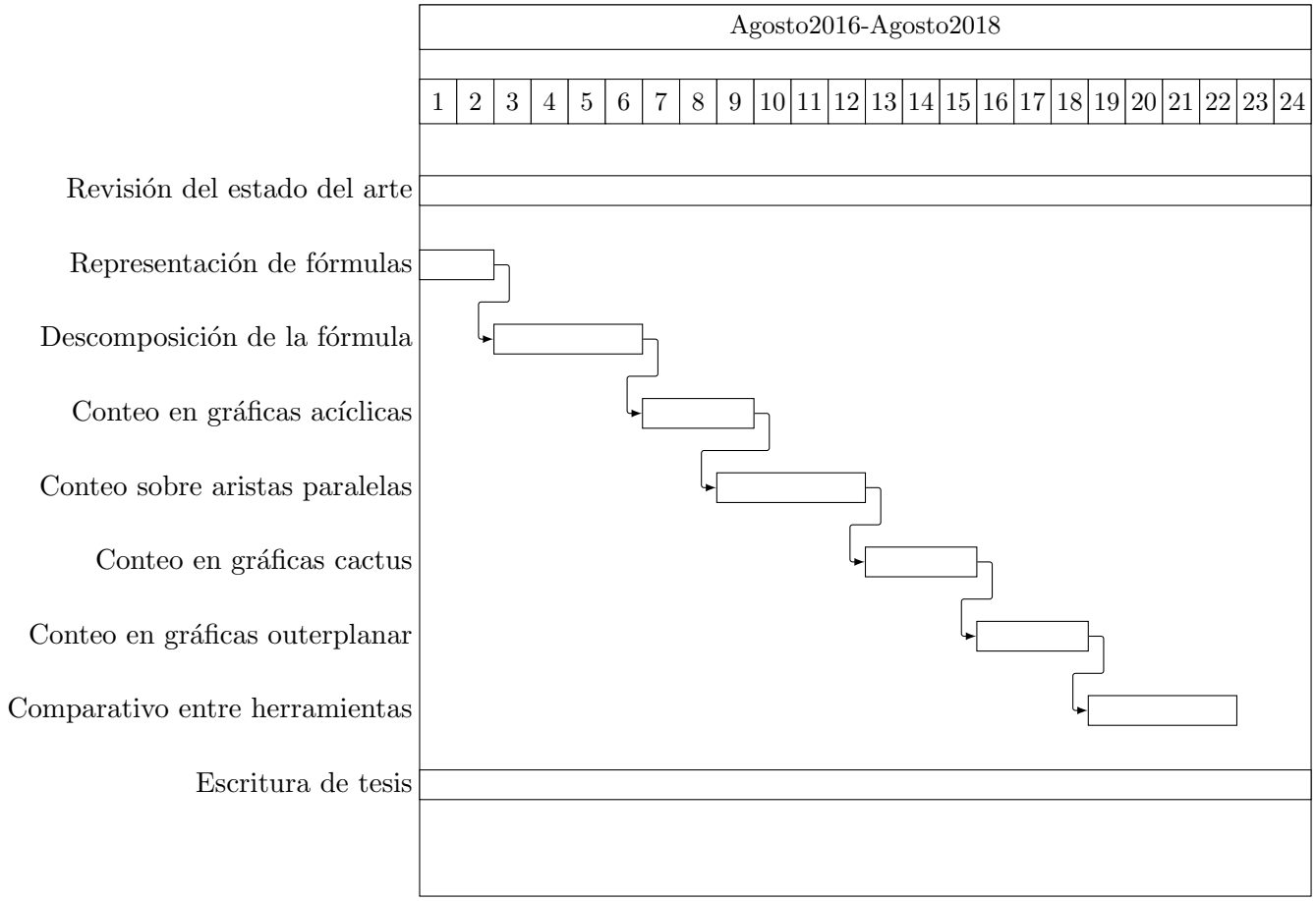
6. Desarrollar un método para el conteo de modelos sobre gráficas con ciclos simples e intersectados, con la restricción de que cualquier par de ciclos solo tengan en común una arista, que en la gráfica no exista la subgráfica K_4 ni $K_{2,3}$, es decir que sea outerplanar.

7. Hacer un comparativo con la herramienta que presenta una mayor eficiencia, sharpSAT.

Con estos pasos se sigue un esquema, los datos y su representación en el punto 1, preprocesamiento de la información en el punto 2, procesamiento de la información en los puntos 3, 4, 5, y la evaluación del resultado obtenido en el punto 7.

2.9. Cronograma de actividades

Las actividades que se van a realizar y el tiempo que se tomará para todas ellas se describe en el siguiente diagrama:



2.10. Infraestructura requerida

Para poder llevar a cabo este trabajo de investigación se requiere la siguiente infraestructura:

- Un equipo PC portátil o de escritorio con las siguientes características:
  - 16GB de memoria RAM
  - Disco duro de 250 GB.
  - Procesador intel core i3 o superior

2.11. Contribuciones esperadas

Las contribuciones originales que se espera se deriven de este trabajo de investigación son las siguientes:

- Un Artículo.
- Una Ponencia en Congreso.
- Trabajo escrito de obtención de grado.

Capítulo 3

Artículos publicados

En este capítulo se incluyen los artículos obtenidos del trabajo de investigación que fueron enviados a revistas y conferencias para su evaluación y publicación.

3.1. A Linear Time Algorithm for Solving #2SAT on Cactus Formulas

Este artículo fue enviado a la IEEE Transactions on Latin America, se encuentra en revisión actualmente.

A Linear Time Algorithm for Solving #2SAT on Cactus Formulas

M. A. López, J. R. Marcial, G. De Ita, H. A. Montes-Venegas and R. Alejo

***Abstract*— An $O(n+m)$-time algorithm is presented for counting the number of models of a two Conjunctive Normal Form Formula F that represents a Cactus graph, where n is the number of variables and m is the number of clauses of F. Although, it was already known that this class of formulas could be computed in polynomial time, we compare our proposal algorithm with two state of the art implementations for the same problem, *sharpSAT* and *countAntom*. The results of the comparison show that our algorithm outperforms both implementations, and it can be considered as a base case for general counting of two Conjunctive Normal Formulas.**

***Keywords*— #SAT, #2SAT, graph theory, complexity theory.**

I. INTRODUCCIÓN

EL PROBLEMA $SAT(F)$, donde F es una fórmula Booleana, consiste en decidir si F tiene un modelo, es decir, una asignación a las variables de F tal que, al ser evaluada con respecto a la lógica proposicional, devuelve un valor verdadero. Si F esta en dos Forma Normal Conjuntiva (2-FNC), entonces $SAT(F)$ se puede resolver en tiempo polinomial. Sin embargo, si F esta en k-FNC, $k > 2$, entonces $SAT(F)$ es un problema NP-Completo. Por otro lado, existe el problema de conteo denotado como $\#SAT(F)$ que consiste en contar el número de modelos de F. A diferencia de $SAT(F)$ que se puede considerar como un problema de decisión, $\#SAT(F)$ es un problema de conteo. $\#SAT(F)$ pertenece a la clase $\#P$-Completo aún cuando F este en 2-FNC, este último denotado como $\#2SAT$ [1].

Aunque el problema $\#2SAT$ es $\#P$-Completo, existen instancias que se pueden resolver en tiempo polinomial [2]. Por ejemplo, si la gráfica que representa la fórmula es acíclica, entonces $\#2SAT$ puede resolverse en tiempo polinomial. $\#2SAT$ es considerado un problema fundamental en el establecimiento de la frontera entre problemas de conteo intratables y aquellos que pueden resolverse eficientemente.

$\#SAT(F)$ puede reducirse a diferentes problemas en el área de razonamiento aproximado. Por ejemplo, cuando se quiere estimar el grado de creencia, en la generación de explicaciones para consultas en bases de datos lógicas, en la inferencia Bayesiana y en el mantenimiento de sistemas de razonamiento [3][4][5]. Los anteriores problemas provienen de aplicaciones de la Inteligencia Artificial, tales como planeación, sistemas expertos, razonamiento automático, etc.

Actualmente, los algoritmos utilizados para $\#SAT(F)$ para cualquier fórmula F en 2-FNC, descomponen F en subfórmulas hasta obtener casos base en los que se puede contar modelos de forma eficiente. El algoritmo de menor orden de complejidad conocido hasta el momento fue desarrollado por Wahlström [3], éste es de $O(1.2377^n)$, donde n representa el número de variables de la fórmula. El algoritmo de Wahlström utiliza como criterio de elección de una variable, el número de veces que aparece en la fórmula (sea la variable o su negación). Los dos criterios de paro del algoritmo son cuando $F = \emptyset$ o cuando $\emptyset \in F$.

Por otro lado, están las implementaciones para $\#SAT$ en donde el objetivo es buscar estrategias que permitan resolver el problema de forma eficiente para instancias en donde el número de variables es considerable. Las herramientas que a la fecha se consideran las más eficientes son *relsat* [6] y *sharpSAT* [7], ambas bajo el paradigma secuencial. También se han implementado métodos paralelos, como: *countAntom* [8].

relsat esta basado en el algoritmo DPLL (Davis-Putnam algorithm) cuya ventaja es poder procesar rápidamente fórmulas disjuntas, es decir, cuyos conjuntos de variables no se intersectan. La herramienta *sharpSAT* es una mejora de *relsat* en donde se elige una literal heurísticamente. Asimismo, utiliza descomposición y *cache* de componentes para agilizar el cálculo. La herramienta *countAntom* es una implementación paralela basada en *sharpSAT* cuyos resultados muestran que se mejora el tiempo en las instancias de prueba con respecto a *sharpSAT*.

En este artículo se presenta un algoritmo de compeljidad lineal en tiempo para el conteo de modelos de fórmulas en 2-FNC y cuya gráfica de restricciones sea tipo cactus. Los experimentos presentados muestran que nuestra propuesta mejora sustancialmente el tiempo para realizar el conteo de modelos con respecto a las herramientas de software actuales; *relsat, sharpSAT* y *countAntom*, por lo que nuestro algoritmo puede utilizarse como caso base en los algoritmos de conteo de modelos basados en descomposición de fórmulas en FNC.

II. PRELIMINARES

Sea $X = \{x_1, \ldots, x_n\}$ un conjunto de n variables Booleanas. Una literal es una variable x_i (x_i^1) o la variable negada $\neg x_i$ (x_i^0). Una cláusula es una disyunción de literales distintas. Una fórmula Booleana F en forma normal conjuntiva (FNC) es una conjunción de cláusulas.

Sea $v(Y)$ el conjunto de variables involucradas en el

M. A. López, Universidad Autónoma del Estado de México, México, mlopezm158@alumno.uaemex.mx

J. R. Marcial, Universidad Autónoma del Estado de México, México, jrmarcialr@uaemex.mx

G. De Ita, Benemérita Universidad Autónoma de Puebla, México, deita@cs.buap.mx

H. A. Montes-Venegas, Universidad Autónoma del Estado de México, México, hamontesv@uaemex.mx

objeto Y, dónde Y puede ser una literal, una cláusula o una fórmula Booleana. Por ejemplo, para la cláusula $c = \{x_1 \vee \neg x_2\},\ v(c) = \{x_1, x_2\}$.

Una asignación s para F es una función Booleana $s: v(F) \to \{0,1\}$. Una asignación puede también considerarse como un conjunto de pares de literales no complementario. Si $x^e \in s \quad e \in \{0,1\}$, siendo s una asignación, entonces s convierte a x^e en verdadero y a x^{1-e} en falso. Considerando una cláusula c y una asignación s como un conjunto de literales, se dice que c se satisface por s sí y solo si $c \cap s \neq \emptyset$ y si para toda $x^e \in c$, si $x^{1-e} \in s$, entonces s falsifica a c.

Sea F una fórmula Booleana en FNC, se dice que F se satisface por la asignación s si cada cláusula en F se satisface por s. Por otro lado, se dice que F se contradice por s si al menos una cláusula de F se falsifica por s. Un modelo de F es una asignación para $v(F)$ tal que satisface F.

Dada una fórmula F en FNC, *SAT* consiste en determinar si F tiene un modelo, mientras que *#SAT* consiste en contar el número de modelos que tiene F sobre $v(F)$. Por otro lado, *#2SAT* denota *#SAT* para fórmulas en 2-FNC.

II.I. La gráfica restringida de una 2-FNC.

Existen algunas representaciones gráficas de una Forma Normal Conjuntiva, en este caso, se utilizará la gráfica primal signada (gráfica restringida) [9].

Sea F una 2-FNC, su gráfica restringida se denota por $G_F = (V(F), E(F))$ donde los vértices de la gráfica son las variables $V(F) = v(F)$, y las cláusulas las aristas $E(F) = \{\{v(x), v(y)\} : \{x \vee y\} \in F\}$, esto es, para cada cláusula $\{x \vee y\}$ en F existe una arista $\{v(x), v(y)\} \in E(F)$. Para $x \in V(F)$, $\delta(x)$ denota su grado, es decir, el número de aristas incidentes en x. Cada arista $c = \{v(x), v(y)\} \in E(F)$ se asocia con un par (s_1, s_2) de signos, que se asignan como etiquetas de la arista que conecta las variables de la cláusula en la gráfica. Los signos s_1 y s_2 pertenecen a las variables x y y respectivamente. Por ejemplo, la cláusula $(x^0 \vee y^1)$ determina la arista con etiqueta "$x \overset{-+}{—} y$", que es equivalente a la arista "$y \overset{+-}{—} x$".

Sea $S = \{+, -\}$ un conjunto de signos. Una gráfica con aristas etiquetadas en S es el par (G, ψ), dónde $G = (V, E)$ es una gráfica restringida, y ψ es una función con dominio E y rango S. $\psi(e)$ se denomina a la etiqueta de la arista $e \in E$. Sea $G = (V, E, \psi)$ una gráfica restringida con aristas etiquetadas en $S \times S$ y x y y vértices en V, si $e = \{x, y\}$ es una arista y $\psi(e) = (s, s')$, entonces $s(s')$ es el signo adyacente a $x(y)$.

