Algoritmo numérico en paralelo para el problema de satisfacción lógica y su impacto sobre la clase NP

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: *Una nueva versión modificada de un algoritmo numérico y paralelo para resolver el problema de satisfacción lógica con cláusulas en forma conjuntiva normalizada es descrita. La forma de resolver el problema es sin usar algebra, ni estrategias de búsqueda computacionales como ramificación limitada, búsqueda adelante y atrás, representación por árboles, etc. El método se basa en una clase especial de problemas de satisfacción lógica, problema simple de satisfacción lógica. El diseño del principal algoritmo incluye ejecución paralela, orientación a objetos y terminación abrupta, como en la versión anterior, pero en esta versión se incluye guardar información de los casos fallidos resultado de la ejecución en paralelo para mejorar la eficiencia y favorecer la terminación abrupta. El resultado es un algoritmo lineal con respecto al número de cláusulas más un proceso de datos sobre las soluciones parciales de sub problemas simples de satisfacción lógica y con límite 2^n, donde n es el número de variables lógicas. La novedad de la solución es un algoritmo lineal, cuya complejidad es menor o igual que la complejidad de los algoritmos del estado del arte. La implicación para la clase NP es presentada en detalle.*

Palabras clave: Teoría de la Computación, Lógica, SAT, K-SAT, Complejidad de Algoritmos, Clase NP.

Abstract: *A novel modified numerical parallel algorithm for solving the classical Decision Boolean Satisfiability problem with clauses in conjunctive normal form is depicted. The approach for solving SAT is without using algebra or other computational search strategies such as branch and bound, back-forward, tree representation, etc. The method is based on the special class of problems, Simple Decision Boolean Satisfiability problem. The design of the main algorithm includes parallel execution, object oriented, and short termination as the previous versions but it keeps track of the parallel tested unsatisfactory binary values to improve the efficiency and to favor short termination. The resulting algorithm is linear with respect to the number of clauses plus a process data on the partial solutions of the Simple Decision Boolean Satisfiability problems and it is bounded by 2^n iterations where n is the number of logical variables. The novelty for the solution is a linear algorithm, such its complexity is less or equal than the algorithms of the state of the art. The implication for the class NP is depicted in detail.*

Keywords: Theory of Computation, Logic, SAT, K-SAT, Complexity of Algorithms, Class NP.

1. Introducción

La modelación para la resolución del problema de satisfacción lógica (SAT) se base en el concepto de reducción (ver[Bar10], capítulo 6). Este termino significa la habilidad de resolver un problema complicado mediante la resolución de problemas más simples. El algoritmo de este trabajo es resultado de aplicar a un problema SAT una reducción a subproblemas llamados simple SAT (SSAT).

La notación y convenciones para fórmulas lógicas son: los valores lógicos son 0 (falso) y 1 (verdadero). Los operadores lógicos son **not**: $\overline{x}$; **and**: $\wedge$, y **or**: $\vee$. De aquí en adelante, $\Sigma = \{0,1\}$ es el correspondiente alfabeto binario y X es el conjunto de n variables $\{x_{n-1}, \ldots, x_0\}$. Una cadena binaria $w \in \Sigma^n$ es identificada con un número binario en $[0, 2^n - 1]$ y recíprocamente. Además a una cláusula CNF, por ejemplo $x_{n-1} \vee \overline{x}_{n-2} \vee \ldots \overline{x}_1 \vee x_0$ le corresponde la cadena binaria b=10…01= $b_{n-1}b_{n-2}\ldots b_1b_0$ donde $b_i = \begin{cases} 0 & \text{si } \overline{x}_i, \\ 1 & \text{de otra forma.} \end{cases}$ Note que $\overline{b}$ es el número binario $\overline{b}_{n-1}\overline{b}_{n-2}\ldots\overline{b}_1\overline{b}_0$ donde los b_i son los dígitos de b.

El problema SAT consiste en determinar cuando una fórmula lógica φ, sin perdida de generalidad en forma conjuntiva normalizada (CNF), pertenece o no pertenece al lenguaje SAT ($\mathbb{L}$(SAT)), donde $\mathbb{L}$(SAT)= $\{\varphi \,|\, \varphi$ es una fórmula lógica para la existen valores booleanos que la satisfacen$\}$ o equivalentemente, existe un testigo $w \in \{0,1\}^n$ tal que $\varphi(w) \equiv 1$ donde n es el número de variables lógicas de φ. La principal característica de los problemas SAT es que sus cláusulas pueden usar n o menos variables. La principal característica de los problemas SSAT es que todas sus cláusulas usan n variables. Ambos problemas pueden tener muchas cláusulas repetidas y en desorden, i.e., con sus variables en cláusulas alejadas.

El que se limite el problema SAT a que las fórmulas φ tengan cláusulas en CNF está justificada por la equivalencia lógica entre fórmulas lógicas y la amplia literatura. Mencionar un algoritmo para resolver el problema SAT es inmediatamente relacionado con la famosa clase de problemas computacionales

NP y sus algoritmos [Pud98,ZMMM01,ZM02,Tov84], [Woe03,For09,GSTS07].

Los problemas clásicos están descritos como la satisfacción de fórmulas (k,n), más propiamente CNF (k,n)-satisfiability o (k,n)-SAT donde n es el número de variables, las cláusulas usan k variables y $n \geq k$. Por ejemplo, con $n = 7$ una fórmula de (3,7)-SAT es $(x_2 \vee \overline{x}_4 \vee x_6) \wedge (\overline{x}_0 \vee x_1 \vee x_2) \wedge (\overline{x}_1 \vee x_3 \vee \overline{x}_4)$.

El algoritmo del artículo es para cualquier tipo de SAT, esto significa que las cláusulas están en CNF con una o a lo más n variables . Por ejemplo, con $n = 8$, una fórmula arbitraria de SAT es $(x_4 \vee \overline{x}_5 \vee x_7) \wedge (\overline{x}_2 \vee \overline{x}_4) \wedge (x_0 \vee \overline{x}_1 \vee x_2) \wedge (x_1 \vee \overline{x}_3 \vee x_4 \vee \overline{x}_5)$. Este algoritmo es una versión mejorada del algoritmo especial paralelo para resolver cualquier tipo de SAT sin usar algebra [Bar16c,Bar16a].

