Un algoritmo para calcular #2SAT

Marco A. López Medina[1], J. Raymundo Marcial-Romero[1], Guillermo De-Ita[2], José A. Hernández[1]

[1] Facultad de Ingeniería, UAEM, México

[2] Facultad de Ciencias de la Computación, BUAP, Puebla, México

valgirmanda@gmail.com, jrmarcialr@uaemex.mx, deita@cs.buap.mx, xoseahernandez@uaemex.mx

Resumen. El problema de conteo de modelos en fórmulas booleanas pertenece a la clase #P-completo. Por tal motivo, no existe algoritmo qué, de forma eficiente, calcule el número exacto de modelos de una fórmula booleana. En este artículo, se presenta una implementación para contar el número de modelos de una fórmula booleana basada en su representación mediante grafo. Así mismo se mostrará que, para ciertas topologías del grafo, el algoritmo presentado establece el conteo de modelos de forma eficiente comparado con otras implementaciones reportadas en la literatura.

Palabras clave: #SAT, combinatoria, teoría de grafos.

1. Introducción

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. 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 de dos variables (2-Conjunctive Normal Form). El problema #SAT pertenece a la clase #P completo, por tanto no existe algoritmo eficiente que resuelva el problema en general. Existen implementaciones que permiten calcular los modelos de una fórmula dada como por ejemplo, relsat [5] la cual es una herramienta basada en el algoritmo DPLL (Davis-Putnam-Logemann-Loveland algorithm) de búsqueda exhaustiva de asignaciones. La mejora que provee en comparación a otras herramientas es el poder procesar rápidamente fórmulas disjuntas, es decir, que las variables de una fórmula no intervengan en alguna otra. Cachet [9] es una herramienta basada en el modelo de conteo *Caching* que es sucesora del árbol de búsqueda de un modelo de conteo DPLL; ajustando variables y simplificando la fórmula, ya que se pueden encontrar sub-fórmulas que han aparecido en una rama del árbol de búsqueda. Si esto sucede, no es necesario volver a contar los

modelos, sólo es necesario saber el conteo de la rama de búsqueda anterior y utilizarlo.

Por otro lado sharpSAT [1] está basada en el modelo *caching* e implementa un mejor razonamiento en cada nodo que evalúa. Utiliza una técnica conocida como *look ahead* que permite hacer una prueba sobre alguna variable en la fórmula para ver si es necesario que sea siempre verdadera o falsa, permitiendo que solo se cuenten los modelos necesarios para el resto de la fórmula.

En este artículo, se propone una estrategía para el conteo de modelos convirtiendo la fórmula de entrada en un grafo. Así mismo, se presentan fórmulas booleanas para las cuales las implementaciones existentes no son capaces de regresar un resultado y la que aquí se presenta lo puede realizar.

2. Preliminares

Los conceptos utilizados pueden consultarse en [7]. Sea $X = x_1, ..., x_n$ un conjunto de n variables booleanas (es decir pueden tomar únicamente dos valores de verdad). Una literal es una variable x_i o la variable negada $\bar{x}_i$. Una cláusula es una disjunción de literales distintas. Una fórmula booleana en forma normal conjuntiva (CNF, Conjunctive Normal Form) 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(F) es el conjunto de literales que aparecen en una CNF F.

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 $x^e \in s$, siendo s una asignación, entonces s convierte x^e en verdadero y x^{1-e} en falso, $e \in \{0, 1\}$. 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 $x^e \in c$, $x^{1-e} \in s$ entonces s falsifica a c.

Sea F una fórmula booleana en CNF, 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 CNF, 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-CNF.

2.1. El grafo restringido de una 2-CNF

Existen algunas representaciones gráficas de una Forma Normal Conjuntiva, por ejemplo el grafo primal signado o también conocido como grafo restringido.

Sea F una 2-CNF, su grafo restringido 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 vertices 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. Un grafo con aristas etiquetadas en S es el par (G, ψ), dónde $G = (V, E)$ es un grafo restringido, 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 $G = (V, E, \psi)$ un grafo restringido 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 un grafo conectado 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.