Note que la gráfica restringida $G_F = (V, E, \psi)$ de una 2-FC F puede contener aristas paralelas. Nosotros consideraremos gráficas simples (sin aristas paralelas), ya que éstas últimas pueden ser pre-procesadas, tal y como se presenta en [4], sin que este pre-procesamiento modifique la complejidad en tiempo de nuestra propuesta algorítmica.

Sea G una gráfica conectada de n vértices, un árbol de expansión de G es un subconjunto de $n-1$ aristas tal que forman un árbol de G. Se denomina coárbol al subconjunto de aristas que son el complemento de un árbol, cada una de estas aristas forman un ciclo en la gráfica.

Una gráfica cactus es una gráfica $G = (V_G, E_G)$ dónde:

- Cada arista de E_G pertenece a lo más a un ciclo.
- Cualquier par de ciclos comparten a lo más un vértice.

En este artículo, para encontrar el árbol de expansión y el coárbol se utiliza el método de búsqueda primero en profundidad [10] que permite construir la tupla (árbol, coárbol) de una gráfica.

III. ALGORITMO

En esta sección se presenta el algoritmo utilizado para el conteo de modelos en gráficas cactus en tiempo lineal. El método principal consiste de 3 pasos: construcción de una tabla en donde se almacenan las cláusulas, construcción de un árbol de expansión en donde se marcan los ciclos de la gráfica y finalmente, se realiza el conteo de modelos sobre el árbol de expansión. El algoritmo I muestra las entradas y salidas de cada paso.

ALGORITMO I
CONTEO DE MODELOS EN GRÁFICAS CACTUS

Entrada: Fórmula F
$t = \text{construcción\_Tabla(F, nF)}$; //nF = # variables de F
$arbol = \text{crear\_Arbol(t)}$;
$\{(\alpha_1, \beta_1), (\alpha_2, \beta_2)\} = \text{cuenta(arbol)}$;
return $\alpha_1 + \beta_1$;
Salida: Número de modelos de F

III.I. Construcción de la Tabla de Cláusulas.

Ya que la lectura de cláusulas se realiza desde un archivo en formato DIMACS (http://logic.pdmi.ras.ru/~basolver/dimacs.html), se utilizan dos tablas dinámicas para almacenar cada cláusula leída. El índice del renglón h de cada tabla representa a la variable x_h. Por cada cláusula (x_i, x_j), se agrega la entrada x_j al renglón i en la tabla uno, y la entrada x_i al renglón j en la tabla dos. La duplicidad de cláusulas permite agilizar las búsquedas durante la construcción del árbol de expansión. Adicionalmente, las cláusulas se insertan considerando el índice menor de sus dos variables. El algoritmo II muestra la construcción de la tabla.

III.II. Creación del árbol.

El par (árbol, coárbol) se crea a partir de las tablas que almacenan cláusulas. El algoritmo utilizado para la construcción del árbol es *primero en profundida* (*depth first search*, por sus siglas en inglés) [10]. Cada vez que se lee una cláusula, se revisa si ambos vértices están ya en el árbol, si es el caso, se marca el camino entre ambos vértices para denotar que se tiene una arista del coarbol, es decir, se encontró un ciclo.

El algoritmo III muestra la forma en que se recorre la tabla para insertar sus elementos en el árbol, mientras que el

algoritmo IV presenta la forma de insertar una variable en el árbol considerando si ésta aparece de forma positiva o negativa en la cláusula de la que proviene.

ALGORITMO II
CONSTRUCCIÓN DE LA TABLA DE CLÁUSULAS

Entrada: Fórmula F, # variables de F nF
vector t_1 de tamaño $nF+1$;
vector t_2 de tamaño $nF+1$;
para cada *clausula* $C=\{v_1,v_2\}$ *de* F **hacer**
  si $v_1 \leq v_2$ **entonces**
    agregar $\{v_1,v_2\}$ a $t1[v_1]$;
    agregar $\{v_1,v_2\}$ a $t2[v_2]$;
  en otro caso
    agregar $\{v_2,v_1\}$ a $t1[v_2]$;
    agregar $\{v_2,v_1\}$ a $t2[v_1]$;
  fin
fin
return t;
Salida: Tabla de cláusulas t

Como se puede apreciar en el algoritmo IV, se tiene la condición para conocer si la gráfica de entrada es cactus, lo anterior con la finalidad de hacer una comparación equitativa con las herramientas *sharpSAT* y *countAntom*, ya que en ellas no se conoce de antemano si la gráfica de entrada es cactus.

ALGORITMO III
CONSTRUCCIÓN DEL ÁRBOL UTILIZANDO BÚSQUEDA PRIMERO EN PROFUNDIDAD

Entrada: Tablas de cláusulas t_1, t_2
$i=1, T=NULL$;
mientras $t_1[i]==NULL$ *and* $t_2[i]==NULL$ **hace**
  i++
fin
mientras *true* **hacer**
  si $t_1[i]\neq NULL$ *or* $t_2[i]\neq NULL$ **entonces**
    si $t_1[i]\neq NULL$ **entonces**
      $c=t_1[i][0]$;
    en otro caso
      $c=t_2[i][0]$;
    fin
    i = agregar(c,T);
  en otro caso
    si i *tiene padre* **entonces**
      i = padre de i en T;
    en otro caso
      return T;
    fin
  fin
fin
Salida: Un árbol T

III.III. Conteo de modelos.

Una vez que se tiene el árbol de la gráfica cactus con los caminos de los ciclos marcados (coárbol), se realiza el conteo de los modelos mediante un recorrido en postorden del árbol. A cada nodo x_i del árbol se le asigna un par (α_i,β_i) donde α_i denota el número de modelos en donde la variable x_i toma un valor verdadero y β_i el número de modelos en donde x_i toma un valor falso. Los modelos contabilizados por cada par serán relativos a la subfórmula hasta ese momento calculada.

Si el nodo del árbol esta marcado como de apertura de un ciclo, se genera un segundo par $(\alpha_{2_i},\beta_{2_i})$ que estará activo hasta que el ciclo se cierre. El número de modelos sobre el nodo de cierre de ciclo, se calcula como: $(\alpha_i,\beta_i)-(\alpha_{2_i},\beta_{2_i})$, donde la resta se realiza a pares.

ALGORITMO IV
AGREGAR VÉRTICES AL ÁRBOL Y DETERMINAR SI LA GRÁFICA ES CACTUS.

Entrada: Cláusula c=v_1, v_2, árbol T
Si v_1 no está en T, agregar v_1 como la raíz de T;
si $v_1==v_2$ **entonces**
  si los signos de v_1 y v_2 son iguales marcar v_1 como unitario;
  si no es unitario ignorar la cláusula;
  return v_1;
fin
si v_2 *no está en* T **entonces**
  agregar v_2 como nodo hijo de v_1 return v_2;
en otro caso
  dibujar o marcar el camino de v_1 a v_2;
  si *si el camino ya fue marcado* **entonces**
    la gráfica no es cactus, termina el algoritmo;
  en otro caso
    marcar dependiendo del signo, el inicio (el nodo que se encuentre en un nivel más bajo del árbol) y cierre (la última arista marcada en el camino hacia el nodo final) del ciclo para el conteo;
  fin
fin
Salida: entero i

Inicialmente, a los nodos hoja se les asigna el par (1,1) y si el nodo hoja es a la vez la apertura de un ciclo, se le asigna como segundo par (0,1). Durante el recorrido, el par de cada nodo interior se calcula utilizando la recurrencia (1) que se aplica de acuerdo a los signos de las variables del nodo actual y del nodo hijo (es decir, de la cláusula representada por la arista hijo-padre en el árbol).

$$(\alpha_i,\beta_i)=\begin{cases}(\beta_{i-1},\alpha_{i-1}+\beta_{i-1})\ if\ (\epsilon_i,\delta_i)=(-,-)\\(\alpha_{i-1}+\beta_{i-1},\beta_{i-1})\ if\ (\epsilon_i,\delta_i)=(-,+)\\(\alpha_{i-1},\alpha_{i-1}+\beta_{i-1})\ if\ (\epsilon_i,\delta_i)=(+,-)\\(\alpha_{i-1}+\beta_{i-1},\alpha_{i-1})\ if\ (\epsilon_i,\delta_i)=(+,+)\end{cases} \quad (1)$$

Si un nodo interior es la apertura de un ciclo, su segundo par se calcula dependiendo de su signo, la recurrencia (2) establece el valor para el segundo par

$$(\alpha_{2_i}, \beta_{2_i}) = \begin{cases} (0, \beta_{2_i-1}) & \text{si } s = + \\ (\alpha_{2_i-1}, 0) & \text{si } s = - \end{cases} \quad (2)$$

Finalmente, si el nodo que se esta evaluando tiene mas de un hijo, el número de modelos se calcula mediante el producto de los pares de cada uno de ellos. El algoritmo V presenta el conteo de los modelos en el árbol.

Más detalles del método de conteo se puede consultar en [11], en donde se muestra cómo contar el número de modelos para árboles y ciclos por separado. En este mismo artículo se puede ver la demostración de la validez del método. Finalmente, el total de modelos se obtiene cuando se llega al nodo raíz x_r del árbol, y se deriva de la suma de sus dos componentes: α_r y β_r.

ALGORITMO V
CONTEO DE MODELOS EN EL ÁRBOL QUE REPRESENTA LA GRÁFICA CACTUS.

Entrada: árbol T
para cada *D hijo de T* **hacer**
  $\{(\alpha_{D_1}, \beta_{D_1}), (\alpha_{D_2}, \beta_{D_2})\}$=cuenta(D);
fin
si *T es hoja* **entonces**
  $(\alpha_{T_1}, \beta_{T_1}) = (1,1)$;
  $(\alpha_{T_2}, \beta_{T_2}) = (1,1)$;
  aplicar (2) si es necesario;
  return $\{(\alpha_{T_1}, \beta_{T_1}), (\alpha_{T_2}, \beta_{T_2})\}$;
en otro caso
  $(\alpha_r, \beta_r) = (1,1)$;
  para cada *D hijo de T* **hacer**
    aplicar (1) para $\{(\alpha_{D_1}, \beta_{D_1}), (\alpha_{D_2}, \beta_{D_2})\}$;
    si *la arista entre D y su padre T está marcada como cierre de ciclo* **entonces**
      aplicar (2) para $(\alpha_{D_2}, \beta_{D_2})$;
      aplicar (3) $(\alpha_i, \beta_i) = (\alpha_{D_1}, \beta_{D_1}) - (\alpha_{D_2}, \beta_{D_2})$;
    fin
    $(\alpha_r, \beta_r) = (\alpha_r, \beta_r) * (\alpha_i, \beta_i)$;
  fin
  return $\{(\alpha_r, \beta_r), (\alpha_r, \beta_r)\}$;
fin
Salida: $\{(\alpha_1, \beta_1), (\alpha_2, \beta_2)\}$

IV. RESULTADOS

En esta sección se muestran los resultados de comparar la propuesta aquí presentada contra *sharpSAT* y *countAntom*, las dos herramientas más eficientes hasta el momento reportadas en la literarura para resolver #SAT. Se realizaron dos tipos de pruebas, la primera consistió en generar de manera aleatoria 22 gráficas cactus que tienen entre 9,000 y 240,000 vértices y entre 12,500 y 320,000 aristas. La segunda prueba consisitió en generar gráficas cactus con el número máximo de ciclos que puede tener este tipo de gráficas, este es el peor de los casos en donde por cada tres vértices se generó un ciclo. Ya que *sharpSAT* y *countAntom* utilizan la estructura de la fórmula para realizar el conteo, éste segundo tipo de prueba generan los mejores casos para estas herramientas y para este tipo de gráficas.

Todas las pruebas se realizaron en una Computadora con procesador de dos núcleos a una velocidad de 2.4 Mhz con 8 GB en RAM y sistema operativo Ubuntu, ya que éste es el sistema operativo en donde se provee el código fuente para poder compilar y ejecutar ambas herramientas.

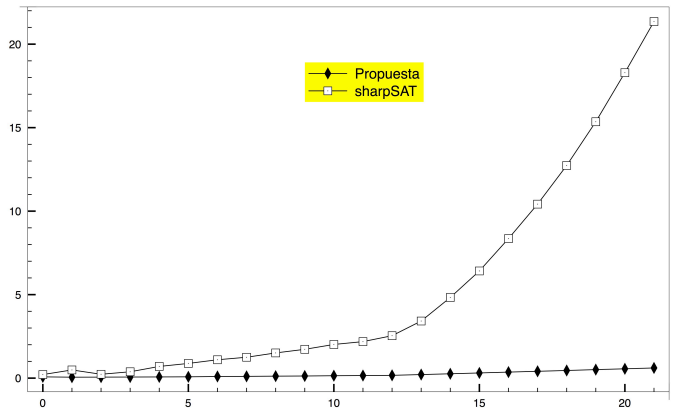
La Tabla 1 muestra los resultados de realizar el primer tipo de pruebas. Como se puede observar, nuestra propuesta obtiene mejores tiempos de ejecución en los 22 casos considerados, con respecto a las dos herramientas antes mencionadas. El tiempo de ejecución máximo para el conteo de modelos fue de 25 minutos. También se puede observar que *countAntom* no fue capaz de producir una respuesta para 17 de los 22 casos.

