v. 2, n. 26, 2022

Journal of
Engineering
Research

ALGORITMO NUMÉRICO EN PARALELO PARA EL PROBLEMA DE SATISFACCIÓN LÓGICA Y PROBLEMAS NP

Carlos Barrón Romero
Universidad Autónoma Metropolitana - Unidad Azcapotzalco
CDMX, México
ORCID * : 0000-0003-2435-6645

Resumen: Se presenta una nueva versión modificada de un algoritmo numérico y paralelo para resolver el problema de satisfacción lógica con cláusulas en forma conjuntiva normalizada. La forma de resolver el problema no usa algebra, ni estrategias de búsqueda computacionales como ramificación limitada, búsqueda adelante y atrás, representación por árboles, etc. El algoritmo se basa en la 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 resultantes 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 subproblemas simples de satisfacción lógica y con límite 2^n iteraciones, donde 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 relación con la clase NP es presentada al final.

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

INTRODUCCIÓN

La modelación para la resolución del problema de satisfacción lógica (SAT) se base en el concepto de reducción (ver [Bar10], capítulo 6). Este término significa 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). Este artículo es la versión revisada artículo publicado [Bar17].

La notación y convenciones para fórmulas lógicas son las usuales: 0 (falso), 1 (verdadero), los operadores lógicos son not: $\bar{x}$, and: ^ y or: ∨.De aquí en adelante, $\sum=\{0,1\}$ es el correspondiente alfabeto binario y x es el conjunto de n variables $x_{n-1},...,x_0$. Una cadena binaria w $\epsilon \sum^n$ es identificada con su valor binario $[0,2^n-1]$ y recíprocamente. Además, a una cláusula CNF, por ejemplo, $x_{n-1} \vee \cdots \bar{x}_1 \vee x_0$ le corresponde la cadena binaria 1...01, donde se toma 1 si se tiene x y se toma 0 si se tiene $\bar{x}$ en la variable i-ésima. Note que $\bar{b}= \bar{b}_{n-1} \ldots \bar{b}_1 \bar{b}_0$ es el número binario con los dígitos negados de *b*.

El problema SAT consiste en determinar cuándo una fórmula lógica φ, sin pérdida de generalidad en forma conjuntiva normalizada (CNF), pertenece o no pertenece al lenguaje SAT (L(SAT)), donde L(SAT)= {φ | φ es una fórmula lógica para la existen valores booleanos que la satisfacen} o equivalentemente, existe un testigo w $\epsilon \sum^n$ tal que φ(w) ≡ 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 exactamente n variables. Ambos problemas pueden tener muchas cláusulas repetidas, en desorden, o con sus variables en cláusulas alejadas. En este trabajo se asume que las variables están ordenadas por su subíndices: $x_{n-1},...,x_0$.

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 ≥ k. Por ejemplo, para n=7 una fórmula de (3,7)-SAT es $(x_2 \vee x_4 \vee x_6) \wedge (x_0 \vee x_1 \vee x_2) \wedge (x_1 \vee x_3 \vee x_4)$.

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 $(x_4 \vee x_5 \vee x_7) \wedge (x_2 \vee x_4) \wedge (x_0 \vee x_1 \vee x_2) \wedge (x_1 \vee x_3 \vee x_4 \vee x_5)$. Este algoritmo es una versión mejorada del algoritmo especial paralelo para resolver cualquier tipo de SAT sin usar algebra [Bar16c,Bar16a, Bar17].

Un resumen de los resultados anteriores es:

1. El espacio de soluciones (o espacio de búsqueda o espacio de las posibles asignaciones satisfactorias) es $\sum^n$.

2. Se determina con complejidad O(1) que una fórmula SSAT con n variables y m cláusulas pertenece a L(SAT) (sin necesidad de testigo). La comparación de m versus 2^n es una condición suficiente para responder si pertenece o no pertenece a L(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 $\sum$ y en $\sum^2$:

x_0
1
0

x_1	x_0
0	0
1	1
0	1
1	0

4. Si $b \in \sum^n$ corresponde a la negación de la traducción de una cadena de una cláusula de una fórmula SSAT φ entonces 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, Bar17] 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 7).