3. Conteo de modelos en grafos acíclicos

El conteo de modelos para fórmulas booleanas cuyo grafo restringido es acíclico se puede realizar en tiempo polinomial [4]. A continuación se presenta un algoritmo que, mediante un recorrido en preorder, permite contar modelos en un grafo acíclico.

3.1. #2SAT para fórmulas en 2-CNF que representan un camino

Se dice que un grafo G_F representa un camino para una 2-CNF F, si

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

donde $\delta_i, \epsilon_i \in \{0, 1\},\ i \in [1..m]$. Sea f_i una familia de cláusula de la fórmula F construida como sigue: $f_1 = \emptyset; f_i = \{C_j\}_{j<i}$, $i \in [1..m]$. Note que $n = |v(F)| = m + 1$, $f_i \subset f_{i+1}$, $i \in [1..m-1]$. Sea $SAT(f_i) = \{s : s \text{ satisface } f_i\}$, $A_i = \{s \in SAT(f_i) : x_i \in s\}$, $B_i = \{s \in SAT(f_i) : \overline{x}_i \in s\}$. Sea $\alpha_i = |A_i|;\ \beta_i = |B_i|$ y $\mu_i = |SAT(f_i)| = \alpha_i + \beta_i$.

Para cada nodo $x \in G_F$ se calcula el par (α_x, β_x), donde α_x indica el número de veces que la variable x toma el valor 'verdadero' y β_x indica el número de veces que la variable x toma el valor 'falso' en el conjunto de modelos de F. El primer par es $(\alpha_1, \beta_1) = (1, 1)$ ya que x_1 puede tomar el valor verdadero o falso para satisfacer a f_1. Los pares (α_x, β_x) asociados a cada nodo x_i, $i = 2, \ldots, m$ se calculan de acuerdo a los signos (ϵ_i, δ_i) de las literales en la cláusula c_i por la siguiente ecuación de recurrencia:

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

Note que, si $F = f_m$ entonces $\#SAT(F) = \mu_m = \alpha_m + \beta_m$. Se denota con $\rightarrow$ la aplicación de una de las cuatro reglas de recurrencia (1).

Ejemplo 1 *Sea* $F = \{(x_1, x_2), (\overline{x}_2, \overline{x}_3), (\overline{x}_3, \overline{x}_4), (x_4, \overline{x}_5), (\overline{x}_5, x_6)\}$ *una fórmula en 2-CNF cuyo grafo restringido representa un camino. Las series* $(\alpha_i, \beta_i), i \in [1..6]$*, se calculan como:* $(\alpha_1, \beta_1) = (1, 1) \rightarrow (\alpha_2, \beta_2) = (2, 1)$ *ya que* $(\epsilon_1, \delta_1) = (+, +)$*, y la regla 4 tiene que aplicarse. En general, aplicando la regla correspondiente de la recurrencia (1) de acuerdo a los signos de* (ϵ_i, δ_i)*,* $i = 2, ..., 5$*, se tiene que* $(2, 1) \rightarrow (1, 3) \rightarrow (3, 4) \rightarrow (3, 7) \rightarrow (\alpha_6, \beta_6) = (10, 7)$*, y entonces,* $\#SAT(F) = \mu_6 = \alpha_6 + \beta_6 = 10 + 7 = 17$.

3.2. Conteo en grafos acíclicos

Sea F una fórmula en 2-CF donde su grafo asociado G_F es acíclico. Se puede asumir que G_F tiene un nodo raíz, un recorrido del grafo permite generar un árbol que tiene un nodo raíz ya que es acíclico. Un árbol tiene tres clases de nodos: raíz, interior y hojas.