TABLA I
TIEMPO DE EJECUCIÓN DEL CONTEO DE MODELOS PARA 22 GRÁFICAS CACTUS. TIEMPO REPORTADO EN SEGUNDOS

	Vértices	Aristas	Ésta Propuesta	sharpSAT	countAntom
1	9382	12508	0.076	0.213	11.297
2	19999	26664	0.056	0.490	851.425
3	10000	13332	0.060	0.231	427.430
4	16000	21332	0.065	0.380	197.042
5	25999	34664	0.071	0.700	1334.007
6	30001	40000	0.078	0.880	----------
7	36001	48000	0.098	1.1	----------
8	40000	53332	0.106	1.250	----------
9	46000	61332	0.118	1.509	----------
10	49999	66664	0.127	1.725	----------
11	55999	74664	0.144	2.016	----------
12	60001	80000	0.154	2.186	----------
13	66001	88000	0.167	2.540	----------
14	79999	106664	0.213	3.419	----------
15	100000	133332	0.260	4.828	----------
16	120001	160000	0.310	6.418	----------
17	139999	186664	0.361	8.354	----------
18	160000	213332	0.409	10.420	----------
19	180001	240000	0.457	12.734	----------
20	199999	266664	0.510	15.354	----------
21	220000	293332	0.558	18.293	----------
22	240001	320000	0.609	21.353	----------

La Gráfica 1 muestra el crecimiento en el tiempo de ejecución, tanto para *sharpSAT* como para nuestra propuesta. En la gráfica no se incluyen los tiempos de *countAntom*, ya que a partir de la sexta entrada, esta herramienta tardó mas de 25 minutos en dar una respuesta, por lo que se considera que sus tiempos de respuesta no son competitivos con respecto a los otros dos programas. Como se puede observar en la Gráfica 1, nuestra propuesta obtiene mejores tiempos de ejecución en todos los casos que los obtenidos con *sharpSAT*.

GRÁFICA I
CRECIMIENTO DEL TIEMPO DE EJECUCIÓN PARA 22 GRÁFICAS CACTUS. EL TIEMPO REPORTADO ESTA EN SEGUNDOS



Para realizar pruebas considerando los peores casos para nuestra propuesta, se generaron gráficas cactus con el máximo número de ciclos posibles, es decir, se formó un ciclo por cada tres vértices. La diferencia principal con este tipo de gráficas es que el árbol generado tiene un tercio de nodos de inicio y un tercio de nodos de cierre de ciclos lo cual significa que se tienen dos variables de conteo en todo el recorrido del árbol.

La Tabla II muestra los resultados del tiempo de ejecución para el conteo de modelos sobre este tipo de gráficas. Como se puede observar en los 7 casos, nuestra propuesta obtiene menores tiempos de ejecución que el de las otras dos herramientas. Similar al primer caso, *counAntom* dio un resultado para los primero 5 casos, sin embargo, para los dos últimos no fue capaz de producir un resultado en los 5 minutos de ejecución que se dieron como máximo em esta segunda prueba.

TABLA II
TIEMPO DE EJECUCIÓN DEL CONTEO DE MODELOS EN 7 GRÁFICAS CACTUS CON EL MÁXIMO NÚMERO DE CICLOS. TIEMPO REPORTADO EN SEGUNDOS

	Vértices	Aristas	Ésta Propuesta	sharpSAT	countAntom
1	9999	14997	0.028	0.240	1.637
2	15999	23997	0.05	0.399	8.562
3	19999	29997	0.062	0.515	8.960
4	39999	59997	0.115	1.312	50.023
5	79999	119997	0.225	3.572	239.328
6	159999	239997	0.446	11.256	----------
7	239999	359997	0.669	23.413	----------

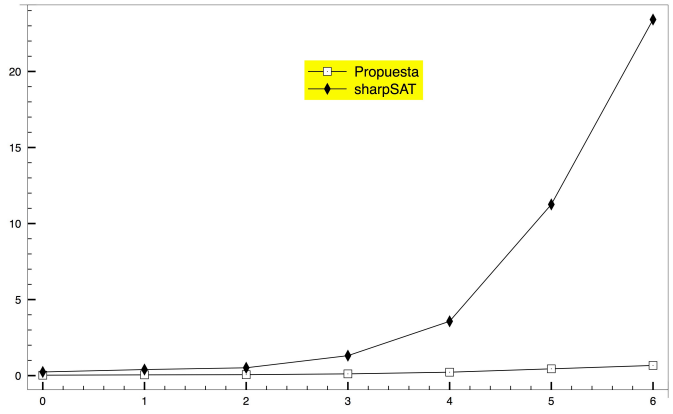
La Gráfica II muestra el crecimiento de los tiempos de ejecución del conteo de modelos para *sharpSAT* y para nuestra propuesta. Al igual que en la Gráfica I, no se muestran los resultados para *countAntom,* ya que sus tiempos reportados no son competitivos con respecto a los reportados para los otros dos programas. Similar a los resultados reportados para la primera prueba, nuestra propuesta mejora significativamente el tiempo de ejecución con respecto al de *sharpSAT*.

IV. COMPLEJIDAD EN TIEMPO DEL ALGORITMO

El algoritmo de esta propuesta consta de tres partes, crear tabla, crear árbol y conteo de modelos en el árbol. Para la creación de la tabla es suficiente con recorrer las cláusulas de la fórmula de entrada por lo que la complejidad de este paso es de orden $O(m)$. Mientras que para creación del árbol, se recorre cada entrada en la tabla, lo que requiere también del orden $O(m)$ operaciones básicas.

Finalmente, el recorrido del árbol es en postorden, y mientras se visitan los nodos del árbol, se va también visitando cada una de las aristas (cláusulas) de la gráfica de restricciones, al mismo tiempo que se van aplicando las recurrencias (1) y (2). Este último proceso nos genera del orden de $O(n+m)$ operaciones básicas. Así, la complejidad en tiempo de nuestra propuesta algorítmica es lineal y de orden $O(n+m)$.

GRÁFICA II
CRECIMIENTO DEL DEL TIEMPO DE EJECUCIÓN PARA 7 GRÁFICAS CACTUS QUE REPRESENTAN EL PEOR CASO PARA NUESTRA PROPUESTA. TIEMPO REPORTADO EN SEGUNDOS



V. CONCLUSIONES

En este artículo se presentó un algoritmo para el conteo de modelos de fórmulas Booleanas en 2-FNC y cuya gráfica de restricciones es tipo cactus. El algoritmo presentado tiene una complejidad en tiempo de orden $O(m+n)$, donde m es el número de cláusulas y n el número de variables de la fórmula de entrada. Nuestra propuesta detecta en tiempo lineal y al inicio de su procesamiento, si la fórmula de entrada es representada efectivamente por una gráfica de restricciones tipo cactus simple.

Las pruebas realizadas sobre dos diferentes clases de fórmulas muestran que la propuesta aquí presentada obtiene mejores tiempos de ejecución con respecto a las dos herramientas que se reportan en la literatura como las más eficientes; *sharpSAT* y *countAtom*para resolver este tipo de problemas de conteo. Por tanto, se considera que nuestro algoritmo puede incluirse como un caso base en el conteo de modelos de *sharpSAT* o de otras herramientas que realicen descomposición de la fórmula de entrada, típico de los métodos de ramificación y corte, hasta llegar a los casos bases.

Para el caso general #SAT, se podrían realizar descomposiciones sobre la fórmula de entrada, hasta generar gráficas cactus y entonces, procesar éstas últimas subfórmulas usando nuestro algoritmo, lo que creemos que puede impactar en la complejidad en tiempo de estos algoritmos de conteeo.

REFERENCIAS

[1] Brifhtwell G., Winkler Peter. (1991). Counting linear extensions. *Order*, Vol. 8, issue e, pp. 225-242.

[2] Paulusma, D., Slivovsky, F., Szeider, S. (2013). Model counting for CNF formulas ofbounded modular treewidth. In: Portier, N., Wilke, T. (eds.) STACS. LIPIcs,vol. 20, pp. 55–66. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

[3] Magnus Wahlström. (2008). A tighter bound for counting max-weight solutions to 2SAT instances. *In Proc. 3rd Int. Workshop on Parametrical and Exact Computation(IWPEC)*, pp. 202-213.

[4] De Ita G. (2004). Polynomial Classes of Boolean Formulas for Computing the Degree of Belief, *Lectures Notes in Artificial Intelligence* Vol. 3315, pp. 430-440.

[5] Roth D., (1996). On the hardness of approximate reasoning, *Artificial Intelligence,* Vol. 82, pp. 273-302

[6] Bayardo R. and R. Schrag. (1997). Using CSP look-back techniques to solve real-world SAT instances. In: *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*. New Providence, RI, pp. 203-208.

[7] Thurley M. (2006). sharpSAT – Counting models with Advanced Component Caching and Implicit BCP. *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing (SAT 2006)*. pp. 424-429.

[8] Jan Burchard, Tobias Schubert, Bernd Becker. (2015). Laissez-Faire Caching for Parallel #SAT Solving. *SAT 2015*. pp. 46-61.

[9] Szeider S. (2004). On fixed-parameter tractable parameterizations of SAT. *In: Giunchiglia, E., Taccella, A. (eds.) SAT 2003*. LNCS, vol 2919, pp. 188-202. Springer, Heidelberg.

[10] Bondy J. A. and Murty U.S.R. Graph Theory. Springer Verlag, Graduate Texts in Mathematics, 2010.

[11] Marcial-Romero J. R., De Ita G., Hernández, J. A., Valdovinos, R. M. A. (2015). Parametric Polynomial Deterministic Algorithm for #2SAT, *Lecture Notes in Computer Science*, Vol. 9413, pp. 202-213.

Marco Antonio López Medina, En el 2016 obtiene el título de Ingeniero en Computación en la Universidad Autónoma del Estado de México. Actualmente estudia la Maestría en Ciencias de la Ingeniería con línea de acentuación en Computación en la Universidad Autónoma del Estado de México.

José Raymundo Marcial-Romero, Actualmente profesor investigador de tiempo completo en la Facultad de Ingeniería de la Universidad Autónoma del Estado de México. Miembro del Sistema Nacional de Investigadores del Consejo Nacional de Ciencia y Tecnología, Nivel 1. En el año 2000 obtuvo el título de Licenciatura en Ciencias de la Computación y en el año 2007 el grado de Doctor en Ciencias de la Computación por The University of Birmingham UK en The School of Computer Science.

Guillermo De Ita Luna, Obtuvo su doctorado en Ingeniería Eléctrica por el CINVESTAV-IPN, México. Ha trabajado como desarrollador y consultor en sistemas de bases de datos y sistemas de información geográfica para diferentes empresas en México. Ha realizado estancias de investigación en la Universidad de Chicago, Texas A&M, INAOEP Puebla, en el instituto INRIA en Lille-1 y en la Fac. de Ing. de la UAEMEX. Actualmente, es profesor investigador de la Facultad de Ciencias de la Computación, BUAP, Puebla, México.

Héctor Alejandro Montes-Venegas, Es actualmente profesor investigador de tiempo completo en la Facultad de Ingeniería de la Universidad Autónoma del Estado de México. Es también ingeniero en Sistemas Computacionales por el Instituto Tecnológico de León y Maestro en Ciencias Computacionales por el Instituto Tecnológico y de Estudios Superiores de Monterrey.

Roberto Alejo Eleuterio Doctor en Ciencias Computacionales, adscrito al Instituto de Estudio Superiores de Jocotitlán, Miembro del Sistema Nacional de Investigadores del CONACYT. Los intereses en investigación se centran en la aplicación de inteligencia artificial a la solución de problemas reales.

3.2. A fast and efficient method for #2SAT via graph transformations

Este artículo fue enviado y aceptado en la conferencia MICAI 2017, publicado en Lecture Notes in Artificial Intelligence Vol. 10632 (eBook ISBN 978-3-030-02837-4, Softcover ISBN 978-3-030-02836-7) - 10633 (eBook ISBN 978-3-030-02840-4, Softcover ISBN 978-3-030-02839-8), .

A fast and efficient method for #2SAT via graph transformations

Marco A. López[1], J. Raymundo Marcial-Romero[1], Guillermo De Ita[2], and Rosa M. Valdovinos[1]

[1] Facultad de Ingeniería, UAEM
[2] Facultad de Ciencias de la Computación BUAP
mlopezm158@alumno.uaemex.mx,jrmarcialr@uaemex.mx, deita@cs.buap.mx, rvaldovinosr@uaemex.mx

Abstract. In this paper we present an implementation (markSAT) for computing #2SAT via graph transformations. We transform the input formula into a graph and test whether it is what we call cactus. If it is not the case the formula is decomposed until cactus sub-formulas are obtained. We compare the efficiency of markSAT against sharpSAT which is the leading sequential algorithm in the literature for computing #SAT obtaining better results.

1 Introduction

Given a Boolean formula F, SAT(F) consists on deciding whether F has a model, e.g. whether there exists an assignment to the variables of F whose evaluation with respect to propositional logic is true. If F is in two Conjunctive Normal Form (2-CNF), then SAT(F) can be computed in polynomial time [11]. However if F is given in k-CNF, ($k > 2$) then $SAT(F)$ is NP-Complete. On the other hand, #SAT(F) consists on counting the number of models that F has. In this way #SAT(F) belongs to the $\#P$-Complete class even if F is in 2-CNF, denoted as #2SAT [3].

Even though #2SAT is #P-Complete, there are instances which can be solved in polynomial time [5]. For example, if the graph which represents the input formula is acyclic, then #2SAT can be solved in polynomial time.

Actually, the algorithms to solve #SAT(F) for any F in k-CNF decompose the formula in sub-formulas until some criteria are met in order to determine their models. Nowadays, the algorithm with the smallest time complexity reported in the literature for 2-CNF formulas was given by Wahlström [9]. He reported a time complexity of $O(1.2377^n)$ where n is the number of variables of the input formula. Schmitt et al. [6] present an algorithm for #SAT where the criteria to decompose the input formula consists on choosing a clause, however their algorithm does not improve the bound given by Wahlström for #2SAT.

