Un algoritmo numérico para problemas de satisfacción booleana sin álgebra

Carlos Barrón Romero

cbarron@correo.azc.uam.mx

Universidad Autónoma Metropolitana, Unidad Azcapotzalco, México
Av. San Pablo No. 180, Col. Reynosa Tamaulipas, C.P. 02200
Ciudad de México, México

Resumen: *Con un método novedoso de resolución para el clásico problema de decisión de Satisfacción Booleana (SAT) formulado con cláusulas CNF se describen algoritmos que permiten determinar cuando una formula pertenece al lenguaje SAT, o no pertenece, sin recurrir al álgebra. El método se basa en la clase especial de problemas SAT, que denominamos Simple SAT (SSAT). El resultado es un algoritmo numérico computable en la base binaria cuya complejidad es lineal con respecto al numero de cláusulas mas un proceso de datos sobre las soluciones parciales y que esta acotado por a lo mas 2^{n-1} iteraciones. Se presentan resultados teóricos de la conmutabilidad y de la complejidad de los algoritmos similares o mejores a los del estado del arte para resolver SAT.*

Palabras clave: Lógica, SAT, CNF, Complejidad Algorítmica, Problemas NP, Álgebra Booleana

Abstract: *A set of algorithms with a novel method for solving the classical Boolean Satisfiability Decision problem (SAT) formulated with CNF clauses. They are for determining when a formula belongs (or not) to the SAT's language but without algebraic resolution. The method is based on the special class of SAT problems, that we call Simple SAT (SSAT). The resulting algorithm is a numerical computable based on the binary number system whose complexity is linear with respect to the number of clauses plus a process data on the partial solutions and it is bounded by at most 2^{n-1} iterations. Theoretical results depicts that the computability and complexity of the main algorithm is similar or better of the algorithms of the state of the art for solving SAT.*

Keywords: Logic, SAT, CNF, Algorithmic Complexity Problems NP, Boolean algebra

1. Introducción

El problema clásico de satisfacción Booleana (SAT por su nombre en inglés: *Boolean satisfiability problem*) es un problema estudiado y mencionado repetidamente en la enorme literatura de la Ciencia de la Computación. En [CLRS90] se define como un problema de lenguajes, o sea en la determinación de las fórmulas que pertenecen al lenguaje SAT. Debido a la equivalencia entre formulaciones lógicas y al crecimiento en la complejidad (se trata de un problema de la clase NP), gran parte de la literatura del problema SAT se reduce a formulaciones de cláusulas en forma normal conjuntiva (CNF), por ejemplo: $\varphi_1(x_4, x_3, x_2, x_1, x_0) = (x_4 \vee x_3 \vee \overline{x}_2) \wedge (\overline{x}_3 \vee x_2 \vee \overline{x}_1) \wedge (\overline{x}_2 \vee x_1 \vee \overline{x}_0)$. En este artículo nos centramos en resolver problemas SAT en CNF. La estrategia de resolución se basa en las propiedades del problema especial SAT, que denominamos Simple SAT (SSAT) donde las cláusulas de una fórmula lógica tienen combinaciones en verdad o negación de las mismas variables. Por ejemplo, $\varphi_2(x_2, x_1, x_0) = (x_2 \vee x_1 \vee \overline{x}_0) \wedge (\overline{x}_2 \vee x_1 \vee \overline{x}_0)$ es un SSAT$(3, 2)$ de 3 variables y 2 cláusulas. En este trabajo se presentan las propiedades de SSAT para la formulación de un algoritmo paralelo capaz de determinar para cualquier fórmula φ en CNF si $\varphi \in$ SAT, o no, sin recurrir al álgebra.

La siguiente sección presenta un breve marco teórico, hacia el estado del arte del problema SAT en formulación CNF. En la subsección 2.1 se presenta el modelo SSAT y sus propiedades que son la base de los algoritmos del artículo. La sección 3 presenta en forma detallada el algoritmo paralelo y los algoritmos que lo componen, junto con una explicación de su diseño y funcionamiento. La siguiente sección 4 presenta los resultados. Finalmente, la sección 5 contiene las conclusiones y trabajo futuro.

2. Teoría del dominio y trabajos previos

Este artículo se basa en mis trabajos previos sobre problemas la clase NP [Bar05,Bar10,Bar15,Bar16].

Los valores lógicos se representan por : 0 (falso) o 1 (verdadero). De aquí en adelante, $\Sigma = \{0, 1\}$ denota el alfabeto de las cadenas binarias o de los números binarios. Aquí en particular, se ha tomado una formulación de SAT como un problema de decisión similar a la formulación [CLRS90]. Es decir, determinar cuando $\varphi \in$ SAT, donde el lenguaje SAT $= \{\varphi \,|\, \exists x_i = v_i, v_i \in \{0, 1\}, i = 0, \ldots, n \wedge \phi(x_n, \ldots, x_0) \equiv 1\}$ y φ se forma bajo las reglas:

1. Las variables booleanas: $x_n, \ldots, x_0$.
2. Los operadores lógicos: **no** $(\overline{x})$, **o** $(\vee)$, **y** $(\wedge)$. Como se trata la versíon CNF, los operadores **si-entonces** $(\Rightarrow)$ y **si-y-solo-si** $(\Leftrightarrow)$ se han omitido.
3. Y los paréntesis "(",")"para formar cláusulas lógicas en la forma CNF. (Se ha restringido el uso de los paréntesis para que no considerar anidamientos).

La restricción a cláusulas CNF esta ampliamente justificada por la equivalencia lógica en la literatura al respecto de los problemas NP Completos y los algoritmos que puedan ser eficientes para resolverlos [Pud98,ZMMM01,ZM02,Tov84,Woe03,JW,For09], [Coo00,GSTS07].