Se denota con (α_v, β_v) el par asociado con el nodo v ($v \in G_F$). Se calcula $\#SAT(F)$ mientras se recorre G_F en post-order con el siguiente algoritmo.
Algoritmo Conteo_Modelos_para_arbol(G_F)
Entrada: G_F - un grafo que representa un árbol.
Salida: El número de modelos de F
Procedimiento:
Recorrer G_F en post-order, para cada nodo $v \in G_F$, asignar:

1. $(\alpha_v, \beta_v) = (1, 1)$ si v es un nodo hoja en G_F.
2. Si v es un nodo interior con una lista de nodos hijos asociados, i.e., $u_1, u_2, ..., u_k$ son los nodos hijos de v, una vez que se han visitado los hijos, los pares calculados son $(\alpha_{u_j}, \beta_{u_j})$ $j = 1, ..., k$. Sean $e_1 = v^{\epsilon_1} u_1^{\delta_1}, e_2 = v^{\epsilon_2} u_2^{\delta_2}, \ldots, e_k = v^{\epsilon_k} u_k^{\delta_k}$ las aristas que conectan v con cada uno de sus nodos hijos. El par $(\alpha_{e_j}, \beta_{e_j})$ se calcula para cada arista e_j basado en la recurrencia (1) donde $\alpha_{e_j - 1}$ is α_{u_j} y $\beta_{e_j - 1}$ es β_{u_j} para $j = 1, \ldots k$. Entonces, sea $\alpha_v = \prod_{j=1}^{k} \alpha_{e_j}$ y $\beta_v = \prod_{j=1}^{k} \beta_{e_j}$. Note que este paso incluye el caso en que v tiene solo un nodo como hijo.
3. Si v es el nodo raíz de G_F entonces regresar $(\alpha_v + \beta_v)$.

Este procedimiento regresa el número de modelos de F en tiempo $O(n + m)$ [8] el cual es el tiempo para recorrer un grafo en post-order.

Ejemplo 2 *Si* $F = \{(x_1, x_2), (x_2, x_3), (x_2, x_4), (x_2, x_5), (x_4, x_6), (x_6, x_7), (x_6, x_8)\}$ *es una fórmula en 2-CF, si* x_1 *es el nodo raíz, después de realizar un recorrido post-order el número de modelos a cada nivel del árbol se muestran en la Figura 1. El procedimiento* $Conteo\_Modelos\_para\_arbol$ *regresa* $\alpha_{x_1} = 41$*,* $\beta_{x_1} = 36$ *y el número total de modelos es:* $\#SAT(F) = 41 + 36 = 77$.

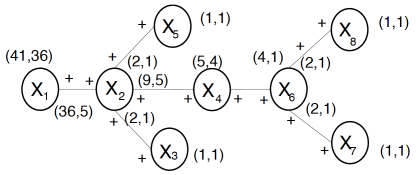


Fig. 1. Conteo de modelos en grafos que representan árboles

4. Conteo de modelos en grafos cíclicos

El problema #2SAT es #P-completo, para fórmulas cuyo grafo restringido es cíclico. En [4] se presenta un algoritmo para el conteo de modelos basado en el números de aristas del co-árbol. Por cada arista del co-árbol el algoritmo duplica el grafo de entrada eliminando la arista del co-árbol. Por lo tanto la complejidad del algoritmo esta dada por 2^r donde r es el número de aristas del co-árbol. Sin embargo, ya que las aristas del co-árbol son aquellas que forman cíclos, un grafo con r cíclos requiere 2^r árboles.

En esta sección mostramos cómo se puede disminuir el número de árboles generados para así hacer más eficiente el procedimiento para cierto tipo de grafos. Los árboles de expansión que se mencionan a continuación se construyen utilizando el método Depth First Search (DFS). El método DFS garantiza que todas las aristas del co-árbol son de retroceso es decir, que si se incorporan al árbol conectan nodos descendientes con nodos ancestros y no nodos que están en diferentes ramas del árbol. Por tal motivo se puede dividir el grafo original en r sub-árboles cada uno con su respectivo co-árbol, donde r es el número de nodos hijos asociados al nodo que se eligió como raíz en el grafo original.