On the other hand, the implementations for #SAT(F) is focusing on searching strategies which allow solving #SAT(F) efficiently although the number of variables increases. The leading sequential implementations are relsat [2] and

sharpSAT [8]. Additionally, a parallel implementation, called countAntom [4] has been proposed.

In this paper we present a method which transforms the input formula F into a graph. If an intersected cycle is found, the formula is decomposed applying the Schmitt's rule. The process is iterated until no intersected cycles are found. For each subgraph of F, generated during the decomposition, without intersected cycles, a linear time algorithm is applied to compute #2SAT(G_i). We experimentally show that our proposal is faster than sharpSAT. It is worth to mention that the use of Schmitt et al. algorithm instead of Wahlström is given by the stop criteria based on cycles rather than variables, so in the worst case the time complexity of our algorithm coincides with Schmitt et al.

2 Preliminaries

Let $X = \{x_1, \ldots, x_n\}$ be a set of n Boolean variables. A literal is either a variable x_i or a negated variable $\overline{x}_i$. As usual, for each $x_i \in X$, we write $x_i^0 = \overline{x}_i$ and $x_i^1 = x_i$. A clause is a disjunction of different literals (sometimes, we also consider a clause as a set of literals). For $k \in N$, a k-clause is a clause consisting of exactly k literals and, a $(\leq k)$-clause is a clause with at most k literals. A variable $x \in X$ appears in a clause c if either the literal x^1 or x^0 is an element of c.

A Conjunctive Normal Form (CNF) F is a conjunction of clauses (we also call F a Conjunctive Form). A k-CNF is a CNF containing clauses with at most k literals.

We use $\nu(Y)$ to express the set of variables involved in the object Y, where Y could be a literal, a clause or a Boolean formula. $Lit(F)$ is the set of literals which appear in a CNF F, i.e. if $X = \nu(F)$, then $Lit(F) = X \cup \overline{X} = \{x_1^1, x_1^0, \ldots, x_n^1, x_n^0\}$. We also denote $\{1, 2, \ldots, n\}$ by $[[n]]$.

An assignment s for F is a Boolean function $s : \nu(F) \to \{0, 1\}$. An assignment can be also considered as a set which does not contain complementary literals. If $x^\epsilon \in s$, being s an assignment, then s turns x^ϵ true and $x^{1-\epsilon}$ false, $\epsilon \in \{0, 1\}$. Considering a clause c and assignment s as a set of literals, c is satisfied by s if and only if $c \cap s \neq \emptyset$, and if for all $x^\epsilon \in c$, $x^{1-\epsilon} \in s$ then s falsifies c.

If $F_1 \subset F$ is a formula consisting of some clauses of F, then $\nu(F_1) \subset \nu(F)$, and an assignment over $\nu(F_1)$ is a partial assignment over $\nu(F)$.

Let F be a Boolean formula in CNF, F is satisfied by an assignment s if each clause in F is satisfied by s. F is contradicted by s if any clause in F is contradicted by s. A model of F is an assignment for $\nu(F)$ that satisfies F. We will denote as $SAT(F)$ the set of models for the formula F.

Given a CNF F, the SAT problem consists on determining if F has a model. The #SAT problem consists of counting the number of models of F defined over $\nu(F)$. In this case #2-SAT denotes #SAT for formulas in 2-CNF.

2.1 The signed primal graph of a 2-CNF

There are some graphical representations of a CNF (see e.g. [7]), in this work the signed primal graph of a 2-CNF is used.

Let F be a 2-CNF, its signed primal graph (constraint graph) is denoted by $G_F = (V(F), E(F))$, with $V(F) = \nu(F)$ and $E(F) = \{\{\nu(x), \nu(y)\} : \{x, y\} \in F\}$. That is, the vertices of G_F are the variables of F, and for each clause $\{x, y\}$ in F there is an edge $\{\nu(x), \nu(y)\} \in E(F)$. For $x \in V(F)$, $\delta(x)$ denotes its degree, i.e. the number of incident edges to x. Each edge $c = \{\nu(x), \nu(y)\} \in E$ is associated with an ordered pair (s_1, s_2) of signs, assigned as labels of the edge connecting the literals appearing in the clause. The signs s_1 and s_2 are related to the literals x^ϵ and y^δ, respectively. For example, the clause $\{x^0, y^1\}$ determines the labelled edge: "$x \overset{-+}{-} y$" which is equivalent to the edge "$y \overset{+-}{-} x$".

Formally, let $S = \{+, -\}$ be a set of signs. A graph with labelled edges on a set S is a pair (G, ψ), where $G = (V, E)$ is a graph, and ψ is a function with domain E and range S. $\psi(e)$ is called the label of the edge $e \in E$. Let $G = (V, E, \psi)$ be a signed primal graph with labelled edges on SxS. Let x and y be vertices in V, if $e = \{x, y\}$ is an edge and $\psi(e) = (s, s')$, then $s(resp.s')$ is called the adjacent sign to $x(resp.y)$. A 2-CNF F is a path, cycle, or a tree if its signed primal graph G_F represents a path, cycle, or a tree, respectively. We will omit the signs on the graph if all of them are $+$.

Notice that a signed primal graph of a 2-CNF can be a multigraph since two fixed variables can be involved in more than one clause of the formula, forming so parallel edges. Furthermore, a unitary clause is represented by a loop (an edge to join a vertex to itself). A polynomial time algorithm to process parallel edges and loops to solve #SAT has been shown in [10] and is presented in Section 3 for completeness.

Let $\rho : \text{2-CNF} \to G_F$ be the function whose domain is the space of Boolean formulae in 2-CNF and codomain is the set of multi-graphs, ρ is a bijection. So any 2-CNF formula has a unique signed constraint graph associated via ρ and viceversa, any signed constraint graph G_F has a unique formula associated.

3 Computing #2SAT According to the Topology of the signed primal graph

In this section we briefly summarize the main results already reported at [10] for completeness. We describe simple topologies on a graph representing a 2-CNF formula F and how is computed the value #2SAT(F). We begin with simple topologies as acyclic graphs.

Let f_i be a family of clauses of the formula F built as follows: $f_1 = \emptyset; f_i = \{C_j\}_{j<i}$, $i \in [\![m]\!]$. Let $SAT(f_i) = \{s : s \text{ satisfies } f_i\}$, $A_i = \{s \in SAT(f_i) : x_i^1 \in s\}$, $B_i = \{s \in SAT(f_i) : x_i^0 \in s\}$. Let $\alpha_i = |A_i|$; $\beta_i = |B_i|$ and $\mu_i = |SAT(f_i)| = \alpha_i + \beta_i$.

For every vertex $x \in G_F$ a pair (α_x, β_x) is computed, where α_x indicates how many times the variable x can take the value 'true' and β_x the number of times x can take value 'false' into the F model's set.

Path Case Notice that if F is a path $n = |\upsilon(F)| = m+1$, $f_i \subset f_{i+1}$, $i \in [\![m-1]\!]$.

$$F = \{C_1, C_2, \ldots, C_m\} = \{\{x_1^{\epsilon_1}, x_2^{\delta_1}\}, \{x_2^{\epsilon_2}, x_3^{\delta_2}\}, \ldots, \{x_m^{\epsilon_m}, x_{m+1}^{\delta_m}\}\},$$

where $\delta_i, \epsilon_i \in \{0,1\},\ i \in [\![m]\!]$. The pairs (α_i, β_i) associated to each vertex x_i, $i = 2, \ldots, m$ are computed according to the signs (ϵ_i, δ_i) of the literals in the clause c_i by the following recurrence equation:

$$(\alpha_i, \beta_i) = \begin{cases} (\beta_{i-1} & ,\alpha_{i-1} + \beta_{i-1}) \text{ if } (\epsilon_i, \delta_i) = (-,-) \\ (\alpha_{i-1} + \beta_{i-1} & ,\beta_{i-1}) \text{ if } (\epsilon_i, \delta_i) = (-,+) \\ (\alpha_{i-1} & ,\alpha_{i-1} + \beta_{i-1}) \text{ if } (\epsilon_i, \delta_i) = (+,-) \\ (\alpha_{i-1} + \beta_{i-1} & ,\alpha_{i-1}) \text{ if } (\epsilon_i, \delta_i) = (+,+) \end{cases} \tag{1}$$

The first pair is $(\alpha_1, \beta_1) = (1, 1)$ since x_1 can be true or false for satisfying f_1.

Note that as $F = f_m$ then $\#SAT(F) = \mu_m = \alpha_m + \beta_m$.

Parallel edges case Suppose, that two clauses are $c_k = (x_{i-1}^{\epsilon_k}, x_i^{\delta_k})$ and $c_j = (x_{i-1}^{\epsilon_j}, x_i^{\delta_j})$, which involve variables x_{i-1} and x_i. Then, we compute the values for (α_i, β_i) associated to the node x_i, according to the signs (ϵ_k, δ_k) and (ϵ_j, δ_j) as:

$$(\alpha_i, \beta_i) = \begin{cases} (\alpha_{i-1},\alpha_{i-1}) \text{ if } (\epsilon_k, \delta_k) = (1,1) \textit{ and } (\epsilon_j, \delta_j) = (1,0) \\ (\mu_{i-1},0\ \ ) \text{ if } (\epsilon_k, \delta_k) = (1,1) \textit{ and } (\epsilon_j, \delta_j) = (0,1) \\ (\beta_{i-1},\alpha_{i-1}) \text{ if } (\epsilon_k, \delta_k) = (1,1) \textit{ and } (\epsilon_j, \delta_j) = (0,0) \\ (\alpha_{i-1},\beta_{i-1}) \text{ if } (\epsilon_k, \delta_k) = (1,0) \textit{ and } (\epsilon_j, \delta_j) = (0,1) \\ (0\ \ ,\mu_{i-1}) \text{ if } (\epsilon_k, \delta_k) = (1,0) \textit{ and } (\epsilon_j, \delta_j) = (0,0) \\ (\beta_{i-1},\beta_{i-1}) \text{ if } (\epsilon_k, \delta_k) = (0,1) \textit{ and } (\epsilon_j, \delta_j) = (0,0) \end{cases} \tag{2}$$

Let F be a 2-CNF such that three clauses in F involve the same variables, then the value of (α_i, β_i) is computed by recurrence (3).

$$(\alpha_i, \beta_i) = \begin{cases} (0\ \ ,\alpha_{i-1}) \text{ if } \{(x_{i-1}, x_i),(x_{i-1}, \overline{x}_i),(\overline{x}_{i-1}, \overline{x}_i)\} \subseteq F \\ (\mu_{i-1},0\ \ ) \text{ if } \{(x_{i-1}, x_i),(x_{i-1}, \overline{x}_i),(\overline{x}_{i-1}, x_i)\} \subseteq F \\ (\beta_{i-1},\alpha_{i-1}) \text{ if } \{(x_{i-1}, x_i),(\overline{x}_{i-1}, x_i),(\overline{x}_{i-1}, \overline{x}_i)\} \subseteq F \\ (\alpha_{i-1},\beta_{i-1}) \text{ if } \{(\overline{x}_{i-1}, x_i),(x_{i-1}, \overline{x}_i),(\overline{x}_{i-1}, \overline{x}_i)\} \subseteq F \end{cases} \tag{3}$$

Of course, four parallel edges among the same endpoints indicate that the 2-CNF F is unsatisfiable and then $\#2\text{SAT}(F) = 0$.

Unitary Clauses Case. A unitary clause represents a loop in the signed primal graph of a 2-CNF. When (α_i, β_i) is computed over a node x_i which has a loop edge, recurrence (4) is applied.

$$(\alpha_i, \beta_i) = \begin{cases} (0, \beta_i) & \text{if } (x_i^0) \in U \\ (\alpha_i, 0) & \text{if } (x_i^1) \in U \end{cases} \tag{4}$$

Since an unitary clause uniquely determines the value of its variable. Furthermore, when both $(x_i^1) \in U$ and $(x_i^0) \in U$ then the original formula is unsatisfiable. Both of them parallel edges and unitary clauses can be considered

in a pre-processing step of the formula before to apply the general algorithm presented in section 5.

Acyclic Graphs Case Let F be a 2-CNF formula where its associated signed primal graph G_F is acyclic, which may contain loops and parallel edges, then we can assume G_F as a rooted tree, a traversal of the graph allows to built a rooted tree. A tree has three kinds of nodes: a root node, interior nodes and leaf nodes. We denote with (α_v, β_v) the pair associated with the node v ($v \in G_F$). We compute $\#SAT(F)$ while we are traversing G_F in post-order with the following algorithm.

Algorithm Count_Models_for_trees_loops_parallel(G_F**)**
Input: G_F - a tree graph which may contain parallel edges and loops.
Output: The number of models of F
Procedure:
Traversing G_F in post-order, and when a node $v \in G_F$ is visit, assign:

1. $(\alpha_v, \beta_v) = (1, 1)$ if v is a leaf node in G_F.
2. If v is a parent node with a list of child nodes associated, i.e., $u_1, u_2, ..., u_k$ are the child nodes of v, as we have already visited all child nodes, then each pair $(\alpha_{u_j}, \beta_{u_j})$ $j = 1, ..., k$ has been determined. Let $e_1 = v^{\epsilon_1}u_1^{\delta_1}, e_2 = v^{\epsilon_2}u_2^{\delta_2}, \ldots, e_k = v^{\epsilon_k}u_k^{\delta_k}$ be the edges connecting v with each of its child nodes. A pair $(\alpha_{e_j}, \beta_{e_j})$ is computed for each edge e_j based on recurrence (1) where α_{e_j-1} is α_{u_j} and β_{e_j-1} is β_{u_j} for $j = 1, \ldots k$. Then, let $\alpha_v = \prod_{j=1}^{k} \alpha_{e_j} + \beta_v$ and $\beta_v = \prod_{j=1}^{k} \beta_{e_j}$. Notice that this step includes the case when v has just one child node.
3. if v has parallel edges apply recurrence (2) or (3).
4. if v has a loop apply recurrence (4)
5. If v is the root node of G_F then return $(\alpha_v + \beta_v)$.