La búsqueda o el tratar de alcanzar la meta de crear los algoritmos eficientes o el algoritmo eficiente para resolver el problema SAT se plantea particularmente en [ZM02]. Aquí se da un panorama de estrategias relacionadas con el álgebra booleana para lidiar con con el crecimiento de las ramificaciones que se obtienen al relacionar cláusulas y variables en busca de tautologías o contradicciones, al ir asignando valores a las variables de las cláusulas o bien manejar las ramificaciones o los árboles o grafos de alternativas mediante heurísticas apropiadas que conduzcan a una asignación de valores satisfactoria. En el trabajo [Pud98] anterior a [ZM02], se menciona la aceleración que se tendría con el uso de la computación cuántica para resolver SAT. Parafraseando como en el artículo [GSTS07], es fácil construir ejemplos difíciles de resolver por los algoritmos del estado del arte para SAT.

En este artículo usamos la notación SAT(n,m) o SAT$(n,\cdot)$ para denotar formulas en CNF, con n variables, m cláusulas o con un número $(\cdot)$ desconocido de cláusulas. Por ejemplo, $\varphi_3 = (x_3 \vee \overline{x}_2 \vee x_0) \wedge (x_2 \vee x_1 \vee x_0) \wedge (\overline{x}_2 \vee x_1 \vee x_0) \wedge (x_3 \vee \overline{x}_0)$ es un SAT$(4,4)$. Además, se puede afirmar que $\varphi_3 \in$ SAT ya que la asignación $x_3 = 1$, $x_2 = 1$, $x_1 = 0$ y $x_0 = 1$ es una solución. Para verificar esto, se procede a sustituir valores y a realizar el álgebra correspondiente: $\varphi(1,1,0,1) = (1 \vee 0 \vee 1) \wedge (1 \vee 0 \vee 1) \wedge (0 \vee 0 \vee 1) \wedge (1 \vee 0) \equiv 1$.

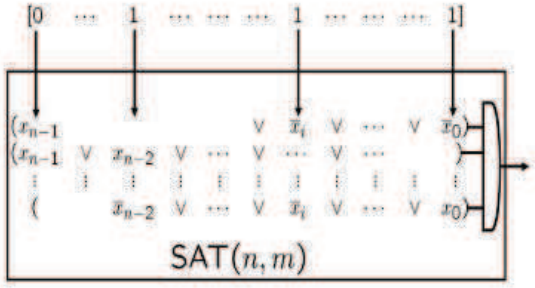

Figura 1. SAT(n,m) es un circuito de compuertas lógicas $\vee$ y $\wedge$ en CNF. Es una caja blanca para la evaluación de cualquier $x \in \Sigma^n$

Debido a que, no se tratan problemas de fórmulas mal escritas, sino de fórmulas que funcionan, una fórmula φ sea SAT, o sea una fórmula CNF, se puede interpretar y manejar como un circuito electrónico lógico dado como se muestra en la figura 1. Por ejemplo mediante multiplexores para una entrada de m líneas, el gran $\wedge$ (y) se puede implementar para con cualquier dirección $x \in Sigma^m$, tal que es 1 si $x = 1 \ldots 1$ y 0 en otro caso. En forma análoga las cláusulas pueden ser multiplexores tales sean 1 en cuando alguna de sus variables coincide con el valor de su linea y 0 otro caso. Esta interpretación tiene como consecuencia que el sistema de fórmulas de un problema SAT se interprete como una función lógica SAT$(n,m) : \Sigma^n \to \Sigma$ en un circuito cuyo tiempo de evaluación es el tiempo en que se activan los multiplexores interconectados en paralelo a sus respectivas líneas de datos de entrada y a la salida del gran multiplexor $(\wedge)$. Más importante, esta suposición conlleva a que el costo de evaluar SSAT$(n,m)(x)$ con $x \in \Sigma^n$ es $\mathbf{O}(k)$ y sin perdida de generalidad, asumir $k = 1$. Esta interpretación cambia que la parte de evaluación de una fórmula de problema SAT no sea resultado de un programa (software) sino que sea por medio de un circuito electrónico (hardware). Es decir, en lugar de programar se construye un circuito equivalente con el sistema de ecuaciones lógicas del problema dado. Este es una posibilidad para no evaluar por software, ya que la evaluación por medio de ciclos de una fórmula φ por un programa si bien es polinomial $(m * n)$, es a la larga costosa.

2.1. El problema especial de satisfacción Booleana, SSAT

El caso especial con el requisito adicional de que todas las cláusulas tengan el mismo número de variables lógicas, se denota como SSAT(n,m). Por ejemplo, una fórmula φ_4=SSAT$(3,2)$ es $(x_2 \vee x_1 \vee x_0) \wedge (\overline{x}_2 \vee x_1 \vee x_0)$.

Como ya es conocido, los valores que pueden tomar las n variables lógicas del problema, corresponden con todas las cadenas de Σ^n, o sea el espacio de los números binarios positivos $[0, 2^n - 1]$. El complemento de una cadena binaria $x \in \Sigma^n$, se denota como $\overline{x}$ y se estima como el complemento dígito a dígito de x. Por ejemplo, sea $x = 010$, entonces $\overline{x} = 101$.

Por la estructura del sistema normal conjuntivo de un problema SAT, usaremos cláusula o renglón para referirnos a una cláusula particular, como si la estructura de la fórmula fuera parecida a un tablero.

Resolver SSAT(n,m), es trivial como se explica más adelante si lo hacemos corresponder con un tablero de números binarios y porqué se tienen los parámetros n (número de variables) y m (número de cláusulas). Como por ejemplo, un SSAT$(3,2)$ es $(x_2 \vee x_1 \vee x_0) \wedge (\overline{x}_2 \vee x_1 \vee x_0)$.