Definición 1 *Sea F una fórmula cuyo grafo restringido $G_F = \rho(F)$ es cíclico. Sea T el árbol de expansión de G_F y $\overline{T}$ su co-árbol; se construye una familia de conjuntos $\mathcal{P}$ de $\overline{T}$ como sigue: un par de aristas $e_1, e_2 \in \overline{T}$ pertenecen al mismo conjunto sí y sólo si $path(e_1) \cap path(e_2) \neq \emptyset$ donde $path(e_i), i = 1, 2$ es el camino de los vértices de las aristas e_1 y e_2 en el árbol T.*

Lema 1 *La familia de conjuntos $\mathcal{P}$ de la Definición 1 es una partición de $\overline{T}$.*

***Prueba* 1** *Sean $X, Y \in \mathcal{P}$, es obvio que $X \cap Y \neq \emptyset$ ya que para cada par de aristas $e_1, e_2 \in \overline{T}$ hay un único camino en T. Dado que cada arista $e \in \overline{T}$ pertenece a una única partición entonces $\bigcup_{X \in \mathcal{P}} X = \overline{T}$*

Ahora se puede construir una partición de G_F.

Definición 2 *1. Para cada $P \in \mathcal{P}$, se construye un subgrafo como sigue: $\forall e \in P$,*

$$G_P(V(path(e)), E(path(e) \cup e))$$

2. Se define $G_R = G_F \setminus \bigcup G_{P \in \mathcal{P}}$.

Lema 2 *Los conjuntos $E(G_P), P \in \mathcal{P}$ junto con $E(G_R)$ forman una partición de $E(G_F)$.*

Lema 3 *Para cada par de grafos G_{P_1}, G_{P_2}, del Lema 2, ya sea que $V(G_{P_1}) \cap V(G_{P_2}) = \emptyset$ o $V(G_{P_1}) \cap V(G_{P_2}) = \{v\}$ es decir es vacío o un conjunto de un solo elemento.*

***Prueba* 2** *Por contradicción, supóngase que $V(G_{P_1}) \cap V(G_{P_2}) \neq \emptyset$ y $V(G_{P_1}) \cap V(G_{P_2}) \neq \{v\}$ lo que significa que hay al menos dos vértices v_1, v_2 en la intersección, lo que significa que la arista $e = (v_1, v_2)$ pertenece a la intersección contradiciendo la hipótesis de que G_{P_1} and G_{P_2} tienen un conjunto de aristas disjuntas.*

Teorema 1 *Para cada par de grafos G_{P_1}, G_{P_2} del lema 2*

1. *Si $G_{P_1} \cap G_{P_2} = \emptyset$ entonces los modelos que representa G_{P_1} son independientes de los modelos que representa G_{P_2} es decir $Models(G_{P_1} \cup G_{P_2}) = Models(G_{P_1}) \times Models(G_{P_2})$.*
2. *Si $G_{P_1} \cap G_{P_2} = \{v\}$ entonces*

$$\begin{aligned} Modelos(G_{P_1} \cup G_{P_2}) = Modelos(G_{P_1}|_{v^1}) \times & Modelos(G_{P_2}|_{v^1}) \\ + & \\ Modelos(G_{P_1}|_{v^0}) \times & Modelos(G_{P_2}|_{v^0}) \end{aligned}$$

***Prueba* 3** *1. Si $G_{P_1} \cap G_{P_2} = \emptyset$ ninguno de los vértices o aristas son compartidos. Ya que cada vértice de los grafos representan una variable de la fórmula de entrada, $\rho^{-1}(G_{P_1}) \cap \rho^{-1}(G_{P_2}) = \emptyset$, es decir no hay variables en común de las fórmulas representadas por cada sub-grafo. Es bien sabido que los modelos de fórmulas sin variables en común se pueden calcular como el producto de los modelos de cada fórmula.*