El artículo está organizado como sigue: La sección 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,Bar17]). 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 mínima 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 describe los resultados teóricos y las consecuencias

que tiene el algoritmo paralelo para SAT en conjunto con los algoritmos que traducen de la clase NP a problemas SAT. Finalmente, la última sección presenta las conclusiones y trabajo futuro.

PROPIEDADES DE SAT

Por razones de espacio y 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,Bar17].

Proposición 1. Cualesquiera cláusula CNF x sobre las variables (x_{n-1}, ...,x_0) le corresponde un número binario $b=b_{n-1}b_{n-2}...b_0$. Entonces $x(b)\equiv 1$ y $x(\bar{b}) \equiv 0$ donde b es la *traducción de la cláusula CNF x* y los valores de las variables de x se toman de las cadenas $b,\bar{b}$ de $\sum^n$. Por ejemplo, $x=(\bar{x}_2 \vee x_1 \vee \bar{x}_0)$ le corresponde y b=010 ($\bar{b}$=101), entonces $x(b) = (1 \vee 0 \vee 1) \equiv 1$ y $x(\bar{b})= (0 \vee 0 \vee 0) \equiv 0$.

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 una fórmula lógica y una variable, que no está en F.

Entonces $F = \begin{matrix}(F \vee v)\\ \wedge (F \vee \bar{v})\end{matrix}$. Por ejemplo, $\varphi_3 = \begin{matrix}(x_3 \vee \bar{x}_2 \quad \vee x_0)\\ \wedge (\quad x_2 \vee x_1 \vee \bar{x}_0)\end{matrix}$, se simplifica a $\varphi_0=x_3 \vee x_1$ y son equivalente por la proposición anterior con $\varphi_4 = \begin{matrix}(x_3 \vee \bar{x}_2 \vee x_1 \vee x_0)\\ \wedge (x_3 \vee \bar{x}_2 \vee \bar{x}_1 \vee x_0)\\ \wedge (x_3 \vee x_2 \vee x_1 \vee \bar{x}_0)\\ \wedge (\bar{x}_3 \vee x_2 \vee x_1 \vee \bar{x}_0)\end{matrix}$.

Mediante factorizaciones y simplificaciones, se destaca que φ_0, φ_3 y φ_4 tienen asignaciones satisfactorias cuando x_3=1 y x_1=1, las otras variables pueden tomar 0 o 1. Note que φ_4 tiene las asignaciones satisfactorias: {1010, 1011, 1110, 1111} y las soluciones de las tres fórmulas se corresponden. Este pequeño ejemplo se puede verificar 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.

ALGORITMOS PARA SAT

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

Algoritmo 1. Id_SSAT

Entrada: $x = \{x_k,...,x_1,x_0\}$: 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 índices interpretados como dígitos de la base numérica n con los valores {n − 1,n − 2,...,1,0}

```
Inicio
  ix = 0;
  k = |x|; // | · | cardinalidad de un conjunto.
  Si k =l 0 entonces
    salida: "ix";
    regresar;
  Fin si
  base == 0;
  itera j = 0 a k − 1
    base=base+C(n,j); // · coeficiente binomial
```

Fin itera

construir v = $\{v_k,...,v_0\}$; // conjunto de variables con los menores // índices en orden descendente de tamaño k + 1;

Mientras (índices(x) < (índices(v)) hacer.

t = v; repite

t = incrementa(t,1); // incrementa en uno los índices de t como un número // en base n

Si índices(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"; // identificación

Regresar;

Fin

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_candidato_fallido

Entrada: n : número de variables y φ fórmula del problema;

Recepción de mensajes c: número $\epsilon\sum^n$;

Hacer: recibir_mensaje(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 , doblemente // encadenado para manejarlo como una lista circular; // 1: viable, 0: no satisface φ

siguiente:=0: entero // apuntador a lista L_cand_stat;

anterior:= 2^{n-1} // apuntador a lista L_cand_stat;

Inicio

Mientras (por siempre) hacer

Mientras no vacía (L_c) hacer

c := sacar_candidato_de_algún_mansaje(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, siguiente, anterior);

n_cand := n_cand - 1;

Si (n_cand = 0) entonces

Mensaje("El algoritmo 2 confirma que φ L(SAT) después de

revisar .");

Alto total;

Fin si

Fin mientras

Si (n_cand > 0) entonces

c := siguiente; // saca de la lista circular a un candidato

// que no ha sido verificado

Si (φ(c) = 1) entonces

Mensaje("El algoritmo 2 confirma que φ ϵL(SAT), c es una asignación satisfactoria.");