Se tiene una correspondencia entre las cláusulas de un SSAT(n,m) y cadenas de Σ^n, como sigue. Dada una cláusula $(x_{n-1} \vee \ldots x_1 \vee x_0)$ se le hace corresponder variable a variable la cadena binaria $b_{n-1} \ldots b_1 b_0$ donde

$$b_i = \begin{cases} 0 & \text{si } \overline{x}_i, \\ 1 & \text{en otro caso.} \end{cases}$$

Entonces a un SSAT(n,m) le corresponde un arreglo de cadenas binarias, que denominaremos tablero

$$\begin{bmatrix} b^1_{n-1} b^1_{n-2} \cdots b^1_1 b^1_0 \\ b^2_{n-1} b^2_{n-2} \cdots b^2_1 b^2_0 \\ \vdots \ \cdots \ \cdots \ \cdots \ \cdots \ \vdots \\ b^m_{n-1} b^m_{n-2} \cdots b^m_1 b^m_0 \end{bmatrix}.$$

Por ejemplo, dos tableros SSAT(1, 2) y SSAT(2, 4) son:

x_0
1
0

x_1	x_0
0	0
1	1
0	1
1	0

El tablero de SSAT(1, 2) representa al sistema: $\begin{matrix}(x_0)\\ \wedge\,(\overline{x}_0)\end{matrix}$ que no tiene solución. En forma similar el tablero de SSAT(2, 4) no tiene solución, porqué corresponde con el sistema de fórmulas: $\begin{matrix}(\overline{x}_1 \vee \overline{x}_0)\\ \wedge\,(x_1 \vee x_0)\\ \wedge\,(\overline{x}_1 \vee x_0)\\ \wedge\,(x_1 \vee \overline{x}_0)\end{matrix}$ que no tiene solución.

La propiedad que tienen en común, la denominamos tablero bloqueado. Se puede generalizar para cualquier número de variables y consiste en que todas las cadenas binarias $x \in \Sigma^n$ están bloqueadas por su cadena complemento $\overline{x}$.

Proposición 1. *Sea φ del tipo SSAT$(n, 2^n)$ con todas sus cláusulas diferentes. Entonces SSAT$(n, 2^n)$ no tiene solución, i.e., $\varphi \notin$SAT.*

Demostración. Como todas las cláusulas son diferentes, estas corresponden con todas las combinaciones de cadenas binarias de Σ^n. Por lo que x y su complemento $\overline{x}$ se bloquean mutuamente. O sea, se trata de un tablero bloqueado.

Esta propiedad se puede heredar a un problema SAT, en el sentido de detectar o identificar un subproblema SSAT(n', m') bloqueado dentro de un SAT$(n, \cdot)$.

Proposición 2. *Dado una fórmula φ de tipo SAT(n, m). Si esta contiene un subproblema SSAT$(l, 2^l)$ para un subconjunto de variables de tamaño $l \leq n$ y con cláusulas diferentes. Entonces $\varphi \notin$SAT.*

Demostración. El subproblema SSAT$(l, 2^l)$ satisface la proposición 1. Y no se puede construir una asignación satisfactoria para φ con las n variables lógicas.

Por otro lado, se tiene la siguiente proposición.

Proposición 3. *Para cualquier fórmula φ de tipo SSAT(n, m) las condiciones a) $m < 2^n$ o b) $m = 2^n$ con todas sus fórmulas diferentes, garantizan que a) $\varphi \in$SAT y b) $\varphi \notin$SAT. La complejidad de la determinación de la pertenencia de φ a SAT bajo cualquiera de las condiciones anteriores y los parámetros n y m es* $\mathbf{O}(1)$.

Demostración. Para a) $\varphi \in$ SAT ya que aun con cláusulas repetidas, estas no cubren las cadenas de Σ^n. Para b) por la proposición 1. En ambos casos basta responder si $m < 2^n$ o si $m = 2^n \wedge$ con m cláusulas diferentes para determinar la pertenencia de φ a SAT.

Es importante notar una gran diferencia entre las condiciones a) y b) de la proposición 3 que es la suposición de cláusulas diferentes. Cuando se tiene $m < 2^n$ no se requiere del conocimiento de saber que las cláusulas son diferentes porque aún con fórmulas duplicadas, la hipótesis $m < 2^n$ significa que no se cubren todas las combinaciones de Σ^n. Del otro lado, b) (o la proposición 1) si lo requiere porqué tener todas las cláusulas diferentes es la justificación necesaria para un tablero bloqueado, i.e., para abarcar las todas las cadenas binarias de Σ^n.

Otra consecuencia de la identificación de las cláusulas como cadenas binarias de un tablero es que permite formular como construir un problema SSAT para que tenga como solución un conjunto dado de cadenas binarias o para que no tenga solución.

Proposición 4. *Sea n el número de variables lógicas.*

1. *Para construir un problema SSAT$(2, 2^n)$ sin solución, basta con que todas sus fórmulas correspondan con las diferentes combinaciones binarias de Σ^n.*
2. *Para construir un problema SSAT$(2, 2^n - 1)$ con una única solución. Sin perdida de generalidad, sea $s \in [0, 2^n - 1]$ el número binario solución. Entonces el problemas SSAT$(2, 2^n - 1)$ con s como asignación satisfactoria es el que tiene sus fórmulas en correspondencia con $\{s\} \cup \{x \in [0, 2^n - 1] \,|\, x \neq s \vee \overline{x} \neq s\}$.*
3. *Para construir un problema SSAT$(2, m)$ con un conjunto $S \neq \emptyset$ de soluciones. Sin perdida de generalidad, sea , $S \subset [0, 2^n - 1]$ el conjunto de números binarios solución. Entonces el problema SSAT$(2, m)$ con S como el conjunto de asignaciones satisfactorias es el que tiene sus fórmulas en correspondencia con $\{x \in [0, 2^n - 1] \,|\, (x \neq s \vee \overline{x} \neq s) \forall s \in S\} \cup \{x \in S \,|\, \overline{x} \notin S\}$.*

Demostración. 1. El SSAT construido corresponde con un tablero bloqueado para las n variables lógicas.