This procedure returns the number of models for F in time $O(n + m)$ which is the necessary time for traversing G_F in post-order.

Simple Cycles Case. Let $C_n = (V, E)$ be a simple cycle graph, so $|V| = n = |E| = m$, i.e. every vertex in V has degree two. We decompose the cycle C_n as: $P_n \cup \{v_1^{\epsilon_1}, v_n^{\epsilon_2}\}$, where $P_n = (V, E')$, $E' = \{c_1, ..., c_{n-1}\}$. P_n is the internal path of the cycle, and $c_n = \{v_1^{\epsilon_1}, v_n^{\epsilon_2}\}$ is called the cycle *frond edge*.

In order to process the number of models of C_n based on the P_n models, two *computing threads* or just *threads* are used. A computing thread is a sequence of pairs $(\alpha_i, \beta_i), i = 1, \ldots, n$ used for counting the number of models on a path graph. One thread, called the main thread L_m, is used for computing $\#2SAT(P_n)$. Since L_m counts the assignments where both $v_1^{\epsilon_1-1}$ and $v_1^{\epsilon_2-1}$ appear, they need to be removed from the models of C_n due to $c_n = \{v_1^{\epsilon_1}, v_n^{\epsilon_2}\}$ belongs to C_n, so a second thread L_s is used for computing $|\{s \in \#2SAT(P_n) : v_1^{\epsilon_1-1} \in s \wedge v_n^{\epsilon_2-1} \in s\}|$, e.g. the number of assignments where both $v_1^{\epsilon_1-1}$ and $v_1^{\epsilon_2-1}$ appear, these assignments will be subtracted from $\#2SAT(P_n)$ in orden to compute $\#2SAT(C_n)$. The main thread L_m begins with the pair

$(\alpha_1, \beta_1) = (1, 1)$, since the models where both v_1^0 and v_1^1 appear are counted. On the other hand L_s begins with the pair $(\alpha_1, \beta_1) = (0, 1)$ since just the assignments where $v_1^{\epsilon_1 - 1}$ appears need to be counted. If the last pair of L_s is (α'_n, β'_n), just the value for β'_n is considered since β'_n holds the number of assignments where $v_n^{\epsilon_2 - 1}$ appears. Therefore, if $L_m = (\alpha_n, \beta_n)$ and $L_s = (\alpha'_n, \beta'_n)$ then $\#2SAT(C_n) = \alpha_n + \beta_n - \beta'_n$. The time complexity of this procedure is $O(2m)$ where m is the number of clauses of the cycle.

4 Counting #2SAT for Cactus Formulas

A Spanish version of the algorithm for cactus graphs has been submitted to IEEE Transactions on Latin America, the submitted version can be consulted at (http://arxiv.org/abs/1702.08581). For that here we highlight the main idea. A cactus formula is one whose signed primal graph is cactus. A cactus graph is one where each pair of simple cycles intersect in at most one node, e.g each pair of cycles are independent or have a singleton intersection. The algorithm for counting models in cactus graphs combines the procedures described in section 3. The steps of the procedure and its description is presented below.

1. Store the input formula in a dynamically allocation table.
2. Built a spanning tree using a depth first search strategy.
3. Count over the tree in a postorder traversal considering the edges that form simple cycles.

Dynamically Allocation Table. The dynamically allocation table is built as follows. Let x_i be a variable of the formula F. The index of the i-th row represents the variable x_i. For each clause $\{x_i^{\epsilon_1}, x_j^{\epsilon_2}\}$ a triple $(j, \epsilon_1, \epsilon_2)$ is stored at the i-th row. A mirror table is also built in order to speed up the tree construction. The second table stores the triple $(i, \epsilon_1, \epsilon_2)$ at the j-th row.

Spanning Tree Construction. From the tables, tuples of the form (Tree, Cotree) are built. If the input formula can be decomposed in more than one tuple (Tree, Cotree), it means that the input formula has several components (subformulas which do not have common variables), hence the number of models of the input formula can be computed using the product of the models of its subformulas. The indices of the tables are visited in order to build a node of the spanning tree, if during the traversal an edge which forms a simple cycle is found this is stored in the Cotree structure. Additionally, the fact that a literal or its negations appears in a clause is stored in the node of the tree (cotree).

Counting over the tree and its cotree. The correctness of counting over the (Tree, Cotree) structure has been reported in [12]. In summary, to each vertex x_i of the tree, a tuple (α_i, β_i) is associated where α_i denotes the number of models where x_i is true and β_i the number of models where β_i is false relative to the

vertices already visited in the tree. When a vertex x_i which belongs to an edge $e = \{x_i, x_j\}$ of the cotree is reached, a second pair $(\alpha_{i_2}, \beta_{i_2})$ is associated to each vertex until reaching x_j in the tree. Since the graph is cactus, at most two pairs can co-exists during the execution of the algorithm. A postorder traversal over the tree computing the pairs gives the number of models of the input formula [1].

To each leaf of the tree the pair $(1, 0)$ is associated, which is the initial value if the formula is considered as a single literal. If the leaf starts a fundamental cycle the pair (0,1) is also counted. When an interior node is visited, a clause $\{x^{\epsilon_1}, y^{\epsilon_1}\}$ has been considered, hence the models are computed according to recurrence (1).

Both the allocation table construction and the tree construction takes m steps each one. The counting procedure traverse also each edge of the tree, hence the time complexity of the algorithm is $O(3m)$ which is linear according to the number of clauses.

5 Counting for General 2-CNF formulas

Schmitt et al. [6] algorithm works as follows: let F be a boolean formula, choose at random a clause $c = \{x_i^{\epsilon_1}, x_j^{\epsilon_2}\} \in F$, in order to satisfy F either $x_i^{\epsilon_1}$ or $x_j^{\epsilon_2}$ must be true. Hence three possible cases can be derived in orden to satisfy c, ($s(x_i^{\epsilon_1}) = 1$ and $s(x_j^{\epsilon_2}) = 1$) or ($s(x_i^{\epsilon_1}) = 1$ and $s(x_j^{\epsilon_2}) = 0$) or ($s(x_i^{\epsilon_1}) = 0$ and $s(x_j^{\epsilon_2}) = 1$).

Each case generates a subformula F_k of F, $k = 1, 2, 3$ where Boolean values for the variables $x_i^{\epsilon_1}$ and $x_j^{\epsilon_2}$ can be fixed for reducing the input formula. This process is iterated until some criteria are met in order to decide the number of models of each subformula. In the method presented in this paper (called markSAT), two variants of Schmith algorithm are considered. Fisrtly, the clause chosen to decompose the input formula is one that when transformed to its signed primal graph, forms an intersected cycle, e.g. a witness that the graph is not cactus. Secondly, our process is iterated until cactus subgraphs are obtained.

To make a comparison we built formulas with a range of 100 to 6500 variables and between 2,495 and 21,121,750 clauses. The standard format used to store the formulas was DIMACS (http://logic.pdmi.ras.ru/ basolver/dimacs.html). Table 1 shows the time comparisons between markSAT and sharpSAT methods. Clauses that represented parallel edges in the graph can be counted by our cactus procedure as well as unitary clauses, hence complete graphs have the maximum number of cycles (instance 1 in Table 1). We reduce the number of edges (clauses) by a factor of 10% in each execution. In column (4) of Table 1 we report graphs (formulas) with 50% of edges (clauses) with respect to a complete graph (Instance 2). Each formula was randomly created. As it can be analyzed in fifteen of the sixteen cases the execution time of our proposal overcomes sharpSAT.

Additionally, we tested our implementation with eighty different instances compared against sharpSAT. We begin with a formula which represents a complete graph and we removed 10% of the clauses in each execution. The name of the Instance XXX_YY in column 1 of Tables 2 and 3 means that there are XXX

				Time in Seconds			
	Variables	Clauses		sharpSAT		markSAT	
		Instance 1	Instance 2	Instance 1	Instance 2	Instance 1	Instance 2
1	100	4,950	2,475	0.031	0.028	0.011	0.006
2	200	19,900	9,950	0.051	0.038	0.042	0.022
3	500	124,750	62,375	**0.283**	0.151	0.285	0.143
4	1,000	499,500	249,750	1.674	0.834	1.166	0.597
5	2,000	1,999,000	999,500	11.889	5.786	4.850	2.445
6	5,000	12,497,500	6,248,750	171.442	82.970	30.870	15.558
7	6,000	17,997,000	8,998,500	293.078	141.755	49.694	22.752
8	6,500	21,121,750	10,560,875	371.091	179.876	58.447	26.448

Table 1. Results of comparing markSAT against sharpSAT.

number of variables with YY% number of clauses with respect to $n * (n-1)/2$ where $n = |\text{XXX}|$ is the number of variables of the formula. Fig. 1

Our implementation, the instances and the comparison table with the eighty formulas can be downloaded at (http://sc.uaemex.mx/rmarcial/markSAT).

6 Conclusions

We have presented an implementation for counting models of 2-CNF formulas. Our implementation combines two procedures, the first decomposes the input formula choosing a clause and the second counts on cactus graphs (formulas). We show that our implementation overcome sharpSAT, the leading sequential implementation in the literature. In fact the time complexity in the worst case remains as the one established by Schmitt et al. [6].

We thanks the Universidad Autónoma del Estado de México for the sponsorship under the project 4315/2017/CI.

References

1. J. A. Bondy and U.S.R. Murty. Graph Theory. Springer, 3rd printing 2008 edition.
2. Roberto J. Bayardo, Jr. and Robert C. Schrag. Using csp look-back techniques to solve real-world sat instances. In *Proceedings of AAAI and IAAI*, AAAI'97/IAAI'97, pages 203–208. AAAI Press, 1997.
3. Graham Brightwell and Peter Winkler. Counting linear extensions. *Order*, 8(3):225–242, 1991.
4. Jan Burchard, Tobias Schubert, and Bernd Becker. *Laissez-Faire Caching for Parallel #SAT Solving*, pages 46–61. Springer International Publishing, Cham, 2015.
5. Daniel Paulusma, Friedrich Slivovsky, and Stefan Szeider. Model counting for cnf formulas of bounded modular treewidth. *Algorithmica*, 76(1):168–194, 2016.
6. Manuel Schmitt and Rolf Wanka. Exploiting independent subformulas: A faster approximation scheme for #k-sat. *Inf. Process. Lett.*, 113(9):337–344, May 2013.

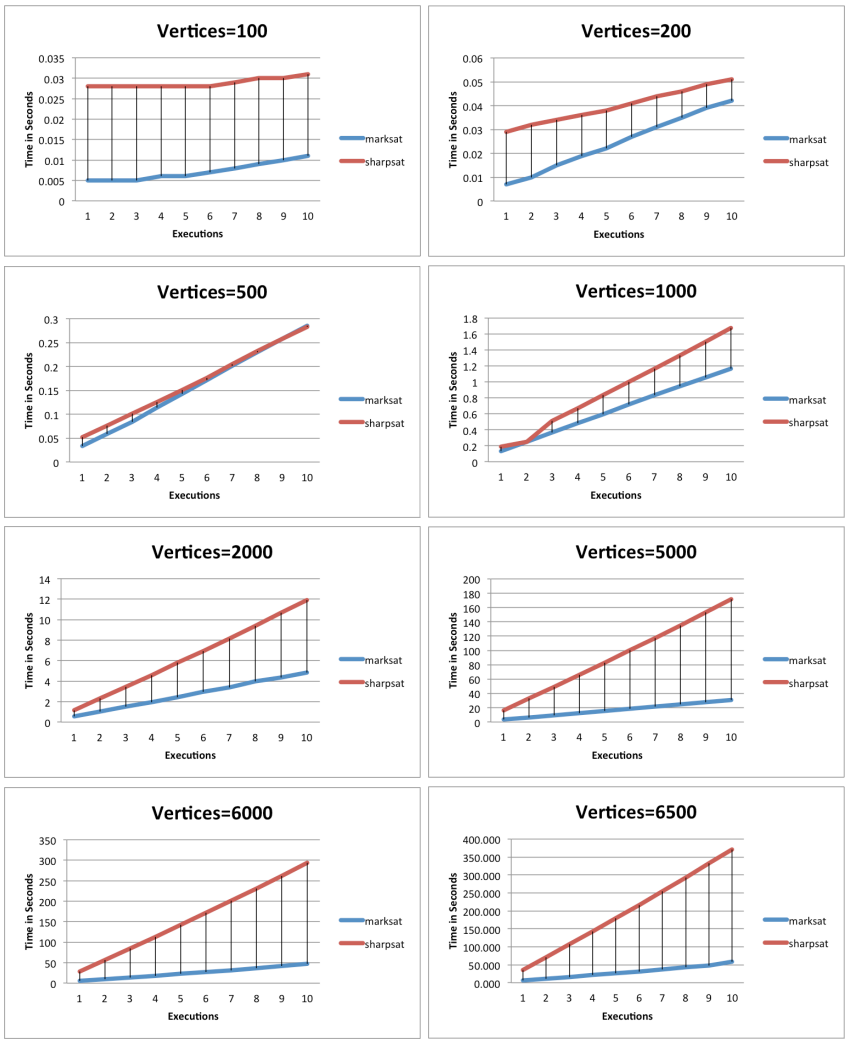


Fig. 1. Time comparison between sharpSAT and markSAT. From left to right and top to bottom number of vertices 100, 200, 500, 1000, 2000, 5000, 6000 and 6500 respectively. Each graphic denotes 10 executions with different number of edges, as described in Tables 2 and 3.