Alto total;

Fin si

L_cand_sta[c] := 0;

Actualizar_lista_circular(L_cand_sta, c, siguiente, anterior);

n_cand := n_cand - 1;

Si (n_cand = 0) entonces

Mensaje("El algoritmo 2 confirma que φ $\notin$L(SAT) después de revisar $\sum^n$.");

Alto total;

Fin si

Fin mientras

Regresa
Fin

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 mínima sin detener a otros procesos porque se ejecuta en procesador propio que maneja su memoria en forma independiente y exclusiva. La pérdida 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 ^{n} aún con pérdida de mensajes o candidatos no satisfactorios repetidos. Note que al inicio todos los elementos del arreglo L_cand_stat son 1 y que los pone a 0 conforme verifica que sean o no una asignación satisfactoria, o bien cuando recibe un mensaje de un candidato no satisfactorio y al estar ligados en una lista doblemente ligada, se puede saltar a los candidatos que no han sido verificados para evaluar φ(c). O sea, la lista derece por un factor 2^{p} de la exploración y cooperación del algoritmo 7.

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 ∑ una asignación de una cláusula traducida con menos de n variables.

Algoritmo 3. Número_$\sum^{n}$

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: (kΣ: cadena de ^{n}).

Memoria: i : entero;

Inicio

itera i:= n − 1 decrementa 0 hacer

Si (i ϵ rw) entonces

kΣ [i] := valor de rv de la variable i;

de otra forma

kΣ [i] := selección aleatoria en Σ;

Fin si

Fin itera

Regresa kΣ;

Fin

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;

Inicio

Si (Id SSAT(rw) ϵ/ SSAT) entonces

Construir objeto SSAT(Id_SSAT(rw));

Fin si

Con SSAT(Id_SSAT(rw)) hacer

k := cláusula a binario (rw);

k_aux := k;

Si (tamaño(k) < φ.n) entonces

k_aux := Número$_{\sum}$n(set of variables(rw),k); // Algoritmo 3

Fin si

Si (φ(k_aux) == 1) entonces

Mensaje(“El algoritmo 4 confirma que φ ϵL(SAT), k_aux es una

asignación satisfactoria.”);

Alto total;

de otra forma

Actualizar_candidato_fallido(k_aux); // Algoritmo 2

Fin si

Si (tamaño(k) = φ.n) entonces
Actualizar_lista_circular(S, k, ct); // Actualiza la lista circular y // el contador ct
Si (ct = 2^n) entonces
Mensaje("El algoritmo 4 confirma que φ ∉L(SAT), SSAT(Id SSAT(rw)) es un tablero bloqueado.");
Alto total;
Fin si
// k de tamaño n, no satisface φ.
// Ver proposición 1.
Actualizar_candidato_fallido(k); // Algoritmo 2
Fin si
Actualizar_lista_circular(S, k, ct); // Actualiza la lista circular y el contador ct
Si (ct = 2^n) entonces
Mensaje("El algoritmo 4 confirma que φ L(SAT). SSAT(Id SSAT(rw)) es un tablero bloqueado.");
Alto total;
Fin si
Fin con
Regresa
Fin

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 ¿φ ϵL(SAT)?

Entrada: n : número de variables y φ fórmula del problema.

Salida: Mensaje y si hay solución el testigo x, tal que φ(x) ≡ 1.

Memoria:
r: conjunto de variables de X;
SSAT:=nulo: Lista de objetos SSAT.
Inicio
Mientras no(no sea la cláusula final de φ);
r := φ.leer_cláusula;
Actualizar_SSAT(SSAT,r). // Algoritmo 4
Fin mientras;
Con lista SSAT hacer
// ×θ es producto cruz y junta natural
Calcular Θ = ×θ ∀ SSAT(Id_SSAT(r));
si Θ = ø entonces
Mensaje("El algoritmo 5 confirma que φ∉(SAT).
Los SSAT(Id_SSAT(r)) son incompatibles.");
Alto total;
de otra forma
Mensaje("El algoritmo 5 confirma que φ ϵL(SAT). s es una asignación satisfactoria, s ϵΘ. Los SSAT(Id SSAT(r)) son compatibles");
Alto total;
Fin con
Fin si
Fin