2. *Por construcción s es la única cadena de Σ^n que no está bloqueada.*
3. *La selección de fórmulas del SSAT corresponden a dos casos: 1) Las cadenas de Σ^n que no bloquean a las cadenas del conjunto solución y 2) las cadenas de la solución que no se bloquean entre ellas.*

Notablemente, cuando $m < 2^n$, con φ una fórmula SSAT, la respuesta de que $\varphi \in$SAT o $\varphi \notin$ SAT es inmediata con base en los parámetros n y m. No se requiere tener en concreto una asignación de valores de Σ^n, ya que el valor de m da una condición suficiente como lo establece la condición a) de la proposición 3.

3. Algoritmos para SAT

En adelante, las variables se identificaran como conjuntos con sus subíndices en orden decreciente, i.e., $x_{n-1}, x_{n-2}, \ldots, x_1, x_0$. Además, para el conjunto $X = \{x_{n-1}, \ldots, x_1, x_0\}$, como en el artículo [Bar10], Prop. 4. se da una enumeración para asignar un único

número entero como identificación de cualquier subconjunto de variables lógicas de X, como sigue:

Subconjunto de variables lógicas	$\mathbb{N}$
$\{\}$	$\leftrightarrow 0$
$\{x_0\}$	$\leftrightarrow 1 = \binom{n}{0}$
$\{x_1\}$	$\leftrightarrow 2 = \binom{n}{0} + 1$
$\vdots$	$\vdots$ $\vdots$
$\{x_1, x_0\}$	$\leftrightarrow k_1 = \sum_{k=0}^{1} \binom{n}{k}$
$\{x_2, x_0\}$	$\leftrightarrow k_2 = \sum_{k=0}^{1} \binom{n}{k} + 1$
$\vdots$	$\vdots$ $\vdots$
X	$\leftrightarrow 2^n - 1 = \sum_{k=0}^{n-1} \binom{n}{k}$

La función $IdSS : 2^X \to [0, 2^n - 1]$ dada por el siguiente algoritmo calcula la correspondencia entre un subconjunto de variables y su número de identificación.

Algoritmo 1 *Identificación de un SSAT*
Entrada: *$x = \{x_k, \ldots, x_1, x_0\}$: conjunto de variables indexadas en $[0, n]$ y en orden descendente.*
Salida: *ix: valor entero;*
// número de identificación en $[0, 2^n - 1]$ para los indices del conjunto x.
Memoria: *base: entero; v, t: (k+1)-ada de valores enteros de indices interpretados como dígitos de la base numérica n con los valores $\{n-1, n-2, \ldots, 1, 0\}$*

$ix = 0$;
$k = |x|$; // $|\cdot|$ cardinalidad de un conjunto.
si *k* ***igual*** *0* ***entonces***
  salida: *"ix";*
  regresar;
fin si
base = 0;
  itera *$j = 0, k-1$* ***hacer***
  base = base + $\binom{n}{j}$; // $\binom{\cdot}{\cdot}$ coeficiente binomial
fin itera
construir *$v = \{v_k, \ldots, v_0\}$; // conjunto de variables con los menores índices en orden descendente de tamaño $k+1$*
mientras *(indices(x) < (indices(v))* ***hacer.***
  $t = v$;
  repite
    $t = incrementa(t, 1)$;
    // incrementa en uno los indices de t como un número en base n
    si *indices(t) diferentes y en orden descendente* ***entonces***
      sal del ciclo *// t es un conjunto cuyos índices son diferentes y en orden descendente*
    fin si
  hasta *falso;*
  $v = t$;
  $ix = ix + 1$;
fin mientras *// el ciclo termina cuando encuentra el índice del conjunto dado*
salida: *"base + ix";*
regresar;

Algoritmo 2 *Actualiza SSAT*
Entrada: *(SSAT: Lista objetos, rw: cláusula).*
Salida: *SSAT: lista de objetos actualizada para el SSAT con identificación $IdSS(r)$.*
Cada SSAT($IdSS(r)$) actualiza S : lista de soluciones, donde $S[0 : 2^{n-1}]$: arreglo de enteros para una estructura doblemente encadenada, previous, next : enteros;
Memoria: *ct:=0 : entero; first = 0: entero; last = $2^n - 1$: entero;*

---

si *no existe SSAT($IdSS(rw)$)* ***entonces***
  crear objecto SSAT($IdSS(rw)$)
fin si
con *SSAT($IdSS(rw)$)*
  k = ***cláusula a binario*** *(rw);*
  si *$S[\overline{k}].previous \neq -1$*
  o *$S[\overline{k}].next \neq -1$* ***entonces***
    $S[S[\overline{k}].previous].next = S[\overline{k}].next$;
    $S[S[\overline{k}].next].previous = S[\overline{k}].previous$;
    si *$\overline{k} = first$* ***entonces***
      $first := S[\overline{k}].next$;
    fin si
    si *$\overline{k} = last$* ***entonces***
      $last := S[\overline{k}].previous$;
    fin si
    $S[\overline{k}].next = -1$;
    $S[\overline{k}].previous = -1$;
    $ct := ct + 1$;
  fin si
  si *$ct = 2^n$* ***entonces***
    salida: *" No hay solución para el problema dado SAT(n, m)*
    SSAT($IdSS(rw)$) es un tablero bloqueado.";
    detener todos los procesos, alto total;
  fin si
fin con
regresar