Un resumen de los resultados anteriores de [Bar16c,Bar16b,Bar16a] es:

1. El espacio de soluciones (o espacio de búsqueda o espacio de las posibles asignaciones satisfactorias) es Σ^n.
2. Se determina con complejidad $\mathbf{O}(1)$ que una fórmula SSAT con n variables y m cláusulas pertenece a $\mathbb{L}(\text{SAT})$ (sin necesidad de testigo). La comparación de m versus 2^n es una condición suficiente para responder si pertenece o no pertenece a $\mathbb{L}(\text{SAT})$sin necesidad de más iteraciones o procesos o testigo.
3. Una fórmula SSAT no es satisfecha cuando tiene $m = 2^n$ cláusulas diferentes. Estos casos especiales de SSAT son denominados tableros bloqueados.

   Por ejemplo, en Σ y en Σ^2:

x_0
1
0

x_1	x_0
0	0
1	1
0	1
1	0

4. Si $\overline{b} \in \Sigma^n$ corresponde a la negación de la traducción a cadena binaria de una cláusula de una fórmula SSAT φ entonces es seguro que $\overline{b}$ no es una asignación satisfactoria para φ.
5. Algoritmos basados en SSAT para resolver problemas SAT son computables y no requieren procedimientos computacionales de clasificación y relación (matching) o de exploración de redes o árboles o de procedimientos de álgebra.

Los cambios con respecto a la versión anterior en [Bar16a] son:

Cooperación. Los algoritmos de búsqueda determinística 5 y de búsqueda aleatoria 7, comparten con un nuevo algoritmo 2, el registro de los candidatos fallidos, el cual busca secuencialmente una asignación satisfactoria.

Paralelismo Intensivo. El algoritmo de búsqueda aleatoria puede ejecutar paralelamente en 2^p procesadores pruebas de candidatos (algoritmo 6).

El artículo está organizado como sigue: La section 2 describe un resumen de propiedades y proposiciones en las que se base el algoritmo paralelo para resolver SAT (más información se tiene en [Bar16c,Bar16b,Bar16a]). La siguiente sección 3 contiene aspectos del diseño, propiedades y las rutinas del algoritmo paralelo. La sección 4 describe en detalle el análisis de complejidad y computabilidad de esta versión modificada. Es necesario volver a analizar la complejidad y la computabilidad, ya que los cambios en las rutinas pueden crear desde un incremento no esperado en el número de iteraciones o que el algoritmo sea no computable, i.e., no resuelva el problema. Por ejemplo sería un error incluir un ciclo de búsqueda de candidatos viables cuando el número de asignaciones satisfactorias es pequeño, ya que esto trae un consumo de iteraciones al desestimar la gran mayoría de posibles asignaciones satisfactorias en la búsqueda de un candidato viable. Esta sección destaca que el diseño con cooperación y con minima interacción en el registro de los candidatos fallidos no altera el funcionamiento y preserva la complejidad y eficiencia de la versión anterior. La siguiente sección 5 presenta los resultados. En particular la subsección 5.1 describe los resultados teóricos y las consecuencias que tiene el algoritmo paralelo para SAT en los algoritmos de la clase NP. Finalmente, la última sección presenta las conclusiones y trabajo futuro.

2. Propiedades de SAT

Por razones de espacio pero para facilitar la claridad se muestran en esta sección las principales propiedades y proposiciones sin demostración. Toda la información usada en la construcción del algoritmo paralelo se tiene en [Bar16c,Bar16b,Bar16a].

Proposición 1. *Cualesquiera cláusula CNF x sobre las variables $(x_{n-1}, \ldots, x_0)$ le corresponde un número binario $b = b_{n-1}b_n \ldots b_0$. Entonces $x(b) \equiv 1$ y $x(\overline{b}) \equiv 0$ donde b es la traducción de la cláusula CNF x y los valores de las variables de x se toman de las cadenas b y $\overline{b}$ de Σ^n. Por ejemplo, $x = (x_2 \vee \overline{x}_1 \vee x_0)$ le corresponde $b = 101$ y $\overline{b} = 010$, entonces $x(b) = (1 \vee \overline{0} \vee 1) \equiv 1$ y $x(\overline{b}) = (0 \vee \overline{1} \vee 0) \equiv 0$.*

La siguiente proposición es esencial para justificar el no usar álgebra o procedimientos computacionales complejos, por ejemplo de clasificación, búsqueda, relación y factorización, simplificación o expansión.

Proposición 2. *Sea F una fórmula lógica y v una variable, que no está en F. Entonces* $(F) \equiv \begin{array}{l}(F \vee v)\\ \wedge\, (F \vee \overline{v})\end{array}$.

Por ejemplo, $\varphi_3 = \begin{array}{l}(x_3 \vee \overline{x}_2 \quad\quad \vee x_0)\\ \wedge \quad\quad (x_2 \vee x_1 \vee \overline{x}_0)\end{array}$ es equivalente por la proposición anterior con

$$\varphi_4 = \begin{array}{l}(x_3 \vee \overline{x}_2 \vee x_1 \vee x_0)\\ \wedge\, (x_3 \vee \overline{x}_2 \vee \overline{x}_1 \vee x_0)\\ \wedge\, (x_3 \vee x_2 \vee x_1 \vee \overline{x}_0)\\ \wedge\, (\overline{x}_3 \vee x_2 \vee x_1 \vee \overline{x}_0).\end{array}$$

Mediante factorizaciones y simplificaciones, se pueden mostrar que φ_3 tiene asignaciones satisfactorias que necesitan $x_3 = 1$ y $x_1 = 1$. Mientras que φ_4 tiene como asignaciones satisfactorias a {1010, 1011,

1110, 1111}. Pero ambas se corresponden. Este pequeño ejemplo puede ser verificado realizando el álgebra de las leyes de factorización y distribución, con el ordenamiento de cláusulas para la identificación de las variables o valores comunes, lo cual requiere de ordenar y relacionar, lo cual es más costoso que la lectura de las cláusulas que es la operación que realiza el algoritmo de búsqueda determinística 5 del algoritmo paralelo 8.