Algoritmo 6. Probar_$\varphi(\cdot)$

// Evalúa un candidato y actualiza candidatos fallidos.
Entrada: c: cadena de $\sum^n$.
Salida: ninguna.
Inicio
Si (φ(c)1) entonces
Mensaje("Los algoritmos 6 y 7 confirman que φ ϵL(SAT), c es una
asignación satisfactoria.");
Alto total;
De otra forma
Actualizar_candidato_fallido(c); // Algoritmo 2
Fin si
Regresa;
Fin

El siguiente algoritmo realiza una búsqueda aleatoria cuyos candidatos son seleccionados arbitrariamente de $\sum^n$ (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 Σ^{n-1} y 1 Σ^{n-1}) o en (b Σ^{n-p} y $\bar{b}$ Σ^{n-p}) para resolver ¿$\varphi \in L(SAT)$?, donde b es una cadena de Σ^p

Entrada: n *número de variables*, φ fórmula del problema,

p *número de procesadores* dobles (2^p),

b cadena de Σ^p

Salida: Mensaje y cuando determina una solución: alto total y el testigo s donde $s \in \Sigma^n$, tal que $\varphi(s) \equiv 1$.

Memoria: T[0:2^{n-p}-1] = Permutación de cadenas de Σ^{n-p}: entero;

Mi =$2^n - 1$: entero; rdm, a: entero;

Inicio

Itera i:=0 a $2^{n-p} - 1$

Ejecución en paralelo

// Concatenación de b y T[i] para generar una cadena de Σ^n;

// Concatenación de $\bar{b}$ y T[i] para generar una cadena de Σ^n;

Probar_φ(bT[i]) // Algoritmo 6;

Probar_φ($\bar{b}$ T[i]) // Algoritmo 6;

// Nota: si la cadena es un testigo satisfactorio, el algoritmo

// 6 detiene todo

Fin ejecución en paralelo

Fin itera

Si (p=1) entonces // caso con solo un procesador doble

Mensaje("El algoritmo 7 confirma que $\varphi \notin L(SAT)$ después probar todas las cadenas de Σ^n");

Alto total;

de otra forma // caso con más un procesador doble

Mensaje("El algoritmo 7 confirma que no hay solución en el bloque de cadenas b Σ^{n-p} $\cup \bar{b}\Sigma^{n-p}$");

Fin si

Fin

En el diseño de la rutina anterior se obtuvo una disminución de las iteraciones a la mitad, al usar al menos un procesador doble (p=1, 2^1) ya que en este caso verifica de dos en dos los 2^n candidatos al paralizar la evaluación de Probar_φ(·), O sea realiza dos verificaciones a la vez.

Se elige un número 2^p de procesadores por la facilidad de dividir el espacio Σ^n. Para p>1, (2^p), o sea procesadores dobles el algoritmo 7 puede explorar particionadamente el espacio de búsqueda Σ^n en 2^{p-1} secciones, o sea cada procesador doble verifica bΣ^{n-p} y $\bar{b}\Sigma^{n-p}$ independientemente para b=0,2^p-1 como cadenas de Σ^p. O sea, con p procesadores dobles el límite superior de iteraciones pasa de 2^n a $\frac{2^n}{2^p}$ 2^{n-p}. El número de posibles candidatos a explorar no se altera, son los 2^n de $\Sigma^n = \cup_{b=0}^{2^{p-1}}$ ($b\Sigma^{n-p} \cup \bar{b}\Sigma^{n-p}$).

La disminución en tiempo es bajo la suposición de que las 2^p evaluaciones de φ(·) son simultaneas en procesadores independientes. Para 2^1 procesadores se tiene $\frac{2^n}{2^1} = 2^{n-1}$. Por ejemplo, con 4 ($2^2$) procesadores independientes, las iteraciones son $\frac{2^n}{2^2}$=2^{n-2} y los 4 candidatos simultáneos son 00x, 01x, 10x y 11x donde $x \in \Sigma^{n-2}$, que se invocan con algoritmo 7(n, φ, 2, 00); algoritmo 7(n, φ, 2, 01).

En general con 2^p procesadores independientes, el límite superior de iteraciones es 2^{n-p}, sin embargo, esto no es necesariamente una mejora indisputable del tiempo, 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 $n >> 0$ un gran número.