2. *Que la intersección sea un conjunto con un solo elemento significa que si $F_1 = \rho^{-1}(G_{P_1})$ y $F_2 = \rho^{-1}(G_{P_2})$ entonces $\nu(F_1) \cap \nu(F_2) = \{x_1\}$, es decir hay una sola variable en común para ambas sub-fórmulas. Una estrategia branch and bound se puede utilizar, donde una rama cuenta los modelos donde x_1 se fija con el valor verdadero y la otra rama cuenta los modelos donde x_1 se fija con el valor falso en ambas sub-fórmulas y al final se suman los modelos. Una vez que x_1 se fijo con un valor ya sea verdadero o falso, no existen mas variables en común entre las sub-fórmulas , así que por 1, sus modelos se pueden calcular de forma independiente y posteriormente realizar el producto.*

Del Teorema 1 se puede concluir que el número de modelos de cada uno de los $G_{P \in \mathcal{P}}$ se puede calcular de forma independiente.

Dado que G_F es conectado, hay sub-grafos G_{P_1} y G_{P_2} tal que $V(G_{P_1}) \cap V(G_{P_1}) = \emptyset$ por lo que debe existir un camino en G_R que los una. Afortunadamente, los modelos de un camino se pueden calcular en tiempo polinomial con el procedimiento que se presentó en la sección 3.1.

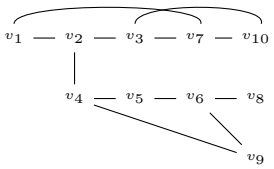
Así, se concluye que

Teorema 2 *Sea F una fórmula en 2-CNF, sea $G = \rho(F)$ su grafo restringido. Si P es una partición de G como se estableció en la Definición 1 y $\overline{T_i}, i = 1, \ldots r$ son las particiones que contienen aristas en el co-árbol, entonces la complejidad en tiempo para calcular $Modelos(G)$ es $O(2^{max\{|E(\overline{T_i})|\}} \cdot poly(|E(T)|))$, donde poly es una función polinomial.*

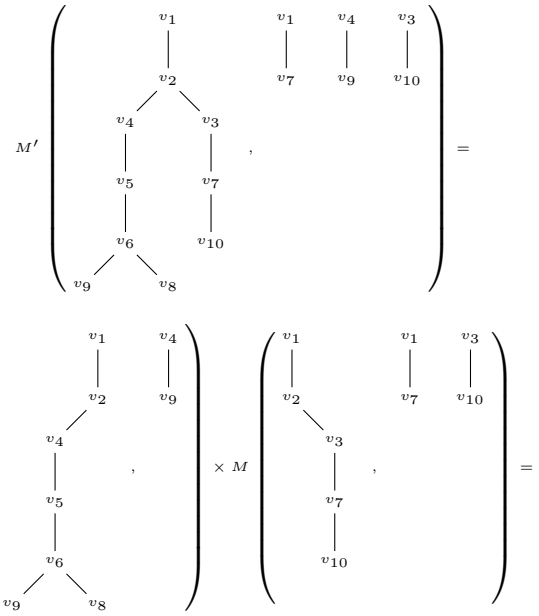
***Prueba* 4** *Por el Teorema 1 cada $\overline{T_i}$ se puede calcular de forma independiente. Así la duplicación máxima del grafo esta dado por $k = max\{|\ E(\overline{T_i})|\}$.*