Por otro lado, una revisión de los algoritmos del estado del arte para resolver SAT muestra que ellos usan procedimientos computacionales sofisticados y complejos, requieren por ejemplo, ramificación y limite (branch and bound), retroceso (backtracking), ordenamiento y relación (sorting and matching), etc. Esto significa mayor número de iteraciones que las que se requieren para una lectura de las cláusulas.

3. Algoritmos para SAT

El primer algoritmo es para una función de 2^X en $[0, 2^n - 1]$ que bijectivamente relaciona un conjunto de variables con un único número para la identificación de subproblemas SSAT.

Algoritmo 1 *Id_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 índices 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;

Los siguientes algoritmos son las versiones modificadas de [Bar16a] que incluyen cooperación y mínima interacción para el registro de los candidatos fallidos y la posibilidad de paralelismo intensivo en la rutina de búsqueda aleatoria 7 modificada apropiadamente.

Algoritmo 2 *Actualizar_cand_fallido*
Entrada: *n : número de variables y* φ *fórmula del problema;*
Recepción de mensajes *c: número* $\in \Sigma^n$*;*
Hacer: *get_in(c, L_C); //se recibe en la lista L_C*
Salida: *nada o mensaje y terminación abrupta.*
Memoria:
  L_c : Lista de números binarios;
  n_cand:= 2^n *: entero;*
  L_cand_stat$[0, 2^{n-1}] := 1$*: arreglo de valores en* Σ*, doblemente encadenado para manejarlo como una lista circular; // 1: viable, 0: no satisface* φ
  next:=0: entero // apuntador a lista L_cand_stat;

---

mientras (1) ***do***
  mientras *not empty (L_c)* ***do***
    c := get_out(L_c);
    si *(L_cand_sta[c] == 0)* ***entonces***
    // nada que hacer ya fue actualizado
      continuar;
    fin si
    L_cand_sta[c] := 0;
    Actualizar_lista_circular(L_cand_sta, c, next);
    n_cand := n_cand - 1;
    si *(n_cand == 0)* ***entonces***
      Salida: *"El algoritmo 2 confirma que* $\varphi \notin \mathbb{L}(\text{SAT})$ *después de revisar* Σ^n*.";*
      alto total;
    fin si
  fin mientras
  si *(n_cand > 0)* ***entonces***
    c := next;
    si $(\varphi(c) == 1)$ ***entonces***
      Salida: *"El algoritmo 2 confirma que* $\varphi \in \mathbb{L}(\text{SAT})$*,*
      c es una asignación satisfactoria.";
      alto total;
    fin si
  L_cand_sta[c] := 0;
  Actualizar_lista_circular(L_cand_sta, c, next);
  n_cand := n_cand - 1;
  si *(n_cand == 0)* ***entonces***
    Salida: *"El algoritmo 2 confirma que* $\varphi \notin \mathbb{L}(\text{SAT})$ *después de revisar* Σ^n*.";*
    alto total;

fin si
fin mientras
regresa

El algoritmo 2 está diseñado para ejecutar en su propio procesador con su memoria exclusiva y comunicarse con los otros por medio de mensajes que se procesan por medio de una lista. Cada que termina de procesar mensajes, prueba un candidato para terminación abrupta o continuar. La comunicación no necesita ser segura o con garantía de no perdida de mensajes. Interactúa en forma minima sin detener a otros procesos porque se ejecuta en procesador propio que maneja su memoria en forma independiente y exclusiva. La perdida de mensajes no es relevante, pero si lo es la actualización en forma exclusiva de la lista circular L_cand_sta y el contador n_cand, ya que esto garantiza que la condición (n_cand == 0) se cumple después de revisar todo Σ^n aún con perdida de mensajes o candidatos no satisfactorios repetidos.

Note que el algoritmo 2 no usa memoria dinámica, i.e., no realiza pedidos de memoria o corrimientos de datos, sino que usa un arreglo con apuntadores de valores enteros para encadenar las asignaciones que aún pueden ser satisfactorias cuando se ejecuta Actualizar_lista_circular.

La siguiente rutina completa con valores aleatorios de Σ una asignación de una cláusula traducida con menos de n variables.

Algoritmo 3 *$Número_{\Sigma^n}$*
Entrada: *(rw: conjunto de valores enteros que identifican las variables de una cláusula, rv: conjunto de valores de Σ de cada variable lógica identificada en rw)*
Salida: *(k_Σ: cadena de Σ^n).*
Memoria:
i : entero;

---

itera $i := n-1$ **decrementa** 0 **hacer**
  si ($i \in rw$) **entonces**
    $k_\Sigma[i]$:= **valor de** rv **de la variable** i;
  de otra forma
    $k_\Sigma[i]$:= *selección aleatoria en* Σ;
  fin si
fin itera
regresa k_Σ;

Algoritmo 4 *Actualizar_SSAT*
Entrada: *(SSAT: Lista de objetos, rw: cláusula).*
Salida: *SSAT: Lista actualizado de los objetos SSAT, particularmente el Id_SSAT(r).*
Cada SSAT(Id_SSAT(r)) actualiza su lista de soluciones S, donde $S[0:2^{n-1}]$: es una lista circular en un arreglo doblemente encadenado de enteros;
Memoria: *$ct := 0$: entero; k, k_aux: entero;*