Note que el algoritmo 7, realiza sus pruebas aleatoriamente, ya que el arreglo T contiene una permutación aleatoria de cadenas de Σ^{n-p}, que es muy fácil de construir en tiempo lineal, **O**(n-p).

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, φ fórmula del problema, p procesadores dobles (2^p)

Salida: Mensaje que confirma si φ ϵL(SAT) o no.

Inicio
Ejecución en paralelo
algoritmo 2(n, φ);
algoritmo 5(n, φ);
Para b=0 a 2^{p-1} // como cadenas de $\sum^p$
    algoritmo 7(n, φ, p, b);
Fin Para
Fin ejecución en paralelo
Fin

ANÁLISIS DE LA COMPLEJIDAD Y DE LA COMPUTABILIDAD DEL ALGORITMO PARALELO 8

Suponiendo que se tienen máquinas 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 ×θ.

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 ×θ es computable y sus iteraciones están limitadas por m (número de cláusulas de φ) más las iteraciones del operador ×θ.

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 ×θ solo se detiene cuando al evaluar φ(·) 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 $\bar{k}$), 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 ×θ para las n variables de φ. El operador ×θ 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(·).S, se tiene que SSAT(Id_SSAT(r)) ×θ SSAT(Id_SSAT(r')) =

1. Si $r \cap r' = \emptyset$ entonces SSAT(Id_SSAT(r)).S × SSAT(Id_SSAT(r')).S.

2. Si $r \cap r' \neq \phi$ y hay cadenas 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 $r \cap r' \neq \phi$ y no hay valores comunes

entre SSAT(Id_SSAT(r)).S y SSAT(Id_SSAT(r')).S para las variables en r ∩ r' entonces ϕ.

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

El algoritmo 5 se comporta como un compilador de una pasada con terminación abrupta y sólo requiere inspeccionar las cláusulas del φ. Ejemplos de los casos del operador ×θ se tienen en [Bar17].

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 máquinas 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 $\varphi(c) \equiv 1$ 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 $\sum^n$. 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 bits que completan los 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 máquinas 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, 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 $\sum^n$ lo que le toma 2^n iteraciones.

Proposición 6. El algoritmo paralelo 8 es computable y su complejidad está 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) $\varphi \in L(SAT)$, porqué existe $x \in \sum^n$ tal que $\varphi(x) \equiv 1$, o 2) $\varphi \notin L(SAT)$, ya que no existe $x \in \sum^n$ que satisface φ(·).

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 realizó una revisión de todas sus cláusulas y se requiere ejecutar la operación ×θ para conciliar las soluciones de los subproblemas SSAT. La operación ×θ no necesita construir todas las asignaciones satisfactorias, solo se requiere una.

El algoritmo 7, cuando se paraleliza con 2^p procesadores explora $\sum^n$ en 2^{n-p} iteraciones, ya que $2^n-k2^p=0$ significa que se ha explorado completamente $\sum^n$ con p procesadores. Sin embargo, con p << 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 ×θ, pero en el peor de los casos, es decir, cuando no se produce una terminación abrupta, los algoritmos 2 y 7 exploran $\sum^n$ en a lo más 2^n iteraciones.

La probabilidad de encontrar una asignación satisfactoria después de k candidatos fallidos es $p(s,k,2^n)=\frac{s}{2^{n-k}}$.

El divisor exponencial 2^n (que corresponde al número de combinaciones de $\sum^n$) 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 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 (o sea no exponencial) de pruebas k << 2^n de candidatos diferentes, la probabilidad permanece insignificante, i.e., $p(1,k,2^n)=\frac{1}{2^n} \approx 0$.