Algoritmo 3 *Resuelve $\varphi \in SAT$.*
Entrada: *n, $\varphi = SAT(n, \cdot)$.*
Salida: *Mensaje y si es el caso regresa x tal que $\varphi(n, \cdot)(x) = 1$.*
Memoria: *r: subconjunto de variables de X; SSAT=nulo: Lista de objetos SSAT.*

Mientras *no(terminen_cláusulas(SAT(n, ·)));*
  r = SAT(n, m).leer_cláusula;
  algoritmo.2 Actualiza SSAT(SSAT,r).
fin mientras;
Con la lista SSAT
  // $\times\theta$ producto cruz y junta natural
  determinar *$\Theta = \times\theta$ todos SSAT($IdSS(r)$);*
  si *$\Theta = \emptyset$* ***entonces***
    salida: *"El algoritmo 3 determina $\varphi \notin SAT$.*

Las soluciones de los SSAT($IdSS(r)$) son incompatibles."
detener todos los procesos, alto total;
en otro caso
salida: *"El algoritmo 3 confirma $\varphi \in SAT$. s $(s \in \Theta)$ es una asignación satisfactoria. Las soluciones de los SSAT($IdSS(r)$) son compatibles.*
detener todos los procesos, alto total;
fin con
fin si

El algoritmo siguiente es una versión del algoritmo 4 de [Bar15]. Los candidatos son seleccionados aleatoriamente y por una sola vez del espacio de búsqueda $[0, 2^n - 1]$.

Algoritmo 4 *Resuelve $\varphi \in SAT$, por exploración aleatoria de $[0, 2^n - 1]$.*
Entrada: *n, $\varphi = SAT(n, \cdot)$.*
Salida: *Mensaje y si es el caso s $(s \in [0, 2^n - 1])$, asignación satisfactoria, $\varphi(n, \cdot)(s) = 1$.*
Memoria: *$T[0 : 2^{n-1} - 1] = [0 : 2^n - 1]$: entero; $Mi = 2^n - 1$: entero; rdm, a: entero.*

---

iterar *i:=0 to* $2^{n-1} - 2$
si $T[i] = i$ **entonces**
// Selección aleatoria $rdm \in [i + 1, 2^{n-1} - 1]$;
$rdm = floor(rand()\ (Mi - i + 1{,}5)) + (i + 1)$;
// rand() numero aleatorio en (0,1);
// floor(x) menor entero de x
$a = T[rdm]$;
$T[rdm] = T[i]$;
$T[i] = a$;
fin si
$rdm = 0T[i]$; *// Concatenación 0 y $T[i]$*
si $SAT(n, m)(rdm) = 1$ **entonces**
salida: *"El algoritmo 4 confirma $\varphi \in SAT$. rdm is una asignación satisfactoria.";*
detener todos los procesos, alto total;
fin si
si $SAT(n, m)(\overline{rdm}) = 1$ **entonces**
salida: *"El algoritmo 4 confirma $\varphi \in SAT$. $\overline{rdm}$ es una asignación satisfactoria.";*
detener todos los procesos, alto total;
fin si
fin iterar
$rdm = 0T[2^{n-1} - 1]$; *// Concatenar 0 y $T[2^{n-1} - 1]$*
si $SAT(n, m)(rdm) = 1$ **entonces**
salida: *"El algoritmo 4 confirma $\varphi \in SAT$. rdm is una asignación satisfactoria.";*
detener todos los procesos, alto total;
fin si
si $SAT(n, m)(\overline{rdm}) = 1$ **entonces**
salida: *"El algoritmo 4 confirma $\varphi \in SAT$. $\overline{rdm}$ es una asignación satisfactoria.";*
detener todos los procesos, alto total;
fin si
salida: *"El algoritmo 4 determina $\varphi \notin SAT$, después de explorar todo el espacio de búsqueda $[0, 2^n]$.";*
detener todos los procesos, alto total;

Las iteraciones del algoritmo anterior tienen como límite superior $\mathbf{O}(2^{n-1})$. Por la estructura del espacio de búsqueda es posible bajar la cota de iteraciones, dividiendo el espacio $[0, 2^n - 1]$ en cuatro. Los cuatro candidatos a verificar serían $00x$, $01x$ y sus complementarios $11\overline{x}$ y $10\overline{x}$ con $x \in [0, 2^{n-2} - 1]$ el número de iteraciones máximo es 2^{n-2} pero el tiempo de iteración aumenta en un factor de dos por las cuatro pruebas, en lugar de las dos que usa el algoritmo 4. Este algoritmo puede terminar antes cuando encuentra una asignación satisfactoria y no importa si las cláusulas se repiten o están desordenadas o se trata de un problema SAT enorme, i.e., $m \gg 2^n$. Para casos con $m = 2^{n-1}$ la probabilidad de encontrar una asignación es muy alta, ya que esto significa que hay representados en cláusulas la mitad de las posibilidades de Σ^n. Sin embargo para casos extremos SSAT diseñados para tener una sola o ninguna asignación satisfactoria, la posibilidad de encontrar la posible asignación requiere tiempo exponencial y el caso sin asignación satisfactoria requiere las 2^{n-1} iteraciones.