---

si (*Id_SSAT(rw)* $\notin$ *SSAT*) **entonces**
  construir objeto *SSAT(Id_SSAT(rw));*
fin si
con *SSAT(Id_SSAT(rw))*
  k := **cláusula a binario** *(rw);*
  k_aux := k;
  si *(size(k)* $< \varphi.n$) **entonces**
    k_aux := $Número_{\Sigma^n}$(set_of_variables(rw), k);
    // Algoritmo 3
  fin si
  si (φ(*k_aux*) $== 1$) **entonces**
    Salida: *"El algoritmo 4 confirma que*
    $\varphi \in$L(SAT),
    k_aux es una asignación satisfactoria.";
    alto total;
  de otra forma
    Actualizar_cand_fallido(k_aux); // Algoritmo 2
  fin si
  si *(size(k)* $== \varphi.n$)
    Actualizar_lista_circular(S,k, ct);
    // Actualiza la lista circular y el contador ct
    si (ct $= 2^n$) **entonces**
      output: *" El algoritmo 4 confirma que*
      $\varphi \notin$L(SAT).
      SSAT(Id_SSAT(rw))
      es un tablero bloqueado.";
      alto total;
    fin si
    // $\overline{k}$ de tamaño n no satisface φ.
    // Ver proposición 1.
    Actualizar_cand_fallido($\overline{k}$); // Algoritmo 2
  fin si
  Actualizar_lista_circular(S,$\overline{k}$, ct);
  // Actualiza la lista circular y el contador ct
  si (ct $= 2^n$) **entonces**
    output: *" El algoritmo 4 confirma que*
    $\varphi \notin$L(SAT).
    SSAT(Id_SSAT(rw))
    es un tablero bloqueado.";
    alto total;
  fin si
fin con
regresa

En la versión anterior del algoritmo 4 (ver [Bar16a,Bar16b]) se muestra que no usa memoria dinámica y que con apuntadores de números enteros se construye el espacio de las asignaciones satisfactorias en una lista circular. Aquí se omite la actualización y manejo de los apuntadores, en su lugar aparece un llamado a Actualizar_lista_circular.

Algoritmo 5 *Búsqueda determinista para resolver ¿$\varphi \in$L(SAT)?*
Entrada: *n : número de variables y φ fórmula del problema.*
Salida: *Mensaje y si hay solución el testigo x, tal que $\varphi(x) \equiv 1$.*
Memoria:

r: conjunto de variables de X;
SSAT:=null: Lista de objetos SSAT.

---

mientras *not(eof_clauses(φ));*
  r = φ.read_clause;
  Actualizar_SSAT(SSAT,r). // Algoritmo 4
fin mientras;
con lista *SSAT*
  // ×θ es producto cruz y junta natural
  calcula $\Theta = \times\theta\ \forall\ SSAT(Id_SSAT(r))$;
  si $\Theta = \emptyset$ **entonces**
    Salida: *"El algoritmo 5 confirma que* $\varphi \notin \mathbb{L}(\mathrm{SAT})$.
    Los SSAT(Id_SSAT(r)) son incompatibles."
    alto total;
  de otra forma
    Salida: *"El algoritmo 5 confirma que* $\varphi \in \mathbb{L}(\mathrm{SAT})$.
    s es una asignación satisfactoria, $s \in \Theta$.
    Los SSAT(Id_SSAT(r)) son compatibles.
    alto total;
  fin con
fin si

Algoritmo 6 *Probar_φ(·)*
// Evalua un candidato y actualiza candidatos fallidos.
Entrada: *c: cadena de* Σ^n.
Salida: *none.*

---

si $(\varphi(c) == 1)$ **entonces**
  Salida: *"Los algoritmos 6 y 7 confirman que* $\varphi \in \mathbb{L}(\mathrm{SAT})$.
  c es una asignación satisfactoria.";
  alto total;
de otra forma
  Actualizar_cand_fallido(c); // Algoritmo 2
fin si
regresa

El siguiente algoritmo realiza una búsqueda aleatoria cuyos candidatos son seleccionados arbitrariamente de Σ^n (identificados como números binarios de $[0, 2^n - 1]$). Para mantenerlo eficiente se considera: 1) que la búsqueda aleatoria sea sobre una permutación aleatoria del espacio de asignaciones $[0, 2^n - 1]$ y 2) que la permutación no implique un costo adicional de iteraciones, ya que es posible realizarla al vuelo, i.e., al mismo tiempo de la selección de candidatos.

Algoritmo 7 *Búsqueda aleatoria en* $[0, 2^n - 1]$ *para resolver ¿*$\varphi \in \mathbb{L}(\mathrm{SAT})$*?*
Entrada: *n : número de variables y φ fórmula del problema.*
Salida: *Mensaje y si hay solución, el testigo s* ($s \in [0, 2^n - 1]$), *tal que* $\varphi(s) \equiv 1$.
Memoria: $T[0 : 2^{n-1} - 1]$=$[0 : 2^n - 1]$*: entero;* $Mi = 2^n - 1$*: entero; rdm, a: entero.*

---

itera $i{:=}0$ *to* $2^{n-1} - 2$
  si $T[i] = i$ **entonces**
    // selección aleatoria de $rdm \in [i+1, 2^{n-1} - 1]$;
    $rdm = floor(rand()\ (Mi - i + 1{,}5)) + (i + 1)$;
    // rand(): número real aleatorio en (0,1);
    // floor(x): entero menor que x
    $a = T[rdm]$;
    $T[rdm] = T[i]$;
    $T[i] = a$;
  fin si
  ejecución en paralelo
    // Concatenación de 0 y $T[i]$;
    // Concatenación de 1 y $T[i]$;
    *Probar_*$\varphi(0T[i])$ *// Algoritmo 6;*
    *Probar_*$\varphi(1T[i])$ *// Algoritmo 6;*
  fin ejecución en paralelo
fin itera
// caso final
  ejecución en paralelo
    // Concatenación de 0 y $T[2^{n-1} - 1]$;
    // Concatenación de 1 y $T[2^{n-1} - 1]$;
    *Probar_*$\varphi(0T[2^{n-1} - 1])$ *// Algoritmo 6;*
    *Probar_*$\varphi(1T[2^{n-1} - 1])$ *// Algoritmo 6;*
  fin ejecución en paralelo
Salida: *"El algoritmo 7 confirma que* $\varphi \notin \mathbb{L}(\mathrm{SAT})$ *después probar todas las cadenas de* Σ^n*.";*
alto total;

En el diseño de la rutina anterior se obtuvo una disminución de las iteraciones al tomar de dos en dos los 2^n candidatos y paralizar la evaluación de $\varphi(\cdot)$, por lo que el limite superior de iteraciones es 2^{n-1}. El número de posibles candidatos no se altera, son los 2^n de Σ^n.