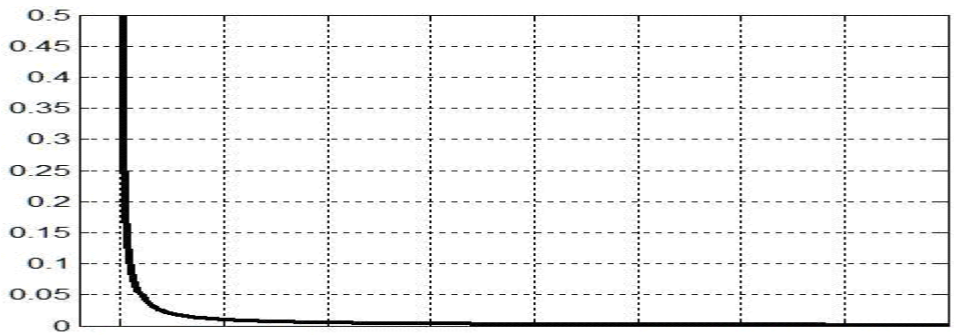

Figura 1. Gráfica de la función de probabilidad p(s, k 2^n) donde es el número de asignaciones satisfactorias, k es el número de candidatos fallidos y 2^n corresponde al número de combinaciones de $\sum^n$.

Proposición 7. Para cualquier problema SAT se cumple $p_2(s, k, 2^n) \leq p_7(s, k, 2^n) \leq p_5(s, k, 2^n)$ donde $p_i(s, k, 2^n)$ 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:

$$p_5(s, k, 2^n) = \frac{s}{2^{n-k}}.$$

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

$$p_7(s, k, 2^n) = \frac{s}{2^{n-k} - k2^p}.$$

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):

$$p_2(s, k, 2^n) = \frac{s}{2^{n-k} - k(2^p + 2)}.$$

Se denomina problema SAT extremo a un problema SAT con n >> 0 variables, donde sus cláusulas usan como máximo n variables, las cláusulas podrían repetirse y estar 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 insignificante $(\frac{1}{2^n})$ y para ninguna solución es 0. Por lo tanto, para una serie de M > 0 problemas SAT extremos, el valor esperado para solucionarlo es 0.

Proposición 8. Para un problema SAT extremo con n >> 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 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 $\sum^n$ en 2^{n-p} iteraciones, pero como p << n, las iteraciones que realiza son aproximadamente 2^n.

Lo cual significa que la probabilidad de resolverlo $p(s, k) = \frac{s}{2^{n-k} - k(2^p+2)}$ se mantiene inalterada y casi nula para k y 2^p pequeños comparados con 2^n.

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

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. Se plantean las posiciones:

1. Es una hipótesis, de un experimento mental, o sea una suposición teórica válida.

2. Los avances en la investigación de nanotecnología, clústeres de moléculas y estructuras cristalinas pronto proporcionarán circuitos electrónicos capaces y complejos como SAT extremos.

3. *Se* puede realizar un experimento tipo caja negra utilizando la equivalencia lógica entre CNF y DNF (Forma normal disyuntiva), que llamaremos problemas entrelazados y usarlos como cajas negras sin acceso a las cláusulas. Un problema SAT extremo en DNF es el conjunto vacío de cláusulas o un conjunto de tamaño m de los números binarios que son las únicas soluciones, como las cláusulas del problema DNF entrelazado. Por ejemplo, si la solución única de un problema SAT extremo es 001, entonces la cláusula DNF es $(\bar{x}_2 \wedge \bar{x}_1 \wedge x_0)$. O sea, es viable simular un SAT extremo para corroborar el resultado de la proposición 8 y el algoritmo numérico paralelo, las cláusulas de la lectura de la fórmula CNF extrema se pueden generar aleatoria y fácilmente a partir del DNF entrelazado, ya que son todas las cláusulas que sean falsas respecto del conjunto de números binarios de solución.

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].

La relación entre los problemas SAT, los de la clase NP bajo la aplicación del algoritmo numérico paralelo para la clase NP se demuestra en la siguiente proposición.

Proposición 9. Un *problema en* NP es tratable cuando su equivalente SAT cumple que el número esperado de soluciones es razonablemente grande respecto del número de cláusulas del problema SAT asociado.

Demostración. La relación entre las formas formales CNF y DNF entrelazados, significa ambos tienen las mismas soluciones, por ejemplo, sea el CNF: $(\bar{x}_1 \vee x_0) \wedge (x_1 \vee \bar{x}_0) \wedge (x_1 \wedge x_0)$, con solución 11 y su entrelazado es el DNF: $(x_1 \wedge x_0)$ con solución 11. Establece la siguiente relación entre la versión SAT extrema CNF y su entrelazada DNF respecto del número de cláusulas $c>2^n$ (de la CNF) y el número de soluciones m (cláusulas de la DNF). Cuando c es grande y m es pequeño se trata de un problema CNF extremo y un DNF sencillo, más aún el problema CNF es extremo, intratable e intraducible en tiempo eficiente (por el gran número de sus cláusulas, ver proposición 8), mientras que el equivalente DNF es tratable, traducible porque m el número de sus cláusulas es pequeño. Nota: La forma DNF tiene complejidad **O**(1) para ser resuelta, o sea no puede ser extrema, ya que independientemente del número de sus cláusulas, cualesquiera de ellas al traducirla a número binario es una solución y su versión entrelazada CNF solo es extrema cuando el número de cláusulas de la versión DNF es pequeña. Por tanto, para la clase NP, se tiene:

