Propuestas algorítmicas para la resolución de los problemas de satisfactibilidad

Tesista: Guillermo De Ita Luna
Fac. Cs. de la Computación, B.U.A.P
gdeita@fcfm.buap.mx

Asesor: Guillermo Morales Luna
Centro de Inv. y Estudios Avanzados-IPN
gmorales@cs.cinvestav.mx

Resumen doctoral

1 Introducción

En esta investigación doctoral se analizaron los problemas de satisfactibilidad en el cálculo proposicional. Como resultado del análisis se diseñaron y construyeron diversas propuestas algorítmicas para la resolución de tales problemas. Enfasis especial se puso en la la determinación de la complejidad en tiempo de los peores casos que aparecen al trabajar con las propuestas algorítmicas.

El planteamiento de los problemas de satisfactibilidad puede resumirse en la forma siguiente: Dada una fórmula booleana F en *Forma Conjuntiva* (FC), el problema SAT consiste en decidir si F es satisfactible. El problema de optimización MaxSAT consiste en determinar el número máximo de cláusulas que se pueden satisfacer simultáneamente. Y el problema de conteo #SAT consiste en calcular el número de asignaciones que satisfacen a F, a éste último número lo denotaremos por $\#\mathrm{SAT}(F)$.

Si las cláusulas de la fórmula tienen exactamente k literales se dice que ésta es de la forma k-FC y, si cada variable aparece a lo más r veces se dice que la fórmula es del tipo r, k-FC. Los problemas de satisfactibilidad heredan la notación de sus instancias de entrada, por ejemplo, el problema #2,2-SAT denota el problema #SAT restringido a considerar sólo fórmulas del tipo 2,2-FC.

El problema SAT es un problema central en la teoría de la computación. En la práctica, SAT es fundamental en la resolución de problemas de razonamiento automático, en el diseño y fabricación asistido por computadora, en el desarrollo de base de datos, en el diseño de circuitos integrados, en la integración de módulos para lenguajes de alto nivel, etc. Por lo que el diseño y análisis de algoritmos que resuelven el problema SAT juega un rol importante para el avance de la tecnología computacional.

Aunque el planteamiento original del problema MaxSAT pareció un tanto artificial, actualmente, MaxSAT es crucial en el desarrollo de la teoría de la complejidad para problemas de optimización, amén de su importancia práctica en el desarrollo de bases de datos deductivas, redes de comunicación, robótica y, en general, en el área de optimización combinatoria.

Actualmente, se ha incrementado el número de investigaciones sobre el diseño y análisis de algoritmos para la resolución de MaxSAT, debido posiblemente, a los resultados recientemente obtenidos sobre algoritmos de aproximación y sobre la determinación de factores de garantía para aproximar problemas de optimización difíciles en la clase NP.

El problema #SAT ha sido un elemento clave en el desarrollo de la teoría de la confiabilidad, particularmente en sus aplicaciones al análisis de la seguridad, confiabilidad y conectividad en redes de comunicación.

#SAT tomó un rol importante a partir del trabajo clásico de Valiant [10] en donde se presentó una transformación de tiempo polinomial del problema #2-SAT al problema de contar el número de subredes que contienen un camino entre cualesquiera dos nodos de una red dada (un caso del análisis de la confiabilidad de la red).

Los problemas de satisfactibilidad SAT, MaxSAT y #SAT son problemas completos en las clases de complejidad NP, MAX NP y #P respectivamente [4]. Para cada uno de estos problemas hay instancias restringidas de fórmulas booleanas que pueden resolverse eficientemente (es decir, se resuelven utilizando algoritmos de complejidad polinomial en tiempo). Aunque las instancias resueltas eficientemente para algunos de estos problemas pueden ser instancias intratables para los otros problemas. Por ejemplo, para fórmulas del tipo 2-FC, el problema 2-SAT es decidible en tiempo polinomial, pero tanto Max2SAT como #2-SAT son problemas intratables [4,10].

Un punto importante al trabajar con SAT, MaxSAT y #SAT es que, como representantes de sus respectivas clases de complejidad, hallar mejoras a los algoritmos que los resuelven, implica que tales mejoras se pueden extender a los demás problemas de la clase correspondiente.

Por ejemplo, el obtener algoritmos que mejoren el factor de aproximación para resolver MaxSAT, implica mejorar los factores de aproximación con los que se resuelven los demás problemas de la clase MAX NP.