La disminución en tiempo es bajo la suposición de que las 2^p evaluaciones de $\varphi(\cdot)$ son simultaneas en procesadores independientes. Para 2^1 procesadores se tiene $\frac{2^n}{2^1} = 2^{n-1}$. Es posible disminuir el tiempo, i.e., recorrer más rápido el espacio de las asignaciones Σ^n, por ejemplo, con 4 (2^2) procesadores independientes, las iteraciones son $\frac{2^n}{2^2} = 2^{n-2}$ y los 4 candidatos simultáneos para probar son $00x$,$01x$, $10x$ y $11x$ donde $x \in [0, 2^{n-2} - 1]$. Se elige un número 2^p de procesadores por la facilidad de dividir el espacio Σ^n. Las modificaciones apropiadas para que el algoritmo 7 se ejecute en paralelo intensivamente corresponden a la reducción del espacio de asignaciones posibles y sus concatenaciones con las cadenas de Σ^p. En general con 2^p procesadores independientes el limite superior de iteraciones es $\frac{2^n}{2^p} = 2^{n-p}$, sin embargo esto no es necesariamente una mejora del tiempo indisputable, la proposición 8 lo demuestra. Básicamente porque el número de procesadores 2^p no crece a la par de 2^n si no muy por debajo del número de variables de un posible problema SAT, enorme y arbitrario, con $n \gg 0$ un gran numero.

Los algoritmos modificados 5 y 7 mantienen en su diseño la terminación abrupta cuando encuentran una asignación satisfactoria o cuando se determina que hay

un tablero bloqueado no importando el número de cláusulas o si estas se repiten o están desordenadas o el número de variables del problema. Pero además ellos cooperan con el algoritmo 2 enviando sus candidatos fallidos por mensajes, por lo que el espacio de candidatos posibles que mantiene el algoritmo 2 decrece más rápido por iteración por aproximadamente el factor 2^{p+1} (el número de mensajes que se le envían).

Algoritmo 8 *Algoritmo paralelo para SAT*
Entrada: *n : número de variables y φ fórmula del problema.*
Salida: *Mensaje que confirma si $\varphi \in \mathbb{L}(\text{SAT})$ o no.*

---

ejecución en paralelo
algoritmo 2(n, φ);
algorithm 5(n, φ);
algorithm 7(n, φ);
fin ejecución en paralelo

---

4. Análisis de la complejidad y de la computabilidad del algoritmo paralelo 8

Suponiendo que se tienen maquinas de Turing con cinta de memoria infinita para ejecutar el algoritmo paralelo se analizaran por separado:

1. El algoritmo 5 para la búsqueda determinística y su procesamiento de datos del operador $\times\theta$.
2. El algoritmo 7 para búsqueda aleatoria mediante 2^p procesos en paralelo.
3. El algoritmo 2 que registra los candidatos fallidos de los algoritmos 5 y 7 y busca secuencialmente un candidato satisfactorio.

Proposición 3. *El algoritmo 5 para la búsqueda determinística y su procesamiento de datos del operador $\times\theta$ es computable y sus iteraciones están limitadas por m (número de cláusulas de φ) más las iteraciones del operador $\times\Theta$.*

Demostración. Este algoritmo ejecuta los algoritmos 1, 4 y 3, que se pueden realizar por Maquinas de Turing sencillas para sumar, multiplicar, copiar, identificar, rellenar y cuya ejecución antes del operador $\times\Theta$ solo se detiene cuando al evaluar $\varphi(\cdot)$ se encuentra una asignación satisfactoria o cuando se identifica un tablero bloqueado. Cuando una cláusula de tamaño n no es satisfactoria, se descartan dos candidatos (k y $\overline{(k)}$), de otra forma solo se descarta uno, el complementario de la cláusula (ver proposición 1). Los mensajes de los casos fallidos que le envía al algoritmo 2 no causan retraso o la detención de este algoritmo porque se envían sin protocolo de comunicación de verificación de envío y recepción. Si no se detiene al terminar de leer las cláusulas de φ, el algoritmo 4 construyo las asignaciones satisfactorias o soluciones de subproblemas SSAT que deben ser conciliadas mediante el operador $\times\Theta$ para las n variables de φ. El operador $\times\Theta$ funciona igual que los operadores de bases de datos relacionales producto cruz y junta natural. Para los conjuntos de variables r y r' y las asignaciones satisfactorias $SSAT(\cdot).S$, se tiene que $SSAT(Id_SSAT(r)) \times\theta$ $SSAT(Id_SSAT(r')) =$

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

Las soluciones de $SSAT(\cdot).S$ de los casos 1) y 2) son compatibles y existe una asignación satisfactoria. Para el caso 3) las soluciones de los $SSAT(\cdot).S$ son incompatibles, no hay asignación satisfactoria ya que el resultado del operador $\times\Theta$ es el conjunto vacío. Las iteraciones que realiza este algoritmo son proporcionales al número de cláusulas de φ y al calculo del operador $\times\Theta$.

El algoritmo 5 se comporta como un compilador de una pasada con terminación abrupta y sólo requiere inspeccionar las cláusulas del φ. A continuación se dan ejemplos de los casos del operador $\times\Theta$.

De 1) $\varphi_5 = \text{SAT}(4,5)$ con $(x_3 \vee \overline{x}_2) \wedge (x_3 \vee x_2) \wedge (\overline{x}_3 \vee x_2) \wedge (\overline{x}_1 \vee \overline{x}_0) \wedge\ (x_1 \vee x_0)$. Tiene un SSAT$(2,3)$ de las 3 primeras cláusulas, con soluciones $x_3 = 1, x_2 = 1$. Tiene un SSAT$(2,2)$ de sus últimas dos cláusulas con soluciones $\{(x_1, x_0) \,|\, (0,1) \vee (1,0)\}$. Entonces las asignaciones satisfactorias para φ_5 son $\{(1,1)\} \times \{(0,1), (1,0)\} = \{(x_3, x_2, x_1, x_0) \,|\, (1,1,0,1) \vee (1,1,1,0)\}$.