- Los problemas SAT con m grande (m es el número esperado de soluciones y "grande" respecto a 2^n) son tratables y el algoritmo numérico paralelo es capaz de resolverlos en tiempo eficiente. Si la traducción fuera directamente a un problema DNF, es trivial que sea satisfacible.
- La traducción es eficiente cuando el número cláusulas corresponde a un de numero de soluciones razonable. Ya que en este caso el algoritmo numérico paralelo proporciona un testigo en tiempo razonable, que es una solución del DNF entrelazado, que a su vez se puede traducir en una solución del problema NP relacionado.

Bajo las condiciones anteriores, sus equivalentes de la clase NP son tratables bajo algoritmo numérico paralelo de este artículo, o dicho de otro modo, al traducir un problema NP a uno SAT (de los que existen muchos algoritmos de traducción de tiempo eficiente), el algoritmo numérico paralelo se convierte en el juez de la tratabilidad, ya que si resuelve el SAT equivalente en un tiempo razonable, significa que el número de soluciones es "grande" respecto a (donde es el número de variables del SAT equivalente) y la traducción del testigo del SAT es la solución del problema NP dado, y todo el proceso de resolución es hecho en un tiempo razonable. El SAT equivalente es tratable.

Por otro lado, suspender el algoritmo, no garantiza que haya solución, pero los mensajes del análisis de bloques del algoritmo 7 y el valor de la variable n_cand del algoritmo 2, comparado con 2^n da una medida de la exploración realizada y es un indicador de que probablemente el problema SAT que se está analizando es extremo, o sea, tiene un valor m de soluciones esperadas muy muy pequeño respecto 2^n. Nótese que el algoritmo es computable y tiene también una dependencia positiva, en el sentido de disminuir el tiempo de ejecución, con respecto del número de procesadores p en paralelo dedicados a resolver un problema SAT dado, con un tiempo acotado exponencialmente de valor máximo 2^{n-p} y siempre termina con la

Journal of Engineering Research ISSN 2764-1317 DOI 10.22533/at.ed.3172262222117

respuesta determinante de si hay solución o no.

CONCLUSIONES Y TRABAJO FUTURO

El algoritmo numérico paralelo de este artículo explora el uso de ejecución en paralelo con intercambio masivo de mensajes y con una generación proactiva de candidatos entre 2^p procesadores independientes para reducir al máximo posible el tiempo en determinar un testigo satisfactorio por medio del algoritmo paralelo sin álgebra de [Bar16b,Bar16a,Bar17]. Los resultados principales son: 1) la complejidad del algoritmo numérico paralelo es razonable cuando el número de soluciones m es "grande" respecto a 2^n (donde n es el número de variables de un problema SAT), además con la exploración en paralelo y cooperación de los algoritmos, los candidatos fallidos reducen el tiempo lo más posible para terminar y responder ¿φ ϵL(SAT)? 2) sus implicaciones para resolver los problemas NP: proposición 9, donde el algoritmo es el juez de la tratabilidad, o sea de la determinación de una solución en un tiempo razonable para un problema SAT, que proviene de una traducción en tiempo eficiente de un problema NP. Además, es viable en el futuro extender el algoritmo 8 para fórmulas lógicas que usen paréntesis anidados, los operadores lógicos $\Rightarrow$, $\Leftrightarrow$ y combinaciones más complejas de fórmulas lógicas.

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, páginas 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.

[Bar17] Carlos Barrón-Romero**. Algoritmo numérico en paralelo para el problema de satisfacción lógica y su impacto sobre la clase NP**. In COMTEL 2017, 11 al 13 de octubre de 2017, Lima, Perú, páginas 18–25, 2017.

[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.