2 Panorama algorítmico de los problemas de satisfactibilidad

Se dice que un algoritmo para SAT es *completo* si para cualquier fórmula de entrada éste siempre puede verificar si existe o no una solución. Por ejemplo, un algoritmo simple pero completo consiste en enumerar todas las posibles asignaciones de la fórmula de entrada y verificar si alguna de éstas satisface a tal fórmula.

Un algoritmo *incompleto* para SAT puede encontrar una respuesta correcta para cierta clase de fórmulas, pero para otras clases puede no hallar la solución exacta.

Todos los algoritmos completos disenãdos hasta ahora para resolver a SAT son de complejidad exponencial en tiempo, mientras que los algoritmos incompletos generalmente se diseñan para trabajar bajo un tiempo acotado polinomialmente.

Los métodos utilizados en el diseño de algoritmos completos para SAT pueden adaptarse para proporcionar soluciones exactas de MaxSAT. Mientras que las técnicas utilizadas en los algoritmos incompletos para SAT se adaptan para proporcionar soluciones aproximadas de MaxSAT, puesto que, un algoritmo incompleto puede diseñarse de forma tal que si no es posible determinar la satisfactibilidad de la fórmula de entrada, entonces se indique el número máximo de cláusulas que se satisfacen.

Una clasificación de las diferentes propuestas algorítmicas para resolver SAT, y que por tanto, pueden adaptarse para resolver MaxSAT, es la siguiente:

1. Algoritmos completos que trabajan en un espacio de búsqueda discreto.

Estos van construyendo una asignación solución, considerando en cada iteración a una cláusula o a una variable de la fórmula de entrada. Aplican métodos de búsqueda discreta y trabajan sobre el espacio de posibles asignaciones de la fórmula, además de utilizar procedimientos de inferencia basados en lógica proposicional.

Ejemplos de esta clase de algoritmos son: algoritmos que aplican retrocesos, los que utilizan reescritura de reglas, los basados en sistemas de producción, los que aplican resolución y resolución regular y los que utilizan sistemas de desigualdad matricial.

Muchos de estos algoritmos eliminan una variable en cada iteración, tal y como se presentó en la versión original del procedimiento de Davis y Putnam, o bien, trabajan seleccionando una variable y asignándole cada uno de los dos valores posibles, generando una subfórmula en cada caso, tal y como Loveland lo hace en sus propuestas de modificación al procedimiento de Davis y Putnam.

Ambas técnicas pueden combinarse para construir nuevos algoritmos, como es el caso de los algoritmos de Davis y Putnam simplificados y la aplicación de Davis y Putnam con ordenamiento estricto de variables.

Una clase especial de algoritmos completos se conforma por aquellos algoritmos que transforman instancias del problema SAT a instancias de problemas de programación entera.

2. Algoritmos incompletos que trabajan en un espacio de búsqueda discreto.

Es de mencionar que éstos trabajan partiendo de una asignación completa de la fórmula (o tal vez varias asignaciones) y, en cada iteración, intentan mejorar la asignación de tal forma que se incremente el número de cláusulas satisfechas por la misma.

Algoritmos en esta clase, son: Métodos de búsqeda local, algoritmos que utilizan caminos aleatorios y algoritmos basados en estrategias ávidas, ver por ejemplo [5,6].

Actualmente, una línea activa de investigación es la extensión de heurísticas a los procesos de búsqueda local con el fin de acelerar su convergencia y/o mejorar su factor de aproximación, entre este tipo de propuestas se encuentran las búsquedas locales no obvias, la búsqueda tabú, el recocido simulado y los algoritmos genéticos [1].

Las investigaciones en esta clase de heurísticas han mostrado que pueden ser más rápidas que los métodos discretos, aunque también presentan la particularidad de no ser alternativas robustas, en el sentido de que con ligeros cambios en sus parámetros, tales heurísticas encuentran soluciones muy diferentes.

3. Algoritmos incompletos que trabajan en un espacio de búsqueda continuo.

En esta clase se encuentran los métodos basados en relajaciones de programación entera, se incluyen también las técnicas de ramificación y corte, así como los algoritmos que aplican planos de corte y los métodos de punto interno. Otros métodos que suelen relajar el espacio de búsqueda para considerar espacios continuos son los presentados en [5] y las técnicas derivadas de la aplicación de programación lineal y programación semidefinida.

Entre las diferentes técnicas desarrolladas para resolver #SAT, se pueden mencionar los trabajos de Dubois y Lozinskii [3,9] en donde se aplica la fórmula de inclusión -exclusión para calcular #SAT(F), dada como entrada la fórmula booleana F.