Algoritmo 5 *Algoritmo Paralelo para SAT*
Entrada: *Fórmula φ en forma CNF de SAT de n variables.*
Salida: *Determina si $\varphi \in SAT$ o no.*

---

ejecuta en paralelo
algoritmo 3(n, φ);
algoritmo 4(n, φ);
fin ejecuta

Son dos las ideas para implementar el algoritmo 5 y determinar cuando $\varphi \in$SAT o no. Información detallada en [Bar15,Bar16]. El algoritmo 4 es una búsqueda sobre una permutación única y calculada al vuelo del espacio de soluciones $[0, 2^n - 1]$. El algoritmo 3 depende de leer las cláusulas de φ y determinar sus soluciones sin requerir realizar operaciones algebraicas, sino una depuración del espacio de búsqueda de cada SSAT, quitando las cadenas binarias del complemente de la traslación binaria de cada cláusula en su respectivo SSAT. Esto permite, al terminar de leer todas las cláusulas de φ, tener las soluciones de cada SSAT. Es similar a un proceso de un parser de un compilador en una sola pasada o lectura. Lo que se construye es el espacio de soluciones de los SSAT que después se pueden usar para determinar asignaciones satisfactorias para SAT por medio de la operación de datos $\times\theta$. Se usa el producto cruz y la junta natural (o θ join) de Base de Datos Relacionales en un operador que denominamos operador cruz-junta ($\times\theta$). Supongamos dos conjuntos de variables lógicas r y r', sus respectivos SSAT y sus soluciones SSAT$(\cdot).S$:

$\mathrm{SSAT}(IdSS(r)) \times_\theta \mathrm{SSAT}(IdSS(r')) =$

1. **si** $r \cap r' = \emptyset$ **entonces** $\mathrm{SSAT}(IdSS(r)).S \times \mathrm{SSAT}(IdSS(r')).S$.
2. **si** $r \cap r' \neq \emptyset$ y hay valores comunes entre $\mathrm{SSAT}(IdSS(r)).S$ and $\mathrm{SSAT}(IdSS(r')).S$ para las variables en $r \cap r'$ **entonces** $\mathrm{SSAT}(IdSS(r)).S\,\theta_{r\cap r'}\,\mathrm{SSAT}(IdSS(r')).S$
3. **si** $r \cap r' \neq \emptyset$ y no hay valores comunes entre $\mathrm{SSAT}(IdSS(r)).S$ y $\mathrm{SSAT}(IdSS(r')).S$ para las variables en $r \cap r'$ **entonces** $\emptyset$.

En los casos 1) y 2) $\mathrm{SSAT}(IdSS(r)$ y $\mathrm{SSAT}(IdSS(r')$ son compatibles. En el caso 3) son incompatibles, no hay una asignación satisfactoria para ambos.

Un ejemplo del caso 1), es φ_5 =SAT(4, 5) dado por $(x_3 \vee \overline{x}_2) \wedge (x_3 \vee x_2) \wedge (\overline{x}_3 \vee x_2) \wedge (\overline{x}_1 \vee \overline{x}_0) \wedge\ (x_1 \vee x_0)$. Se obtiene un SSAT(2, 3) de las tres primeras cláusulas de φ_5, cuya solución es $x_3 = 1, x_2 = 1$. Se obtiene SSAT(2, 2) de las dos últimas cláusulas de φ_5, cuyas son soluciones son $\{(x_1, x_0) \,|\, (0,1) \vee (1,0)\}$. Entonces las soluciones de φ_5 son $\{(1,1)\} \times \{(0,1),(1,0)\} = \{(x_3, x_2, x_1, x_0) \,|\, (1,1,0,1) \vee (1,1,1,0)\}$.

Un ejemplo del caso 2), es φ_6 =SAT(4, 5) dado por $(x_3 \vee \overline{x}_2) \wedge (\overline{x}_3 \vee \overline{x}_2) \wedge (\overline{x}_3 \vee x_2) \wedge (x_2 \vee \overline{x}_1 \vee x_0) \wedge (x_2 \vee x_1 \vee \overline{x}_0)$. Un SSAT(2, 3) se obtiene de las tres primeras cláusulas de φ_6, cuya solución es $\{(x_3, x_2) \,|\, (0,0)\}$. Un SSAT(3, 2) corresponde con las dos últimas cláusulas de φ_6, cuyas soluciones son $\{(x_2, x_1, x_0) \,|\, (1,0,0) \vee (1,0,1) \vee (1,1,0) \vee (1,1,1) \vee (0,0,0) \vee (0,1,1)\}$. Entonces $\varphi_6 \in$ SAT, porqué existen asignaciones satisfactorias para el valor común 0 de la variable en común x_2. Las asignaciones satisfactorias son $\{(x_3, x_2, x_1, x_0) \,|\, (0,0,0,0) \vee (0,0,1,1)\}$.

Un ejemplo del caso 3), es φ_7 =SAT(4, 7) dado por $(x_3 \vee \overline{x}_2) \wedge (\overline{x}_3 \vee \overline{x}_2) \wedge (\overline{x}_3 \vee x_2) \wedge (x_2 \vee \overline{x}_1 \vee \overline{x}_0) \wedge (x_2 \vee \overline{x}_1 \vee x_0) \wedge (x_2 \vee x_1 \vee \overline{x}_0) \wedge (x_2 \vee x_1 \vee x_0)$. Se obtiene un SSAT(2, 3) de las tres primeras cláusulas de φ_7, cuya solución es$\{(x_3, x_2) \,|\, (0,0)\}$. Un SSAT(3, 4) corresponde con las últimas cuatro cláusulas de φ_7, cuyas soluciones son $\{(x_2, x_1, x_0) \,|\, (1,0,0) \vee (1,0,1) \vee (1,1,0) \vee (1,1,1)\}$. Entonces $\varphi_7 \notin$ SAT, porqué no es posible construir asignaciones satisfactorias, ya que no hay un valor común para la variable común x_2.

4. Resultados

El titulo del artículo indica que no se usa álgebra para resolver problemas SAT, pero otro aspecto importante es notar que esto conduce a una mayor complejidad en la resolución de un SAT.

Proposición 5. *Sea F una fórmula lógica y v una variable lógica, la cual no aparece en F. Entonces*

$$(F) \equiv \begin{array}{c}(F \vee v) \\ \wedge\,(F \vee \overline{v})\end{array}$$

Demostración. El resultado se tiene por las leyes de factorización y distribución del álgebra Booleana para $(F) \equiv (F \wedge (v \vee \overline{v}))$.