7. Stefan Szeider. *On Fixed-Parameter Tractable Parameterizations of SAT*, pages 188–202. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.
8. Marc Thurley. *sharpSAT – Counting Models with Advanced Component Caching and Implicit BCP*, pages 424–429. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.
9. Magnus Wahlström. *A Tighter Bound for Counting Max-Weight Solutions to 2SAT Instances*, pages 202–213. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.
10. Guillermo De Ita, Pedro Bello, and Meliza Contreras. New polynomial classes for #2sat established via graph-topological structure. *Engineering Letters*, 15:2, 2007.
11. Guillermo De Ita Luna. *Polynomial Classes of Boolean Formulas for Computing the Degree of Belief*, pages 430–440. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.
12. J. Raymundo Marcial-Romero, Guillermo De Ita Luna, J. Antonio Hernández, and Rosa María Valdovinos. *A Parametric Polynomial Deterministic Algorithm for #2SAT*, pages 202–213. Springer International Publishing, Cham, 2015.

			Time in Seconds	
Instance	Variables	Clauses	sharpSAT	markSAT
100_10	100	495	0.028	0.005
100_20	100	990	0.028	0.005
100_30	100	1,485	0.028	0.005
100_40	100	1,980	0.028	0.006
100_50	100	2,475	0.028	0.006
100_60	100	2,970	0.028	0.007
100_70	100	3,465	0.029	0.008
100_80	100	3,960	0.030	0.009
100_90	100	4,455	0.030	0.010
100_100	100	4950	0.031	0.011
200_10	200	1,990	0.029	0.007
200_20	200	3,980	0.032	0.010
200_30	200	5,970	0.034	0.015
200_40	200	7,960	0.036	0.019
200_50	200	9,950	0.038	0.022
200_60	200	11,940	0.041	0.027
200_70	200	13,930	0.044	0.031
200_80	200	15,920	0.046	0.035
200_90	200	17,910	0.049	0.039
200_100	200	19,900	0.051	0.042
500_10	500	12,475	0.052	0.033
500_20	500	24,950	0.076	0.059
500_30	500	37,425	0.101	0.084
500_40	500	49,900	0.125	0.114
500_50	500	62,375	0.151	0.143
500_60	500	74,850	0.176	0.171
500_70	500	87,325	0.205	0.201
500_80	500	99,800	0.232	0.230
500_90	500	112,275	0.258	0.257
500_100	500	124,750	0.283	0.285
1000_10	1,000	49,950	0.189	0.131
1000_20	1,000	99,900	0.247	0.250
1000_30	1,000	149,850	0.512	0.368
1000_40	1,000	199,800	0.670	0.483
1000_50	1,000	249,750	0.834	0.597
1000_60	1,000	299,700	1.003	0.717
1000_70	1,000	349,650	1.167	0.832
1000_80	1,000	399,600	1.331	0.944
1000_90	1,000	449,550	1.507	1.055
1000_100	1,000	499,500	1.674	1.166

Table 2. Formulas with 100, 200, 500 and 1000 variables. The number of clauses goes from 495 to 499,500.

			Time in Seconds	
Instance	Variables	Clauses	sharpSAT	markSAT
2000_10	2,000	199,900	1.162	0.558
2000_20	2,000	399,800	2.298	1.023
2000_30	2,000	599,700	3.465	1.491
2000_40	2,000	799,600	4.594	1.963
2000_50	2,000	999,500	5.786	2.445
2000_60	2,000	1,199,400	6.940	2.941
2000_70	2,000	1,399,300	8.172	3.381
2000_80	2,000	1,599,200	9.375	3.959
2000_90	2,000	1,799,100	10.666	4.352
2000_100	2,000	1,999,000	11.889	4.850
5000_10	5,000	1,249,750	16.314	3.614
5000_20	5,000	2,499,500	32.796	6.496
5000_30	5,000	3,749,250	49.208	9.403
5000_40	5,000	4,999,000	66.069	12.495
5000_50	5,000	6,248,750	82.970	15.558
5000_60	5,000	7,498,500	100.173	18.631
5000_70	5,000	8,748,250	117.360	21.662
5000_80	5,000	9,998,000	134.945	24.827
5000_90	5,000	11,247,750	153.102	27.760
5000_100	5,000	12,497,500	171.442	30.870
6000_10	6,000	1,799,700	27.923	5.248
6000_20	6,000	3,599,400	56.106	9.404
6000_30	6,000	5,399,100	84.246	13.693
6000_40	6,000	7,198,800	112.913	18.087
6000_50	6,000	8,998,500	141.755	22.752
6000_60	6,000	10,798,200	171.544	26.847
6000_70	6,000	12,597,900	200.839	31.258
6000_80	6,000	14,397,600	230.728	35.959
6000_90	6,000	16,197,300	261.389	41.961
6000_100	6,000	17,997,000	293.078	46.694
6500_10	6,500	2,112,175	35.212	6.069
6500_20	6,500	4,224,350	70.990	11.053
6500_30	6,500	6,336,525	106.997	16.075
6500_40	6,500	8,448,700	142.817	21.254
6500_50	6,500	10,560,875	179.876	26.488
6500_60	6,500	12,673,050	216.774	31.536
6500_70	6,500	14,785,225	254.181	37.294
6500_80	6,500	16,897,400	292.413	42,579
6500_90	6,500	19,009,575	331.912	47.816
6500_100	6,500	21,121,750	371.091	58.447

Table 3. Formulas with 2,000, 5,000, 6,000 and 6,500 variables. The number of clauses goes from 199,900 to 21,121,750.

3.3. A linear time algorithm for computing #2SAT for outerplanar 2-CNF Formulas

Este artículo fue enviado y aceptado en el MCPR 2018, Publicado en Lecture Notes in Computer Science Vol. 10880 (DOI 10.1007/978-3-319-92198-3, Softcover ISBN 978-3-319-92197-6).

A linear time algorithm for computing #2SAT for Outerplanar 2-CNF Formulas

Marco A. López[1], J. Raymundo Marcial-Romero[1], Guillermo De Ita[2], and Yolanda Moyao[2]

[1] Facultad de Ingeniería, UAEM
[2] Facultad de Ciencias de la Computación BUAP
mlopezm158@alumno.uaemex.mx, jrmarcialr@uaemex.mx, deita@cs.buap.mx, ymoyao@cs.buap.mx

Abstract. Although the satisfiability problem for two Conjunctive Normal Form formulas (2SAT) is polynomial time solvable, it is well known that #2SAT, the counting version of 2SAT is #P-Complete. However, it has been shown that for certain classes of formulas, #2SAT can be computed in polynomial time. In this paper we show another class of formulas for which #2SAT can also be computed in lineal time, the so called outerplanar formulas, e.g. formulas whose signed primal graph is outerplanar. Our algorithm's time complexity is given by $O(n+m)$ where n is the number of variables and m the number of clauses of the formula.

1 Introduction

#SAT (the problem of counting models for a Boolean formula) is of special concern to Artificial Intelligence (AI), and it has a direct relationship to Automated Theorem Proving, as well as to approximate reasoning [1–3]. #SAT can be reduced to several different problems in approximate reasoning. For example, in the cases of: estimating the degree of belief in propositional theories, the generation of explanations to propositional queries, repairing inconsistent databases, in Bayesian inference, in a truth maintenance systems [2, 4, 3]. The previous problems come from several AI applications such as planning, expert systems, approximate reasoning, etc.

#SAT is #P-complete, even for formulas in two conjunctive normal form, so for complete methods, only exponential-time algorithms are known. The exact algorithm with the best bound until now was presented by Wahlström [5], who provides an $O(1.2377^n)$-time algorithm, where n is the number of variables of the formula. There are also randomized algorithms, in this direction, the algorithm of Dantsin and Wolpert [6] has the best bound $O(1.3238^n)$. #SAT appears to be harder than SAT since 2SAT is polynomial time solvable while #2SAT is #P-complete. However, it has been shown that for some classes of formulas, #2SAT can be computed in polynomial time [1]. The most relevant cases are monotone formulas and cactus formulas [7].

In this paper we present a new class of formulas for which #2SAT can be computed in polynomial time. We call these formulas outerplanar since their

representation via primal graphs provide outerplanar graphs. A graph is outerplanar if it has a crossing-free embedding in the plane such that all vertices are on the same face. The outerplanar graphs contain as subsets to the cactus graphs which are used in the theory of condensation in statistical mechanics and as simplified models of real lattices. Cactus graphs have also found applications in the theory of electrical and communication networks [7, 8].

Another special class of graphs contained into outerplanar graphs is the class of polygonal array graphs that has been widely used in mathematical chemistry, since they are molecular graphs used to represent the structural formula of chemical compounds. In particular, hexagonal arrays are the graph representations of an important subclass of benzenoid molecules, unbranched catacondensed benzenoid molecules, which play a distinguished role in the theoretical chemistry of benzenoid hydrocarbons [9, 10].

In our case, we are more interested in the application of counting models on conjunctive normal form formulas as a medium to develop methods for approximate reasoning. For example, for computing the degree of belief on propositional formulas, or for building Bayesian models. It is relevant to know how many models are maintained while input conjunctive normal form formulas are being updating [2, 4, 6].

Our method firstly decompose the input conjunctive normal form formula to its signed primal graph, secondly a treewidth decomposition of the graph is computed, outerplanar graphs have treewidth at most 2 so a linear time algorithm can be used. Finally, a procedure that uses macros (acumulative basic operations) is applied on the nodes of the treewidth.

The paper is organized as follows, in Section 2 the preliminaries are established. In Section 3 a treewidth decomposition of outerplanar formulas is presented. In Section 4, our main procedure is presented and finally, the Conclusion.

2 Preliminaries

Let $X = \{x_1, \ldots, x_n\}$ be a set of n Boolean variables. A literal is either a variable x_i or a negated variable $\overline{x}_i$. As usual, for each $x_i \in X$, we write $x_i^0 = \overline{x}_i$ and $x_i^1 = x_i$. A clause is a disjunction of different literals (sometimes, we also consider a clause as a set of literals). For $k \in N$, a k-clause is a clause consisting of exactly k literals and, a $(\leq k)$-clause is a clause with at most k literals. A variable $x \in X$ appears in a clause c if either the literal x^1 or x^0 is an element of c.

A Conjunctive Normal Form (CNF) F is a conjunction of clauses (we also call F a Conjunctive Form). A k-CNF is a CNF containing clauses with at most k literals.

We use $\nu(Y)$ to express the set of variables involved in the object Y, where Y could be a literal, a clause or a Boolean formula. $Lit(F)$ is the set of literals which appear in a CNF F, i.e. if $X = \nu(F)$, then $Lit(F) = X \cup \overline{X} = \{x_1^1, x_1^0, \ldots, x_n^1, x_n^0\}$. We also denote $\{1, 2, \ldots, n\}$ by $[[n]]$.

An assignment s for F is a Boolean function $s : \nu(F) \to \{0, 1\}$. An assignment can be also considered as a set which does not contain complementary literals.

If $x^\epsilon \in s$, being s an assignment, then s turns x^ϵ true and $x^{1-\epsilon}$ false, $\epsilon \in \{0,1\}$. Considering a clause c and assignment s as a set of literals, c is satisfied by s if and only if $c \cap s \neq \emptyset$, and if for all $x^\epsilon \in c$, $x^{1-\epsilon} \in s$ then s falsifies c.

If $F_1 \subset F$ is a formula consisting of some clauses of F, then $\nu(F_1) \subset \nu(F)$, and an assignment over $\nu(F_1)$ is a partial assignment over $\nu(F)$.

Let F be a Boolean formula in CNF, F is satisfied by an assignment s if each clause in F is satisfied by s. F is contradicted by s if any clause in F is contradicted by s. A model of F is an assignment for $\nu(F)$ that satisfies F. We will denote as $SAT(F)$ the set of models for the formula F.

Given a CNF F, the SAT problem consists on determining if F has a model. The #SAT problem consists of counting the number of models of F defined over $\nu(F)$. #2-SAT denotes #SAT for formulas in 2-CNF.

2.1 The signed primal graph of a 2-CF

There are some graphical representations of a CNF (see e.g. [11]), we use here the signed primal graph of a two conjunctive normal form.

Let F be a 2-CNF, its signed primal graph (constraint graph) is denoted by $G_F = (V(F), E(F))$, with $V(F) = \nu(F)$ and $E(F) = \{\{\nu(x), \nu(y)\} : \{x, y\} \in F\}$, that is, the vertices of G_F are the variables of F, and for each clause $\{x, y\}$ in F there is an edge $\{\nu(x), \nu(y)\} \in E(F)$. For $x \in V(F)$, $\delta(x)$ denotes its degree, i.e. the number of incident edges to x. Each edge $c = \{\nu(x), \nu(y)\} \in E$ is associated with an ordered pair (s_1, s_2) of signs, assigned as labels of the edge connecting the literals appearing in the clause. The signs s_1 and s_2 are related to the literals x^ϵ and y^δ, respectively. For example, the clause $\{x^0, y^1\}$ determines the labelled edge: "$x \overset{-+}{-} y$" which is equivalent to the edge "$y \overset{+-}{-} x$".

Formally, let $S = \{+, -\}$ be a set of signs. A graph with labelled edges on a set S is a pair (G, ψ), where $G = (V, E)$ is a graph, and ψ is a function with domain E and range S. $\psi(e)$ is called the label of the edge $e \in E$. Let $G = (V, E, \psi)$ be a signed primal graph with labelled edges on SxS. Let x and y be vertices in V, if $e = \{x, y\}$ is an edge and $\psi(e) = (s, s')$, then $s(resp.s')$ is called the adjacent sign to $x(resp.y)$. We say that a 2-CNF F is a path, cycle, a tree, or an outerplanar graph, if its signed constraint graph G_F represents a path, cycle, a tree, an outerplanar graph, respectively. We will omit the signs on the graph if all of them are $+$.