De 2) $\varphi_6 = \text{SAT}(4,5) = (x_3 \vee \overline{x}_2) \wedge (\overline{x}_3 \vee \overline{x}_2) \wedge (\overline{x}_3 \vee x_2) \wedge (x_2 \vee \overline{x}_1 \vee x_0) \wedge (x_2 \vee x_1 \vee \overline{x}_0)$. Tiene un SSAT$(2,3)$ de sus tres primeras cláusulas con soluciones $\{(x_3, x_2) \,|\, (0,0)\}$. Tiene un SSAT$(3,2)$ de sus 2 últimas cláusulas con soluciones $\{(x_2, x_1, x_0) \,|\, (1,0,0) \vee (1,0,1) \vee (1,1,0) \vee (1,1,1) \vee (0,0,0) \vee (0,1,1)\}$. Entonces $\varphi_6 \in \mathbb{L}(\text{SAT})$, porqué hay asignaciones satisfactorias para el valor común 0 de la variable común x_2. Las asignaciones satisfactorias de φ_6 son $\{(x_3, x_2, x_1, x_0) \,|\, (0,0,0,0) \vee (0,0,1,1)\}$.

De 3) $\varphi_7 = \text{SAT}(4,7) = (x_3 \vee \overline{x}_2) \wedge (\overline{x}_3 \vee \overline{x}_2) \wedge (\overline{x}_3 \vee x_2) \wedge (x_2 \vee \overline{x}_1 \vee \overline{x}_0) \wedge (x_2 \vee \overline{x}_1 \vee x_0) \wedge (x_2 \vee x_1 \vee \overline{x}_0) \wedge (x_2 \vee x_1 \vee x_0)$. Tiene un SSAT$(2,3)$ de sus 3 primeras cláusulas con soluciones $\{(x_3, x_2) \,|\, (0,0)\}$. Tiene un SSAT$(3,4)$ de sus 4 últimas cláusulas con soluciones $\{(x_2, x_1, x_0) \,|\, (1,0,0) \vee (1,0,1) \vee (1,1,0) \vee (1,1,1)\}$. Entonces $\varphi_7 \notin \mathbb{L}(\text{SAT})$, porqué no hay valor común para la variable común x_2.

Proposición 4. *El algoritmo 7 para la búsqueda aleatoria de la solución de SAT es computable y sus iteraciones están limitadas por 2^{n-p} cuando se usan 2^p procesadores independientes para el algoritmo 6.*

Demostración. Este algoritmo ejecuta instrucciones fáciles de realizar por medio de maquinas de Turing apropiadas: generación de números naturales e identificación. El punto crucial es la paralelización del algoritmo probar 6 que realiza la prueba de candidatos

$\varphi(c) == 1$ cuyo efecto en caso de cumplirse es el de terminar abruptamente todo porque se tiene una asignación satisfactoria. Y en otro caso no hay causas para aumentar el tiempo de ejecución o de detener su proceso al enviar el candidato fallido al algoritmo 2, ya que esto se realiza sin protocolo de comunicación y verificación de envío y recepción. El efecto de tener 2^p procesadores es el dividir el espacio de asignaciones de Σ^n. O sea, $\frac{2^n}{2^p} = 2^{n-p}$.

Es importante notar que la aleatoriedad de la selección de candidatos del intervalo $[0, 2^{n-p} - 1]$ se pierde con p grande porque corresponde a cadenas en orden, por ejemplo, para $p = 2$, se tiene $00, 01, 10, 11$ de los p bit que completan a n. Este algoritmo proporciona en una iteración en paralelo 2^p candidatos.

Proposición 5. *El algoritmo 2 que registra a los candidatos fallidos es computable y sus iteraciones están limitadas por 2^n.*

Demostración. Este algoritmo ejecuta instrucciones fáciles de realizar por medio de maquinas de Turing apropiadas: generación de números naturales, identificación y simulación de una cola de servicio. Salvo por la parte de recepción de mensajes su cuerpo es similar al del algoritmo de búsqueda aleatoria pero sin la permutación. Este toma en orden secuencial los candidatos de su lista circular de registro de candidatos que no han sido marcados como fallidos. Cuando verifica $\varphi(c) \equiv 1$ termina abruptamente porque se tiene una asignación satisfactoria. No hay causas para aumentar el tiempo de ejecución o de detener su proceso al recibir candidatos fallidos de los otros algoritmos porque esto se realiza sin protocolo de comunicación y verificación de envío y recepción. Además el ciclo de atención a la cola de mensajes es siempre finito. Si este algoritmo no recibiera ningún candidato fallido entonces revisa todo el espacio de asignaciones de Σ^n lo que le toma 2^n iteraciones.

Proposición 6. *El algoritmo paralelo 8 es computable y su complejidad esta limitada superiormente por 2^n iteraciones.*

Demostración. El resultado se sigue de las proposiciones anteriores para los algoritmos 2, 5 y 7.

Para cualquier problema SAT, el algoritmo paralelo 8 contempla dos únicas respuestas: 1) $\varphi \in \mathbb{L}(\text{SAT})$, porqué existe $x \in \Sigma$ tal que $\varphi(x) \equiv 1$, o 2) $\varphi \notin \mathbb{L}(\text{SAT})$, porqué no existe $x \in \Sigma^n$ que satisface $\varphi(\cdot)$.

El algoritmo determinista 5 contempla dos situaciones de terminación abrupta que no requieren la inspección de todas las cláusulas: 1) encontrar un tablero bloqueado, es decir, un subproblema SAT sin solución y 2) encontrar una cláusula cuya traducción a binario sea un arreglo de valores de Σ que satisface SAT. Cuando esto no pasa, se realizo una revisión de todas sus cláusulas y se requiere ejecutar la operación $\times\Theta$ para conciliar las soluciones de los subproblemas SSAT. La operación $\times\Theta$ no necesita construir todas las asignaciones satisfactorias, solo se requiere una.

El algoritmo 7 cuando se paraleliza con 2^p procesadores explora Σ^n en 2^{n-p} iteraciones. Pero con $p \ll n$ para un problema SAT enorme, se tiene que $2^{n-p} \approx 2^n$.

Bajo tales consideraciones, al algoritmo paralelo 8 le toma un tiempo proporcional al tamaño del problema SAT más el costo de la operación $\times\Theta$ pero en el peor de los casos, es decir, cuando no se produce una terminación abrupta, los algoritmos 2 y 7 exploran Σ^n en a lo mas 2^n iteraciones.