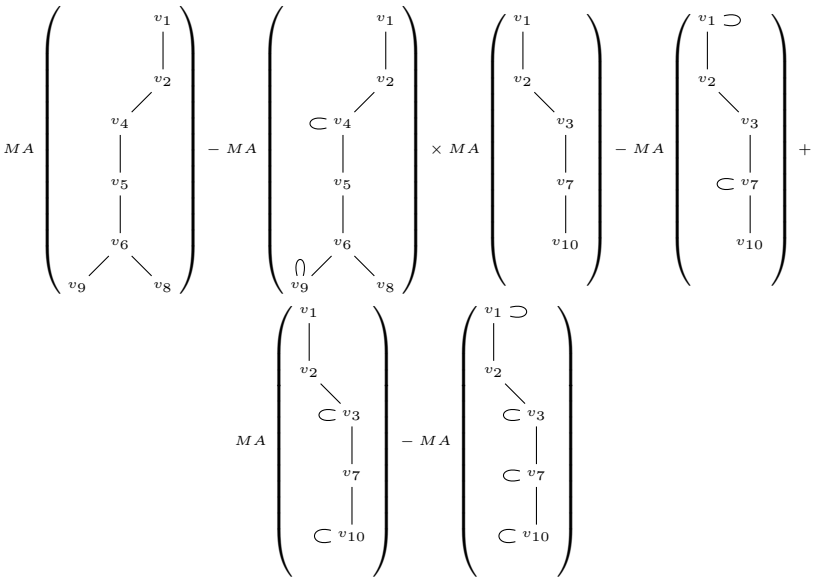
4.1. Ejemplo

Sea $F = \{\{v_1, v_2\}, \{v_1, v_7\}, \{v_2, v_4\}, \{v_2, v_3\}, \{v_3, v_7\}, \{v_3, v_{10}\}, \{v_4, v_5\}, \{v_4, v_9\}, \{v_5, v_6\}, \{v_7, v_{10}\}, \{v_6, v_8\}, \{v_6, v_9\}\}$. El grafo restringido $\rho(F)$ es (se omiten los signos ya que todos son positivos):



La aplicación de DFS del grafo restringido da la siguiente descomposición en árbol y co-árbol donde la artista del co-árbol (v_4, v_9) pertenece a una de las particiones y las aristas del co-árbol $(v_1, v_7), (v_3, v_{10})$ pertenecen a la otra partición. Así sus modelos se pueden calcular de forma independiente como se muestra abajo.





5. Resultados

De las herramientas reportadas en la literatura, aquella que ha dado mejores resultados para el conteo de modelos ha sido sharpSAT, por lo tanto en este artículo se realiza una comparativa con esta herramienta.

El algoritmo implementado, construye el árbol de expansión y posteriormente, divide el árbol de expansión generado en sub-árboles, cada uno con sus respectivos co-árboles para realizar el conteo de modelos. Por cada sub-árbol generado se aplica el procedimiento descrito en la sección anterior y con los árboles generados, se procede a contar sus modelos. Ya que cada sub-árbol se realiza por separado, la complejidad del cálculo disminuye; por ejemplo, en la prueba número 6 de la tabla 1, que contiene 30 cíclos, en lugar de tener 2^{30} árboles se generan dos sub-grafos con 2^{15} cíclos cada una por lo que el número de árboles generados es $2^{15} + 2^{15} = 2^{16}$ árboles en lugar de 2^{30}.

En la tabla 2 se muestran pruebas con grafos que tienen un mayor número de ciclos desde 120 hasta 26660. Ninguno de los algoritmos que se basa en el número de ciclos podría dar una respuesta en un tiempo considerable ya que por ejemplo la primera prueba requeriría 2^{120} cálculos. Sin embargo, ya que el grafo se puede particionar de tal forma que cada sub-grafo tenga únicamente 5 ciclos, se pueden hacer los cálculos de forma independiente para finalmente sumar los resultados de cada uno de ellos. Lo mismo sucede con las otras pruebas que se presentan en la tabla 2. Cabe mencionar que, dé los grafos reportados en la tabla 2 ninguna de las herramientas descritas en la introducción fue capaz de regresar un resultado por lo tanto se omitió la columna de sharpSAT para no dejarla vacía.