La proposición anterior permite la transformación recíproca entre SAT y SSAT. Por ejemplo, un SAT(4, 2) es

$$\begin{array}{c}(x_3 \vee \overline{x}_2 \quad\ \vee x_0) \\ \wedge \quad (x_2 \vee x_1 \vee \overline{x}_0).\end{array}$$

Se transforma en un SSAT(4, 4) aplicando la proposición anterior en:

$$\begin{array}{l}(x_3 \vee \overline{x}_2 \vee x_1 \vee x_0) \\ \wedge\,(x_3 \vee \overline{x}_2 \vee \overline{x}_1 \vee x_0) \\ \wedge\,(x_3 \vee x_2 \vee x_1 \vee \overline{x}_0) \\ \wedge\,(\overline{x}_3 \vee x_2 \vee x_1 \vee \overline{x}_0).\end{array}$$

Ambos son equivalentes y tienen el mismo conjunto de soluciones pero su transformación de uno en el otro como forma de resolución es más costosa que la simple lectura de sus cláusulas.

Por inspección del SAT(4, 2), las asignaciones que lo satisfacen requieren que $x_3 = 1$ y $x_1 = 1$ y las asignaciones de las otras variables no importan. Por otro lado las asignaciones satisfactorias del equivalente SSAT(4, 4) son $\{1010, 1011, 1110, 1111\}$ que justamente fijan los valores de las variables $x_3 = 1$ y $x_1 = 1$.

El realizar procesos algebraicos, como en el ejemplo anterior, muestra que se requiere de distribuir o factorizar, pero esto significa encontrar las partes comunes, ordenar las fórmulas y buscar términos comunes entre las m ecuaciones y las n variables. La revisión del estado del arte de los algoritmos para SAT muestran estrategias más complicadas, uso de grafos, ramificación y regreso a estado anterior (backtracking). Lo cual significa muchas más operaciones que la simple lectura de las cláusulas del problema.

Proposición 6. *Sea φ =SAT$(n, \cdot)$, una fórmula CNF con n variables ($n \gg 0$) y con un número muy grande y desconocido de cláusulas ($\gg 2^n$). Entonces el algoritmo 5 determina y estima una asignación satisfactoria en un máximo de tiempo entre* $\mathbf{O}(|\varphi| + |\times\theta|)$ *y* $\mathbf{O}(2^{n-1})$.

Demostración. Suponiendo suficiente tiempo y memoria para ejecutar el algoritmo 5, este llama a los algoritmos 3 y 4. Los cuales se ejecutan en paralelo e independientemente.

El algoritmo 3 ejecuta el algoritmo 2 para cada cláusula r y este actualiza la información del correspondiente $SSAT(IdSS(r))$. Durante la lectura de las cláusulas de φ puede ocurrir la detección de un $SSAT(IdSS(r))$ que corresponda con un tablero bloqueado y por tanto se terminan todo los procesos, ya que no hay asignación satisfactoria para φ (o sea $\varphi \notin SAT$). Al terminar de leer todas las cláusulas de φ sus correspondiente $SSAT(IdSS(r))$ están actualizados y se tienen sus soluciones en la lsita de los $SSAT(IdSS(r)).S$ Con estas soluciones se determina el conjunto Θ de soluciones de φ por medio de la operación $\times\theta$. Los $SSAT(IdSS(r))$ de la lista SSAT que no tienen variables en común unen sus soluciones por el producto cruz y los $SSATSSAT(IdSS(r))$ que tienen

variables en común, pegan sus resultados sobre los valores comunes de sus soluciones. Si $\Theta = \emptyset$ entonces $\varphi \notin SAT$, ya que las soluciones de los $SSAT(IdSS(r))$ son incompatibles. De otra forma, $\varphi \in SAT$, ya que $\Theta \neq \emptyset$ y cualquier $s \in \Theta$ es una asignación satisfactoria para φ. No es necesario calcular todas las soluciones con el operador $\times$ theta, solo se requiere crear una asignación satisfactoria o comprobar que las soluciones de los SSAT son incompatibles.

Por otra parte, el algoritmo 4 va tomando aleatoriamente dos candidatos del espacio de búsqueda $[0, 2^{n-1} - 1]$. Por ejemplo $x = 0T[k]$ and $\overline{x}$). Si alguno es una asignación satisfactoria por evaluación directa de φ como función en un cirtcuito (ver figura 1), entonces se tiene $\varphi \in SAT$, x o $\overline{x}$ es una asignación satisfactoria y todos los procesos se detienen. De otra forma, después de verificar que todos los elementos de $[0, 2^n - 1]$ no son satisfactorios, se comprueba que $\varphi \notin SAT$.

Finalmente, ambos algoritmos 3 y 4 tienen limites en sus iteraciones, el primero el número de cláusulas de φ más el tiempo que se tarda el operador $\times\theta$ y el segundo la exploración de $[0, 2^{n-1} - 1]$. El algoritmo 4, puede terminar si encuentra una asignación satisfactoria, pero sino, lo limita 2^{n-1} que es el número de candidatos tomados de dos en dos de $[0, 2^n - 1]$, aún si la fórmula tuviera una enorme cantidad de cláusulas $\gg 2^n$.

Una formulación muy difundida de problemas SAT es r, s-SAT, donde r es el número de variables por cláusula y s el número de veces que a lo más se repite una variable (afirmada o negada en las cláusulas).