La probabilidad de encontrar una asignación satisfactoria después de k candidatos fallidos es

$$\mathcal{P}(s, k) = \frac{s}{2^n - k}.$$

El divisor exponencial 2^n (que corresponde al número de combinaciones de Σ^n) ocasiona un decaimiento muy rápido como se muestra en la figura 1 donde $k = 2^n - 2$, $k = 500,000$ y $k = 1$. Esto significa que cuando $n \gg 0$ es un enorme número, entonces solamente después de intentar un número similarmente grande $k = 2^n - 2$ de candidatos diferentes la probabilidad crece a 0,5. Mientras que para un número razonable de pruebas $k \ll 2^n$ de candidatos diferentes, la probabilidad permanece insignificante, i.e., $\mathcal{P}(1, k) \approx \frac{1}{2^n} \approx 0$.

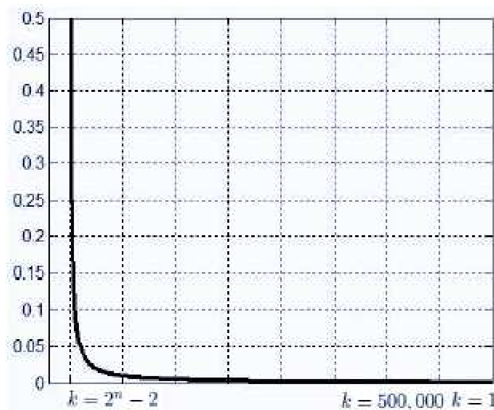

Figura 1. $\mathcal{P}(s, k)$ donde s es el número de asignaciones satisfactorias y k es el número de candidatos fallidos

Proposición 7. *Para cualquier problema SAT se cumple: $\mathcal{P}_2(k, s) \leq \mathcal{P}_5(k, s) \leq \mathcal{P}_7(k, s)$*

donde $\mathcal{P}_i(k, s)$ es la probabilidad obtener la solución (s es el número de soluciones o asignaciones satisfactorias) del algoritmo i después de k iteraciones.

Demostración. Por cálculo directo, la probabilidad del algoritmo 5 de búsqueda determinística es:

$$\mathcal{P}(s, k) = \frac{s}{2^n - k}.$$

Para el algoritmo 7 de búsqueda aleatoria con 2^p procesadores en paralelo es:

$$\mathcal{P}(s, k) = \frac{s}{2^n - k2^p}.$$

Y finalmente para el algoritmo 2 que actualiza los candidatos es aproximadamente (puede ser menor por mensajes perdidos o por los casos fallidos y repetidos que le envíen los algoritmos 5 y 7):

$$\mathcal{P}(s,k) \approx \frac{s}{2^n - k(2^p+2)}.$$

Se denomina problema SAT extremo a un problema SAT con $n \gg 0$ variables, donde sus cláusulas usan como máximo n variables, las cláusulas podrían repetirse y en desorden, pero la característica más importante es que un problema extremo tiene una o ninguna solución. Significa en el caso de una solución que la probabilidad de encontrar la solución es $\frac{1}{2^n}$ y para ninguna solución es 0. Por lo tanto, para una serie de $M > 0$ problemas SAT extremos, el valor esperado para solucionarlo es casi 0.

Proposición 8. *Para un problema SAT extremo con $n \gg 0$ un número enorme de variables lógicas. Entonces el algoritmo 8 y sus algoritmos 2, 5 y 7 requieren de alrededor 2^n iteraciones y no mejoran la eficiencia.*

Demostración. El algoritmo paralelo 8 tiene como complejidad el mínimo número de iteraciones de los algoritmos 2, 5 y 7.

Por la proposición 7 y al tener 2^p procesadores el algoritmo con mayor probabilidad de resolver el problema es el algoritmo 2.

Después de k iteraciones se tiene una exploración de $2^n - k(2^p+2)$ candidatos y no se ha encontrado la asignación satisfactoria porque al ser n un número muy grande y el problema SAT extremo, el termino $k(2^p+2)$ es pequeño ya que, 2^p es el número de procesadores en paralelo que es pequeño contra 2^n y k el número de iteraciones que es mucho menor a 2^n. Así para un problema extremo se tiene que $2^n - k(2^p+2) \approx 2^n$. Por otro lado, el algoritmo 7 con 2^p procesadores realiza la exploración de Σ^n en 2^{n-p} iteraciones, pero como $p \ll n$, las iteraciones que realiza son aproximadamente 2^n.

Lo cual significa que la probabilidad de resolverlo se mantiene inalterada y casi nula $\mathcal{P}(s,k) = \frac{s}{2^n - k2^p} \approx \mathcal{P}(s,k) = \frac{s}{2^n} \approx 0$ para k y 2^p pequeños comparados con 2^n donde $|s| \leq 1$.

Además para el caso $s = 0$ la única forma de conocer que no existe una asignación satisfactoria es explorando todo Σ^n por lo que no hay forma de mejorar la eficiencia.

5. Resultados

Cualquiera de las formulaciones $(r,1)$-SAT o $(r,2)$-SAT o (r,r)-SAT se resuelve mediante el algoritmo 8 en tiempo lineal con respecto al número de cláusulas de φ. Por lo tanto, su eficiencia es menor o igual que los algoritmos para SAT del estado del arte [Pud98,ZMMM01,ZM02,Tov84].

5.1. Implicaciones del algoritmo paralelo para SAT y los problemas NP

En esta apartado se analiza la clase NP como en matemáticas se analiza la incompletud de la clase de los números racionales para resolver ciertos problemas geométricos. La clase de los números racionales no es suficiente para resolver la distancia de la diagonal de un triángulo rectangular de lados 1, ya que la diagonal es de tamaño $\sqrt{2}$, que no es un número racional. Una demostración para verificar que $\sqrt{2}$ no es racional, es por contradicción, suponiendo que $\sqrt{2} = \frac{p}{q}$ es un número racional y que se puede factorizar de forma unívoca por la división de dos enteros sin factores comunes, es decir p y q se pueden identificar como primos relativos. Al analizar la fórmula equivalente, $2 = \frac{p^2}{q^2}$, se obtiene la contradicción que p y q son pares y no primos relativos.