Tabla 1. Pruebas del algoritmo general en grafos que contienen un número pequeño de ciclos

Nodos	Cláusulas	Ciclos	Sharp SAT		Este artículo	
			Modelos	Tiempo (s)	Modelos	Tiempo (s)
13	15	3	401	0	401	0
19	18	0	15660	0	15660	0
19	33	15	3512	0.1	3512	19
19	34	15	-	-	5520	18
37	51	15	-	-	28058400	18
37	66	30	-	-	6294080	37
60	59	0	1.33×10^{13}	0	1.33×10^{13}	0
70	69	0	1.85×10^{16}	0	1.85×10^{16}	0
85	84	0	4.38×10^{19}	0	4.38×10^{19}	0

Tabla 2. Grafos con contienen un mayor número de ciclos

#Nodos	# Cláusulas	# Sub-árboles	Ciclos por Sub-árbol	Ciclos	Modelos
125	240	24	5	120	1.96319612718888E+020
250	490	49	5	245	2.57058291067303E+041
500	990	99	5	495	4.62068181689974E+083
1000	1990	199	5	995	1.49454832733069E+168
2000	3990	399	5	1995	1.56357229190873E+323
2000	5328	333	10	3330	2.61605773705871E+281
4000	7990	799	5	5593	1.711330818417309E+660
4000	10656	666	10	6660	6.843758083624728 E+547
8000	21328	1333	10	13330	3.278591729502E+1114
16000	42656	2666	10	26660	1.074916372876E+2241

6. Conclusiones

Aunque el problema #SAT es en general #P-completo, existen diferentes instancias que se pueden resolver de forma eficiente. Si el grafo restringido de la entrada tiene ciclos, se muestra un procedimiento que permite calcular el número de modelos con complejidad del orden $O(2^{max\{|E(\overline{T_i})|\}} \cdot poly(|E(T)|))$, donde *poly* es una función polinomial y los $T_i, \overline{T_i}$ son los árboles y co-árboles respectivamente de la descomposición del grafo original . Este es un método que muestra que ciertas instancias se pueden resolver de manera eficiente.

Finalmente, se ha mostrado que la implementación del algoritmo produce mejores resultados que al menos otra implementación que se reporta en la literatura y que para cierta clase de grafos que tiene un número considerable de ciclos, el procedimiento regresa el número de modelos de forma eficiente.

Referencias

1. Bubley R.: Randomized Algorithms: Approximation, Generation, and Counting. Distinguished dissertations Springer (2001)
2. Bubley R., Dyer M.: Graph Orientations with No Sink and an Approximation for a Hard Case of #SAT. In: Proc. of the Eight Annual ACM-SIAM Symp. on Discrete Algorithms, pp. 248–257 (1997)
3. Dahllöf V., Jonsson P., Wahlström M.: Counting Satisfying Assignments in 2-SAT and 3-SAT. LNCS 2387, pp. 535–543 (2002)
4. De Ita G., Marcial-Romero J.R., Mayao Y.: An Enumerative Algorithm for #2SAT. Electronic Notes in Discrete Mathematics, vol. 46, pp 81–88 (2015)
5. D. Gonçalves.: Covering planar graphs with forests, one having a bounded maximum degree. Electronic Notes in Discrete Mathematics, 31, pp. 161–165 (2008)
6. Garey M., Johnson D.: Computers and Intractability a Guide to the Theory of NP-Completeness. W.H. Freeman and Co. (1979)
7. J. A. Bondy and U.S.R. Murty.: Graph Theory. Springer Verlag, Graduate Texts in Mathematics (2010)
8. Levit V.E., Mandrescu E., The independence polynomial of a graph - a survey, To appear, Holon Academic Inst. of Technology (2005)
9. M. Garey and D. Johnson: Computers and Intractability: a Guide to the Theory of NP-Completeness. W.H. Freeman and Co. (1997)
10. R. Tarjan: Depth-first search and linear graph algorithms. SIAM Journal on Computing, 1, pp. 146–160 (1972)
11. Telikepalli Kavitha, Christian Liebchen, Kurt Mehlhorn, Dimitrios Michail, Romeo Rizzi, Torsten Ueckerdt, and Katharina A. Zweig: Cycle bases in graphs characterization, algorithms, complexity, and applications. Computer Science Review, 3: pp. 199–243 (2009)
12. Vadhan Salil P.: The Complexity of Counting in Sparse, Regular, and Planar Graphs. SIAM Journal on Computing, Vol. 31, No.2, pp. 398–427 (2001)