Otros trabajos que han tenido repercusión en el desarrollo de algoritmos para #SAT son los trabajos de Heidtmann, Bertschy y Karp, aunque estos autores consideran el problema esencialmente equivalente de calcular #SAT(F) cuando F está en *Forma Disyuntiva* (FD).

Enfasis especial tienen los trabajos donde se aplican procesos aleatorios para resolver problemas en #P, como son los trabajos sobre el conteo de apareamientos perfectos en gráficas bipartitas, la técnica de aproximar utilizando ejemplos y, particularmente, la aplicación de métodos basados en cadenas de Markov, como son los trabajos clásicos de Jerrum y Sinclair en el tratamiento de aproximar el permanente de una matríz. Todas estas técnicas estocásticas han resultado ser de alcance general y por tal, pueden adaptarse en el tratamiento de otros problemas en #P, como es el caso del problema #SAT.

3 Propuestas implementadas para SAT y MaxSAT

Se implementaron diversas heurísticas para resolver eficientemente, aunque de manera aproximada, a SAT y MaxSAT. Las heurísticas se basan en procedimientos ya clásicos, a saber, en la búsqueda local, el recocido simulado, la búsqueda tabú y los algoritmos genéticos, y, como ellos, son de alcance general y por tanto, pueden adaptarse a los procesos de búsqueda local al tratar otros problemas de optimización combinatoria.

La búsqueda local (BL) es una de las formas primitivas de aplicar optimización continua en un espacio de búsqueda discreto. Una forma para analizar la BL es a través de la estructura: $< S_0, f, H >$, donde:
S_0 - es el procedimiento utilizado para generar el punto inicial de búsqueda,
f - es la función objetivo y
H - es la métrica definida en el dominio de búsqueda.

Para el caso particular de SAT y MaxSAT, el procedimiento de búsqueda opera sobre el espacio discreto de posibles asignaciones de la fórmula de entrada, se utiliza como punto inicial de la búsqueda a una asignación generada aleatoriamente y, como métrica entre puntos del dominio, se utiliza la distancia de 'Hamming'.

Como función objetivo hemos utilizado el polinomio que se obtiene de *aritmetizar* la fórmula booleana [1]. Si

$$F = \bigwedge_{i \leq m} \bigvee_{j \leq k_i} x_{\nu_j}^{\epsilon_{\nu_j}}$$

con $\epsilon_{\nu_j} \in \{0, 1\}$ entonces el polinomio que codifica a F y función objetivo en la búsqueda local, es

$$p_F = \sum_{i \leq m} \prod_{j \leq k_i} m_{\nu_j}(x_{\nu_j})$$

donde $m_{\nu_j}(x_{\nu_j}) = (1 - x_{\nu_j})$ si $\epsilon_{\nu_j} = 1$, o bien
$m_{\nu_j}(x_{\nu_j}) = x_{\nu_j}$ si $\epsilon_{\nu_j} = 0$. Así por ejemplo, si $F(x_1, x_2, x_3) = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3)$, el polinomio que codifica a esta fórmula, será:
$p_F(x_1, x_2, x_3) = (1 - x_1)\, x_2 + x_1\, (1 - x_2)\, (1 - x_3)$.

Para una asignación dada, p_F cuenta el número de cláusulas insatisfechas por tal asignación. Decidir la satisfactibilidad será equivalente a minimizar p_F:
F es satisfactible si y sólo si el mínimo de p_F es cero

Diferentes tipos de búsqueda local pueden obtenerse al utilizar diferentes funciones objetivos, a éstos nuevos tipos de búsqueda se les llama comúnmente *búsqueda local no obvia* (BL-N).

Un caso de búsqueda local no obvia para Max-kSAT es la propuesta por Khanna [9], donde la función objetivo no obvia es de la forma:

$$f_{NOB} = \sum_{i=0}^{k} C_i W(S_i)$$

Dada una asignación x, para $i \in \{0, ..., k\}$, S_i denota al conjunto de cláusulas donde exactamente i literales son verdaderas bajo la asignación, $W(S_i)$ es un peso que se asocia a S_i, por ejemplo, $W(S_i) = |S_i|$, y C_i es una constante real seleccionada adecuadamente para cumplir ciertas condiciones del proceso de búsqueda.