Notice that a signed primal graph of a 2-CNF can be a multigraph since two fixed variables can be involved in more than one clause of the formula forming so parallel edges. Furthermore, a unitary clause is represented by a loop (an edge to join a vertex to itself). A polynomial time algorithm to process parallel edges and loops to solve #SAT has been shown in [1].

Let ρ : 2-CNF $\rightarrow G_F$ be the function whose domain is the space of Boolean formulas in 2-CNF and codomain the set of multi-graphs, ρ is a bijection. So any 2-CNF formula has a unique signed constraint graph associated via ρ and viceversa, any signed constraint graph G_F has a unique formula associated.

3 Outerplanar 2-CNF Formulas

An outerplanar 2-CNF formula is one whose signed primal graph is ourterplanar e.g the graph has a planar drawing for which all vertices belong to the outer face of the drawing. Outerplanar graphs may be characterized (analogously to Wagner's theorem for planar graphs) by the two forbidden minors K_4 and $K_{2,3}$, or by their Colin de Verdière graph invariants. They have Hamiltonian cycles if and only if they are biconnected, in which case the outer face forms the unique Hamiltonian cycle. Every outerplanar graph is 3-colorable, and has degeneracy and treewidth at most 2 [12]. The outerplanar graphs are a subset of the planar graphs, of the serial-parallel graphs, and of the circle graphs.

3.1 Tree decomposition

Many hard problems can even be solved efficiently on graphs that might not be trees, but are in some sense still sufficiently treelike. A formal parameter that is widely accepted to measure this likeliness is the treewidth of a graph [13].

Treewidth is one of the most basic parameters in graph algorithms. There is a well established theory on the design of polynomial (or even linear) time algorithms for many intractable problems where their input is restricted to graphs of bounded treewidth. More importantly, there are problems on graphs with n vertices and treewidth at most k that can be solved in time $O(c^k \cdot n^{O(1)})$, where c is a problem dependent constant [14].

For example, a maximum independent set (a MIS) of a graph can be found in time $O(2^k \cdot n)$, given a tree decomposition of width at most k. Therefore, a quite natural approach to compute $i(G)$ would be to find a treewidth T_G of G, and to determine how to join the partial results on the nodes of T_G. However, for any general graph G, finding its minimum treewidth is a NP-complete problem.

A tree decomposition of a graph $G = (V, E)$ is a pair $(\{X_i \mid i \in I\}, T = (I, F))$ with $\{X_i \mid i \in I\}$ a collection of subsets of V, called bags, and $T = (I, F)$ a tree, such that for all $v \in V$ there exists an $i \in I$ with $v \in X_i$, for all $\{v, w\} \in E$ there exists and $i \in I$ with $v, w \in X_i$, and for all $v \in V$, the set $I_v = \{i \in I \mid v \in X_i\}$ forms a connected subgraph (subtree of T).

The width of a tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ is defined as $\max_{i \in I} |X_i| - 1$. The treewidth of a graph G is the minimum width of a tree decomposition of G. It is NP-complete to decide whether the treewidth of a graph is at most k (if k is part of the input) [15]. But, for every fixed k, there is a linear-time algorithm deciding wheter the treewidth is at most k, and when that is the case, producing a corresponding tree decomposition [12]. Algorithm 1 computes the 2-treewidth decomposition of an outerplanar graph [16].

Algorithm 1 keeps the structure of a tree, and then, we can combine the algorithms to be presented in the following section, in order to compute $i(T)$.

Algorithm 1 Procedure that computes the treewidth decomposition $(\{X_i \mid i \in I^*\}, T^*)$ of and outerplanar graph $G = (V, E)$.

```
1: procedure TREEWIDTH(G)
2: if |V| <= 3 then
3:    X_i = G
4:    I={i}
5:    return ({X_i}, T = (I, ∅))
6: else
7:    let (v ∈ V such that δ(v) = 2 or δ(v) = 1)
8:    ({X_i}, T = (I, F))=Treewidth((V − {v}, (E − {(v, w), (v, x)}) ∪ {(w, x)})
9:    let (i ∈ I with w ∈ X_i ∧ x ∈ X_i for some bag X_i)
10:   let (i* ∉ I)
11:   I* = I ∪ {i*}
12:   X_{i*} = {v, w, x}
13:   T* = (I*, F ∪ {(i, i*)})
14:   return ({X_i | i ∈ I*}, T*)
15: end if
```

4 Computing #2SAT on outerplanar 2-CNF formulas

If F consists of disconnected sub-formulas then $\#2SAT(F) = \prod_{i=1}^{k} \#2SAT(F_i)$ where $F_i, i = 1, \ldots, k$, are the disconnected sub-formulas of F [3]. The time complexity for computing $\#2SAT(F)$, denoted as $T(\#2SAT(F))$, is given by the rule $T(\#2SAT(F)) = max\{T(\#2SAT(F_i)) : F_i$ is a disconnected subformula of $F\}$. Thus, a first decomposition of the formula is done via its connected components, and from here on, we consider only outerplanar connected formulas.

Our algorithm is based on the following Theorem.

Theorem 1. *If the treewidth of a graph G is of size k, G has a tree decomposition where each bag is of size at most $k + 1$ [17].*

Due to the fact that the treewidth of outerplanar graphs is 2, there is a tree decomposition of outerplanar graphs where each bag has at most 3 vertices forming so trees, simple cycles or disconected components. Additionally each node (bag) is connected to its descedents or ancestor via a common edge or vertices.

We built a pair of linear equations whose solution represents the values (α, β) on each node of the tree decomposition.

Let B_1 and B_2 be two nodes in the tree decomposition of an outerplannar graph. Assume that B_2 is a child of B_1 in the tree decomposition. By definition of a tree decomposition, B_1 and B_2 share: a common edge or, a common vertex or, two non-adjacent vertices (either connected by a path or disconnected).

We present a procedure for computing $\#2SAT(F)$ traversing in postorder by the nodes B_i $1 \leq i \leq j$ of the tree decomposition. We begin computing the values (α, β) on the leaves, and later on, for interior nodes and finally, for the root node.

Fig. 1. On the left a constrained graph representing the Formula $\{\{x_1, x_2\}, \{x_1, x_3\}, \{x_2, x_4\}, \{x_2, x_3\}, \{x_3, x_4\}\}$. On the right, its tree decomposition.

The computation of $\#2SAT(B_j)$ when B_j is a leaf node can be done according to the procedures presented at [7], since it contains a path, a tree, a simple cycle or disconnected subgraphs. In case that a node contains disconnected subgraphs, the number of models is the product of the models of each subgraph.

The pair (α, β) for each leaf node is not represented by single numbers, instead this pair is represented by a pair of linear equations. Hence, the order of the computation is important since the pair of linear equations has to be associated to the vertex or edge that is joined to its father's node.

Let us show an example, consider the formula $F = \{\{x_1, x_2\}, \{x_1, x_3\}, \{x_2, x_4\}, \{x_2, x_3\}, \{x_3, x_4\}\}$ whose signed primal graph is shown on the left of Figure 1. A tree decomposition is shown on the right of Figure 1.

The leaf node B_2 contains a cycle, so according to the procedure presented at [7] two computing threads have to be computed. Since the edge (x_2, x_3) is the joint to its father's node. The computation should begins at one joint vertex and ends on the other. Lets us assume the computation begins at x_2. The two computing threads: L_p, L_c, and its associated pairs are expressed as basic operations between the symbolic variables: α and β, in the following way:

$$\begin{array}{lccccc} & x_2 & x_1 & x_3 & x_3 \to x_2 \\ L_p: & (\alpha,\beta) \to & (\alpha{+}\beta,\alpha) \to & (2\alpha+\beta,\alpha+\beta) \Rightarrow & (2\alpha+\beta,\alpha+\beta) \\ L_c: & (0,\beta) \to & (\beta,0) \to & (\beta,\beta) \Rightarrow & \text{- } (0,\beta) \\ & & & & \overline{\qquad\qquad\qquad}\text{-} \\ & & & & (2\alpha+\beta,\alpha) \end{array} \quad (1)$$

The symbol $\Rightarrow$ is used to establish that the first component of L_c is set to 0 since we are counting the assignments were both x_2^0 and x_3^0 do not appear.

Instead of evaluating the pair to get the models of B_2 we will associate to the edge (x_2, x_3) the pair of linear equations $(2\alpha + \beta, \alpha)$ which we call, the macro M_1.

A relevant property of a macro, as defined in this paper, is the possibility to represent cumulative operations via symbolic variables, making macros indistinguishable from individual operators. If subsequences of operators are repeated, a hierarchy of macros can represent a more compactly plan than a simple op-

erator sequence, replacing each occurrence of a repeating subsequence with a macro [18].

When the computation of $\#2SAT(B_1)$ starts, for example at vertex x_4, two new threads are created (since the bag contains a simple cycle too) $L_P = (\alpha, \beta)$ and $L_c = (0, \beta)$. When the vertex x_3 is reached, $L_P = (\alpha+\beta, \alpha)$ and $L_c = (\beta, 0)$. When (x_2, x_3) is visited, it implies the substitution of α and β in the macro M_1 by the values given at L_P, and L_c pairwise. In our example $L_P = (2(\alpha + \beta) + \alpha, \alpha + \beta) = (3\alpha + 2\beta, \alpha + \beta)$ and $L_c = (2\beta, \beta)$. The simple cycle is then closed so the last pair of equation is $(3\alpha + 2\beta, \alpha + \beta) - (0, \beta) = (3\alpha + 2\beta, \alpha)$. Finally $\#2SAT(F) = 3\alpha + 2\beta + \alpha$. With the initial values $\alpha = \beta = 1$ we have that $\#2SAT(F) = 6$.

M_1 indicates that it does not matter the values for α and β, the values for those variables can be substituted by a current pair of values in order to obtain a final pair of linear equations and such that in those new equations, the value for $\#2SAT(B_2)$ has been considered as part of its cumulative operations.

In our case, the *expansion* of a macro consists in the substitution of the symbolic variables α and β, appearing in its pair of equations, by the current values already computed when the common edge is reached. This process of expansion is well-defined since no macro appears in its own expansion.

The correctness of our method is based in the following Theorem.

Theorem 2. *Let F_1 and F_2 be two formulas in 2-CNF. If $F_1 \cap F_2 = \{x_1^\epsilon, x_2^\delta\}$, e.g. a single clause then*

$$\begin{aligned}\#2SAT(F_1 \cup F_2) =& \#2SAT(F_1 \mid_{\{x_1^\epsilon, x_2^\delta\} \subseteq s}) \times \#2SAT(F_2 \mid_{\{x_1^\epsilon, x_2^\delta\} \subseteq s}) + \\ & \#2SAT(F_1 \mid_{\{x_1^\epsilon, x_2^{\delta-1}\} \subseteq s}) \times \#2SAT(F_2 \mid_{\{x_1^\epsilon, x_2^{\delta-1}\} \subseteq s}) + \\ & \#2SAT(F_1 \mid_{\{x_1^{\epsilon-1}, x_2^\delta\} \subseteq s}) \times \#2SAT(F_2 \mid_{\{x_1^{\epsilon-1}, x_2^\delta\} \subseteq s})\end{aligned}$$

Proof. In order to satisfy $F_1 \cup F_2$ the clause $\{x_1^\epsilon, x_2^\delta\}$ has to be satisfied, so either $\{x_1^\epsilon, x_2^\delta\} \subseteq s$ or $\{x_1^\epsilon, x_2^{\delta-1}\} \subseteq s$ or $\{x_1^{\epsilon-1}, x_2^\delta\} \subseteq s$. The computation of the satisfying assignments of $F_1 \cup F_2$ is given by

$$\begin{aligned}\#2SAT(F_1 \cup F_2) =& \#2SAT(F_1 \cup F_2 \mid_{\{x_1^\epsilon, x_2^\delta\} \subseteq s}) + \\ & \#2SAT(F_1 \cup F_2 \mid_{\{x_1^\epsilon, x_2^{\delta-1}\} \subseteq s}) + \\ & \#2SAT(F_1 \cup F_2 \mid_{\{x_1^{\epsilon-1}, x_2^\delta\} \subseteq s})\end{aligned}$$

Assigning truth values to the variables x_1 and x_2 to satisfy $\{x_1^\epsilon, x_2^\delta\}$ in $F_1 \cup F_2$ gives two disconnected formula, by the hypothesis that $F_1 \cap F_2 = \{x_1^\epsilon, x_2^\delta\}$, so the conclusion holds. □

The previous theorem states that if we know the models of F_1 where the truth values of the variables x_1 and x_2 which joint F_1 to another formula F_2 via

a clause $\{x_1^\epsilon, x_2^\delta\}$ are known, then we can substitute the models where x_1^ϵ and x_2^δ appears in F_1 into those of F_2 considering the truth values of x_1 and x_2 in F_2.

Algorithm 2 presents the computation of $\#2SAT(F)$ when F is an outerplanar formula.

5 Results

We implement our proposal and compare its runtime against sharpSAT which to the best of our knowledge is the leading sequential implementation. Our outerplannar formulas represent polygonal chain graphs where each polygon has three or four sides. Tabla 1 shows the running time of our proposal against sharpSAT. It is work to said that both implementations are sound and complete hence the exact number of models is computed in both of them.

			Time in Seconds	
Instance	Variables	Clauses	markSAT	sharpSAT
1	102	201	0.002	0.028
2	202	401	0.005	0.035
3	502	1001	0.021	0.083
4	1002	2001	0.076	0.230
5	2,002	4,001	0.295	0.822
6	3,002	6,001	0.649	1.792
7	4,002	8,001	1.134	3.176
8	5,002	10,001	1.726	4.898
9	10,002	20,001	6.927	19.335
10	15,002	30,001	21.821	43.411
11	20,002	40,001	47.379	77.283
12	25,002	50,001	83.371	121.271
13	30,002	60,001	129.012	174.213
14	35,002	70,001	186.094	237.553
15	40,002	80,001	249.291	310.091

Table 1. Formulas whose signed constraint graphs is outerplannar.