Cualquiera de los r,1-SAT o r,2-SAT o r,r-SAT es resuelto por el algoritmo 5 con tiempo menor o similar a los algoritmos del estado del arte [Pud98,ZMMM01,ZM02,Tov84]. Note que se ha considerado que $\mathbf{O}(\times\theta \forall \text{SSAT}(IV(r))$ es estimado por estrategias de terminación rápida para la construcción de una asignación satisfactoria. Por ejemplo, para r,1-SAT se tiene $\bigcap_{r \in \varphi} IV(r) = \emptyset$ y una solución es el primer conjunto de valores del producto cruz $(\text{SSAT}(IV(r)))$. La complejidad es similar para r,2-SAT. Pero en este caso hay que construir una junta natural. Se tiene también, que los r,r-SAT se pueden resolver con el algoritmo 5. La conjetura 2.5 del artículo [Tov84] establece: *If $s \leq 2^{r-1} - 1$, then every instance r, s-SAT is satisfiable.* Si bien una demostración está fuera del alcance de este artículo, el algoritmo 5 puede ser usado para explorar instancias de casos muy complicados de esta conjetura.

5. Conclusiones y trabajos futuros

El resultado principal que se presenta es la determinación de cuando $\varphi \in$ SAT o $\varphi \notin$ SAT en no más de 2^{n-1} iteraciones y sin usar álgebra o estrategias con costo mayor a la lectura de las cláusulas de φ. Para determinar una asignación satisfactoria, no siempre se necesita la construcción de un candidato particular, bastan los parámetros n y m para problemas SSAT. Se sugiere que el operador $\times\theta$ use una estrategia de corto circuito para determinar la compatibilidad de los SSAT. Bajo las condiciones anteriores la complejidad es lineal respecto del número de cláusulas (m), que puede ser un parámetro opcional. Los problemas difíciles que se pueden plantear del tipo r, s-SAT pueden tener variables que aparecen $s = 2^{r-1} - 1$ (conjetura 2.5 del artículo [Tov84]) lo cual conduce a problemas donde el número de las cláusulas es enorme y posiblemente se tenga $m \gg 2^n$. Bajo las suposiciones de suficiente tiempo y espacio de memoria, cualquier instancia de un problema SAT en CNF puede ser resuelto por el algorimto 5.

Las nuevas técnicas del artículo se puede extender para los casos de instancias SAT con los operadores $\Rightarrow$ y $\Leftrightarrow$ y uso de paréntesis anidados. Básicamente el proceso que plantea el algoritmo propuesto es como un compilador lineal que va leyendo las cláusulas y construyendo el espacio de soluciones de los subproblemas SSAT asociados a las cláusulas, si no se logra una contradicción (o sea un tablero bloqueado de algún SSAT), con el espacio de las asignaciones satisfactorias y el operador $\times\theta$ se determina la compatibilidad o incompatibilidad de los subproblemas SSAT que componen la fórmula SAT φ dada. EL caso particular de SSAT se describe en mayor detalle en [Bar15,Bar16].

Los algoritmos presentados son realizables y pueden resolver casos muy difíciles con requerimientos de memoria y velocidad de ejecución muy altos, por lo que su implementación, así como la construcción de problemas usando la proposición 4 o las sugerencias de [GSTS07] requieren y demandan computadoras de computo intensivo numérico, supercomputadoras, novedosos sistemas híbridos hardware-software y que se continúen las investigaciones de la computación cuántica.

REFERENCIAS

[Bar05] Carlos Barrón-Romero. Minimum search space and efficient methods for structural cluster optimization. *arXiv*, Math-ph:0504030-v4, 2005.

[Bar10] Carlos Barrón-Romero. The Complexity of the NP-Class. *arXiv*, arxiv.org/abs/ 1006.2218, 2010.

[Bar15] Carlos Barrón-Romero. Classical and Quantum Algorithms for the Boolean Satisfiability Problem. *ArXiv e-prints*, October, 2015.

[Bar16] Carlos Barrón-Romero. Lower bound for the complexity of the boolean satisfiability problem. *ArXiv e-prints*, February, 2016.

[CLRS90] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 1990.

[Coo00] Stephen Cook. THE P VERSUS NP PROBLEM. *http://www.claymath.org/sites/default/files/pvsnp.pdf*, 2000.

[For09] Lance Fortnow. The Status of the P Versus NP Problem. *Commun. ACM*, 52(9):78–86, September 2009.

[GSTS07] Dan Gutfreund, Ronen Shaltiel, and Amnon Ta-Shma. If NP Languages Are Hard on the Worst-Case, Then It is Easy to Find Their Hard Instances. *Comput. Complex.*, 16(4):412–441, December 2007.

[JW] Gerhard J Woeginger. The P-versus-NP page. http://www.win.tue.nl/ gwoegi/P-versus-NP.htm.

[Pud98] Pavel Pudlák. *Mathematical Foundations of Computer Science 1998: 23rd International Symposium, MFCS'98 Brno, Czech Republic, August 24–28, 1998 Proceedings*, chapter Satisfiability — Algorithms and Logic, pages 129–141. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[Tov84] Craig A. Tovey. A simplified np-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85 – 89, 1984.

[Woe03] Gerhard J. Woeginger. Exact algorithms for np-hard problems: A survey. *Combinatorial Optimization - Eureka, You Shrink!, LNCS*, pages 185–207, 2003.

[ZM02] Lintao Zhang and Sharad Malik. *Computer Aided Verification: 14th International Conference, CAV 2002 Copenhagen, Denmark, July 27–31, 2002 Proceedings*, chapter The Quest for Efficient Boolean Satisfiability Solvers, pages 17–36. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[ZMMM01] Lintao Zhang, Conor F. Madigan, Matthew H. Moskewicz, and Sharad Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *Proceedings of the 2001 IEEE/ACM International Conference on Computer-aided Design*, ICCAD '01, pages 279–285, Piscataway, NJ, USA, 2001. IEEE Press.