Proposición 9. *No hay un algoritmo eficiente para resolver un problema SAT extremo.*

Demostración. Supongamos que existe un algoritmo eficiente para resolver cualquier problema SAT. Obviamente, un problema SAT extremo debe ser resuelto por tal algoritmo.

Se seleccionan dos personas, la persona número uno define el problema SAT extremo con la libertad de decidir una o ninguna solución. La segunda persona tiene una computadora potente y cuenta con el algoritmo eficiente.

¿Cuánto tiempo le llevará a la segunda persona resolver una serie de M problemas SAT extremos? Tiene el algoritmo eficiente, por lo que tiene que dar las dos respuestas posibles en corto tiempo. La asignación de valores satisfactorios o que no hay solución.

Si el algoritmo eficiente falla, no importa el tiempo, es inútil.

Por otro lado, por razones de argumentación, supongamos que la segunda persona con su poderosa computadora y el eficiente algoritmo consigue la solución de una serie de M problemas SAT extremos en un tiempo razonable. El valor de M debe ser un valor grande y razonable para la percepción humana pero con $M \ll 2^n$. Hay una contradicción. Una sucesión de M éxitos consecutivos de problemas SAT extremos significa que la primera persona no decide libremente y arbitrariamente los problemas SAT. El valor esperado para solucionar honestamente una serie de M problemas SAT extremos es 0.

Con colegas y estudiantes, propuse una argumentación alternativa. Una compañía de lotería define su boleto ganador por un problema SAT extremo. Cuando los problemas SAT extremos tienen una solución única, nadie se queja. El boleto del ganador satisface a todos porque, la verificación se hace en tiempo eficiente, y hay un testigo, el boleto ganador. Pero, cuando el problema SAT extremo no tiene solución. No hay ganador. Entonces es difícil de aceptar, porque con un

gran número n, se requiere de mucho tiempo para verificar que ninguno de los 2^n boletos satisface el problema SAT extremo. Y sin esta verificación, el resultado de que no haya ganador recae en la honestidad de la compañía de lotería.

Otro aspecto, es la linealidad del algoritmo con respecto al número de cláusulas SAT. Por supuesto, es lineal, pero ninguna persona o computadora puede responder a SAT sin revisar las cláusulas del SAT. Por otro lado, ¿cómo puede construirse un problema SAT extremo si el número de sus cláusulas es exponencial y alrededor o superior a 2^n. Bueno, aquí tengo tres posiciones: 1) es una hipótesis, una suposición teórica válida, como la de que $\sqrt{2}$ es racional y factorizable unívocamente mediante primos relativos; 2) los avances en la investigación de nanotecnología, clusters de moléculas y estructuras cristalinas pronto proporcionarán circuitos electrónicos capaces y complejos como SAT extremos, y 3) se puede realizar un experimento práctico utilizando la equivalencia lógica entre CNF y DNF (Forma normal disyuntiva). Un problema SAT extremo en DNF es el conjunto vacío de cláusulas o un número binario como una cláusula DNF. Por ejemplo si la solución única de un problema SAT extremo es 001, entonces la cláusula DNF es $(\overline{x}_2 \wedge \overline{x}_1 \wedge x_0)$. Es posible simular un SAT extremo con una permutación aleatoria al vuelo como la del algoritmo 7. Para el segundo, un circuito funcionará, pero la revisión de todas sus cláusulas como SAT extremo es la única manera de asegurarse de que funciona bajo la especificación de su diseño. Para el último, hice el experimento de la lotería para un problema SAT extremo con colegas y estudiantes, las personas persistentes después de semanas de preguntarme, ¿es este el número? se rinden, a pesar de que les ofrezco un millón de dólares como motivación para construir su algoritmo y derrotarme.

La consecuencia de la proposición 9 para la clase NP se demuestra en la siguiente proposición.

Proposición 10. *Para cualquier problema en NP, no existe algoritmo polinomial para resolverlo.*

Demostración. Se supone que existe un algoritmo eficiente (polinomial) para resolver un problema de NP llamado $\mathbb{E}$. Por las propiedades básicas de los problemas de la clase NP y dado que SAT es problema clásico de la clase NP, se asume que es posible construir un algoritmo eficiente para traducir entre las diferentes instancias de los problemas SAT y $\mathbb{E}$ y recíprocamente. Entonces cualquier SAT dado, puede ser resuelto en tiempo polinomial traduciéndolo y resolviéndolo en $\mathbb{E}$ con el algoritmo eficiente de $\mathbb{E}$. La complejidad de todo este procedimiento es de tiempo polinomial. Pero, esto contradice la proposición 9. Por lo tanto, no existe un algoritmo eficiente para ningún problema NP.

6. Conclusiones y trabajo futuro

El algoritmo paralelo modificado de este artículo explora el uso de ejecución en paralelo con 2^p procesadores independientes para mejorar el algoritmo paralelo sin álgebra en [Bar16b,Bar16a]. Los resultados principales son: 1) la complejidad lineal del algoritmo paralelo modificado 8 y 2) sus implicaciones para resolver los problemas NP: proposiciones 9 y 10.

Es viable en el futuro extender el algoritmo 8 para fórmulas lógicas que usen paréntesis anidados, los operadores lógicos $\Rightarrow$, $\Leftrightarrow$ y combinaciones CNF y DNF.

REFERENCIAS

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

[Bar16a] Carlos Barrón-Romero. Un algoritmo numérico para problemas de satisfacción booleana sin álgebra. In *Memoria VIII Congreso Internacional de Computación y Telecomunicaciones(COMTEL 2016), 21 al 23 de septiembre de 2016, Lima, Perú*, pages 31–38, 2016.

[Bar16b] Carlos Barrón-Romero. Decision boolean satisfiability problem without algebra. *ArXiv*, http://arxiv.org/abs/1605.07503, April, 2016.

[Bar16c] Carlos Barrón-Romero. Lower bound for the complexity of the boolean satisfiability problem. *ArXiv*, http://arxiv.org/abs/1602.06867, February, 2016.

[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.

[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.