Conclusion

We present a new class of conjunctive form formulas where counting their number of models can be computed in linear time. These formulas are the logical expressions of outerplanar graphs via two conjunctive normal forms.

Our procedure requires the construction of the tree decomposition of the outerplanar graphs, which in this case it is done in time $O(n)$ on the number of vertices of the input formula. Once a tree decomposition has been built a

Algorithm 2 Procedure that computes a pair of linear equations $(A\alpha, B\beta)$ such that when substituting $\alpha = \beta = 1$ gives $\#2SAT(F) = A + B$. F is an outer planar formula based whose tree decomposition is denoted by T computed by its signed constrained graph G.

procedure #2SAT(T)
if $T = NULL$ **then**
  return
end if
for each child C of T **do**
  #2SAT(C){Postorder traversal of T}
end for
if T is a leaf **then**
  switch (in case the joint of T and its fathers is)
  (a vertex x): compute the pairs of T using x as the root node{e.g leaving the pair (α_x, β_x) as a partial result}
  (an edge: (x, y)) compute the macro M of T using either x as the root vertex{e.g leaving the macro M as a partial result on x, y}
  end switch
end if
if T is an interior node **then**
  let $((\alpha_i, \beta_i)$ the pair associated to each child of T){A macro or a pair (α, β)}
  switch (in case the joint of T and its fathers is)
  (a vertex x): Traverse the vertices of T in postorder using as root the vertex x
  (an edge: (x, y)) compute the macro M of T using either x or y as the root node{e.g leaving the macro M as a partial result}
  end switch
  Substitute M or (α_i, β_i) when the child of T is reached in the traversal {according to the procedures of [1]}
end if
if T is the root **then**
  let $((\alpha_i, \beta_i)$ the pair associated to each child of T) {A macro M or a pair (α, β)}
  Choose a vertex x of T as the root node
  Traverse the vertices of T in postorder
  Substitute M or (α_i, β_i) when the child of T is reached in the traversal {According to the procedures of Section [1]}
end if
return (α_x, β_x).

postorder traversal of both the tree and their bags is done in time complexity $O(m)$, where m is the number of edges in the graph.

References

1. Guillermo De Ita, Pedro Bello, and Meliza Contreras. New polynomial classes for #2sat established via graph-topological structure. *Engineering Letters*, 15:2, 2007.
2. Darwiche A. On the tractability of counting theory models and its application to belief revision and truth maintenance. *Journal of Applied Non-classical Logics*, 11(1-2):11–34, 2001.
3. Dan Roth. On the hardness of approximate reasoning. *Artif. Intell.*, 82(1-2):273–302, April 1996.
4. Guillermo De Ita Luna. *Polynomial Classes of Boolean Formulas for Computing the Degree of Belief*, pages 430–440. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.
5. Magnus Wahlström. *A Tighter Bound for Counting Max-Weight Solutions to 2SAT Instances*, pages 202–213. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.
6. Evgeny Dantsin and Alexander Wolpert. *An Improved Upper Bound for SAT*, pages 400–407. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.
7. M. A. López, José Raymundo Marcial-Romero, Guillermo De Ita Luna, Héctor A. Montes Venegas, and Roberto Alejo. A linear time algorithm for solving #2sat on cactus formulas. *CoRR*, abs/1702.08581, 2017.
8. Zmazek B. *Estimating the traffic on weighted cactus networks in linear time*, pages 536–541. 2005.
9. Shiu W.C. Extremal hosoya index and merri
eld-simmons index of hexagonal spiders. *Discrete Applied Mathematics*, 156:2978–2985, 2008.
10. Stephan Wagner and Ivan Gutman. Maxima and minima of the hosoya index and the merrifield-simmons index. *Acta Applicandae Mathematicae*, 112(3):323–346, 2010.
11. Stefan Szeider. *On Fixed-Parameter Tractable Parameterizations of SAT*, pages 188–202. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.
12. Bodlaender H.L. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal of Computer*, 25(6):1305–1317, 1996.
13. Joachim Kneis and Alexander Langer. A practical approach to courcelle's theorem. *Electronic Notes in Theoretical Computer Science*, 251(Supplement C):65 – 81, 2009. Proceedings of the International Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS 2008).
14. Fedor V. Fomin, Serge Gaspers, Saket Saurabh, and Alexey A. Stepanov. On two techniques of combining branching and treewidth. *Algorithmica*, 54(2):181–207, Jun 2009.
15. Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic Discrete Methods*, 8(2):277–284, April 1987.
16. Hans L. Bodlaender. Classes of graphs with bounded tree-width. Technical report, Utrecht University, 1986.
17. Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1):1 – 45, 1998.
18. C. Bäckström, A. Jonsson, and P. Jonsson. Automaton plans. *Journal of Artificial Intelligence Research*, 51:255–291, 2014.

Capítulo 4

Conclusiones y trabajo futuro

El problema de conteo de modelos sobre fórmulas dadas en FNC #SAT es considerado un problema #P-Completo, por lo cual al no existir procedimientos que puedan resolverlo de manera general en tiempo polinomial, se buscan métodos que al realizar una descomposición de la fórmula permitan obtener instancias específicas, es decir, poder caracterizar estas instancias para generar un procedimiento que las resuelva en tiempo polinomial.

El primer algoritmo desarrollado identifica fórmulas cuya topología de gráfica es de tipo *cactus*, utilizando la generación de un árbol de expansión mediante las cláusulas de la fórmula, además de identificar la independencia de un ciclo con respecto a otro, es decir, que compartan a lo más un vértice en la gráfica. Este procedimiento lo realiza identificando todas las aristas que están dentro de un ciclo, de tal forma que al encontrar una arista que ya fue contemplada en un ciclo anterior se concluye que la fórmula no es cactus y se debe realizar una descomposición sobre el ciclo que intersecta al encontrado previamente.

El segundo algoritmo fue la modificación del primer algoritmo, en este caso se aumentó la capacidad de cálculo del algoritmo identificando conjuntos de ciclos intersectados, este caso permite que al descomponer una fórmula pueda separar subfórmulas independientes. Esto se logra al identificar conjuntos de ciclos intersectados que tienen en común sólo un vértice, lo que permite evaluarlos de manera independiente. Con esto se logra incrementar la eficiencia del algoritmo para la descomposición de una fórmula ya que puede resolver subfórmulas independientes algo que el algoritmo anterior no podía realizar. En este segundo algoritmo se utilizaron como prueba gráficas de manera general y tipo cactus.

El tercer algoritmo fue desarrollado para identificar topologías de gráficas tipo *outerplanar*, mediante la generación de un árbol de expansión que utiliza el primer algoritmo, sin embargo, realiza un conjunto de verificaciones sobre el árbol de expansión para identificar las aristas comunes entre cualquier par de ciclos en la gráfica, ya que una de las carac-

terísticas de esta topología es que cualquier par de ciclos comparten a lo más una arista. Al identificar estas aristas comunes, es posible generar el conjunto de bolsas necesario para la descomposición de la gráfica, ya que una arista común representa un camino de bolsas en la descomposición. En este tercer algoritmo se contempla la identificación de gráficas cactus utilizando el mismo procedimiento de las gráficas outerplanar, ya que al ser una versión más general, éste puede ser utilizado para resolver gráficas cactus.

Ya que los algoritmos obtenidos son completos, es decir que para cualquier instancia obtienen los modelos correctamente, sólo se pueden comparar los tiempos de ejecución en las distintas instancias de prueba.

De manera general el tercer algoritmo es el producto final de esta investigación, ya que incluye todos los casos presentados en los dos algoritmos anteriores y mejora el tiempo de cómputo en los resultados presentados en los artículos.

Los resultados obtenidos permiten afirmar que se logró el propósito de esta investigación, comparando de manera experimental los algoritmos obtenidos contra la herramienta más eficiente reportada en la literatura *sharpSAT*, se logró una mejora en el tiempo de cómputo tanto para fórmulas tipo cactus y outerplanar, realizadas en un 50 % del tiempo utilizado por *sharpSAT* para todas las pruebas de este tipo de gráficas, como en el caso de fórmulas en general, usando un 13 % del tiempo utilizado por *sharpSAT*, trabajando sobre la instancia más grande dentro de las pruebas. Las pruebas se realizaron en fórmulas generales, con porcentajes de cláusulas y variables distintos por un programa en el cual se indicó el porcentaje de cláusulas que tendría una fórmula dependiendo del número de variables que se tenían incrementando los porcentajes en 10 %. Además de utilizar instancias propias de fórmulas cactus y outerplanar considerados para verificar que se cumple la hipótesis.

Los resultados obtenidos en esta investigación muestran que ampliar los casos base en la descomposición de una fórmula permiten mejorar el tiempo de conteo de modelos. Por lo tanto estos casos base son factores determinantes para el desarrollo de futuros procedimientos.

La limitante más significativa para el tercer algoritmo radica en la implementación, ya que al determinar casos base que se pueden resolver en tiempo polinomial se generan procedimientos para resolverlos en tiempo lineal, sin embargo, el uso de espacio en memoria usado por el programa sólo permitió considerar instancias de manera general con 6500 variables, las cuales llegaron a tener poco mas de 21 millones de cláusulas.

Esta limitación de espacio en memoria se pudo enfrentar utilizando una tabla, en la que se colocan todas las aristas de ciclo, que permitiera gestionar múltiples particiones, sin necesidad de generar una tabla para cada partición.

La búsqueda de casos base y desarrollo de procedimientos para resolverlos es un área de trabajo extensa ya que requiere conocimiento de las fórmulas en FNC y sus distintas

representaciones teóricas sobre las cuales llevar a cabo la búsqueda de procedimientos adecuados para resolverlas.

Referencias

[1] Bondy J. A. and Murty U.S.R. Graph Theory. Springer Verlag, Graduate Texts in Mathematics, (2010).

[2] Brightwell G., Winkler Peter. Counting linear extensions. Order, Vol. 8, Issue e, pp. 225-242, (1991).

[3] Paulusma, D., Slivovsky, F., Szeider, S. Model counting for CNF formulas ofbounded modular treewidth. In: Portier, N., Wilke, T. (eds.) STACS. LIPIcs, Vol. 20, pp. 55-66. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, (2013).

[4] Magnus Wahlström. A tighter bound for counting max-weight solutions to 2SAT instances. In Proc. 3rd Int. Workshop on Parametrical and Exact Computation(IWPEC), pp. 202-213, (2008).

[5] Bayardo R. and R. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In: Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97). New Providence, RI, pp. 203-208, (1997).

[6] Thurley M. sharpSAT - Counting models with Advanced Component Caching and Implicit BCP. Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing (SAT 2006). pp. 424-429, (2006).

[7] Jan Burchard, Tobias Schubert, Bernd Becker. Laissez-Faire Caching for Parallel #SAT Solving. SAT 2015. pp. 46-61, (2015).

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

[9] Schmitt M. and Wanka R.Exploiting Independent Subformulas: A Faster Approximation Scheme for #k -SAT. Information Processing Letters, Vol. 113, Issue 9, pp. 337-344, (2013).

[10] Ramamohan Paturi, Pavel Pudlák, Michael E. Saks, and Francis Zane. An improved exponential-time algorithm for k-SAT. Journal of the ACM, Vol. 52, pp. 337-364, (2005).

[11] Timon Hertli. 3-SAT faster and simpler - Unique-SAT bounds for PPSZ hold in general. In Proc. 52nd IEEE Symp. on Foundations of Computer Science (FOCS), pp. 277-284, (2011).

[12] López M. Marco A., Marcial-Romero J. R., De Ita G., Hernández, J. A. Un algoritmo para calcular #2SAT. Research in Computer Science, Vol. 94, pp. 23-32, (2015).

[13] Marcial-Romero J. R., De Ita G., Hernández, J. A., Valdovinos, R. M. A Parametric Polynomial Deterministic Algorithm for #2SAT, Lecture Notes in Computer Science, Vol. 9413, pp. 202-213, (2015).

[14] Darwiche A., On the Tractability of Counting Theory Models and its Application to Belief Revision and Truth Maintenance, Jour. of Applied Non-classical Logics, , pp. 11-34, (2001).

[15] Dahllöf V., Jonsson P., Wahlström M., Counting models for 2SAT and 3SAT formulae., Theoretical Computer Sciences, 332, Vol. 1, Issue 3, pp. 265-291, (2005).

[16] Khardon R., Roth D., Reasoning with Models, Artificial Intelligence, Vol. 87, No. 1, pp. 187-213, (1996).

[17] Roth D., On the hardness of approximate reasoning, Artificial Intelligence, Vol. 82, pp. 273-302, (1996).

[18] Vadhan Salil P., The Complexity of Counting in Sparse, Regular, and Planar Graphs, SIAM Journal on Computing, Vol. 31, Issue 2, pp. 398-427, (2001).

[19] Bubley R.: Randomized Algorithms: Approximation, Generation, and Counting. Springer, Distinguished dissertations, Ed. 1, (2001).

[20] Armin Biere, Carsten Sinz, Decomposing SAT Problems into Connected Components. Journal on Satisfiability, Boolean Modeling and Computation, Vol. 2, pp. 201-208, (2006).

[21] T. Hertli, R.A. Moser, D. Scheder. Improving PPSZ for 3-SAT using Critical Variables. 28th Symposium on Theoretical Aspects of Computer Science, pp. 237-248, (2011).

[22] Bayardo R., Pehoushek J.D.Counting Models using Connected Components. In AAAI. pp. 157-162. (2000).

[23] Thurley M. An Approximation Algorithm for #k-SAT. 29th Symposium on Theoretical Aspects of Computer Science. pp. 78-87. (2012).