Por ejemplo, la función objetivo propuesta por Khanna para fórmulas del tipo 2-FC, es:

$$f_{NOB} = \frac{3}{2} W(S_1) + 2W(S_2)$$

Se sabe que para Max-kSAT, BL tiene un factor de garantía de $\frac{k}{k+1}$ [6], mientras que BL-N tiene un factor de garantía de $1 - \frac{1}{2^k}$ si las cantidades $\Delta_i = C_i - C_{i-1}$, $i \in \{1, ..., k\}$ satisfacen:

$$\Delta_i = \frac{1}{(k - i + 1)\begin{pmatrix} k \\ i-1 \end{pmatrix}} \left[\sum_{j=0}^{k-i} \begin{pmatrix} k \\ j \end{pmatrix} \right] \tag{1}$$

Por ejemplo, BL tiene un factor de garantía de 2/3 para Max2SAT, mientras que BL-N lo tiene de 3/4.

Analizando las soluciones obtenidas con programas que implementan tanto a BL como a BL-N, se observó que existen instancias de Max-kSAT donde BL obtiene mejores soluciones que BL-N. Un caso simple de esta situación se conforma por la familia de fórmulas del tipo: $F = \{\{x_i, x_j\} | 1 \leq i < j \leq n, n \geq 5\} \cup \{\{\overline{x}_1, \overline{x}_2\}\}$. Por ejemplo, con $n = 5$ y tomando como punto actual en una 1-vecindad la asignación $x = (1, 1, 1, 1, 1)$ se tiene que $f_{NOB}(x) = (3/2)(0) + 2\binom{5}{2} = 20$.

De hecho, x es un óptimo local para f_{NOB} ya que para todo $x' \in N(x), f_{NOB}(x) \geq f_{NOB}(x')$. Para $x_1 = (0, 1, 1, 1, 1) \in N(x)$, se tiene que:
$f_{NOB}(x_1) = (3/2)(5) + 2(6) = 19.5$.

Nótese que aunque x es un óptimo local, esta asignación no satisface a F, en tanto que la asignación x_1 sí satisface a F. En [2] presentamos una nueva función objetivo no obvia que corrige tal anomalía, la función objetivo que se propone es:

$$f_{NOB} = \sum_{i=1}^{k} C_i W(S_i) - W(S_0) \qquad (2)$$

Por ejemplo, para Max2SAT la función objetivo será:

$$f_{NOB} = 2 \cdot W(S_2) + \frac{3}{2} \cdot W(S_1) - W(S_0) \qquad (3)$$

Obsérvese que los valores de los coeficientes C_i, con $i = 1, ..., k$, son importantes para diferenciar entre asignaciones que tienen el mismo número de cláusulas satisfechas, sin embargo, el coeficiente C_0 es crítico para poder considerar el número de cláusulas no satisfechas entre diferentes asignaciones. BL-N basada en esta nueva función objetivo puede inclusive alcanzar mejores cocientes de aproximación que con la función de Khanna.

Puesto que un óptimo local bajo esta nueva función objetivo, cumple la relación:

$$2\Delta_2 W(S_2) + (\Delta_1 - \Delta_2)W(S_1) \geq 2\Delta_1 W(S_0). \qquad (4)$$

Entonces, si al analizar los cocientes: $\frac{W(S_1)}{W(S_0)}$ y $\frac{W(S_2)}{W(S_0)}$, es posible determinar para una instancia dada que se cumple la relación: $W(S_1) \leq p \cdot W(S_0)$ o $W(S_2) \leq p \cdot W(S_0)$, se puede entonces encontrar valores apropiados para los coeficientes C_0, C_1 y C_2 en tal forma que se obtenga bajo la ecuación 4 un cociente de aproximación maximal.

Se construyeron programas que implementan a BL, y a las dos búsquedas locales no obvias; la basada en la función de Khanna y la basada en la función objetivo que proponemos, a ésta última BL-N se le adicionaron diversas estrategias para acelerar la convergencia del proceso de búsqueda. Entre las estrategias adicionadas, se utilizó un proceso tabú y, posteriormente, un proceso de reinicio de búsqueda después de arribar a un óptimo. Como nuevo punto para reiniciar la búsqueda se eligió el punto complementario del último óptimo local encontrado.

Se analizó también el comportamiento de algoritmos basados en el recocido simulado y en el paradigma genético cuando éstos se aplican a la resolución de MaxSAT.

El análisis empírico realizado a las heurísticas programadas, ejecutando éstas sobre instancias de fórmulas booleanas construidas aleatoriamente, mostró que, efectivamente aceleran la convergencia de la búsqueda local y obtienen en la práctica mejores aproximaciones que al utilizar la búsqueda local por sí sola.

Los mejores resultados (en el sentido de estar más cercanos al óptimo), fueron obtenidos por el recocido simulado y con la heurística de búsqueda tabú que utiliza la función objetivo que propusimos, además de que esta última heurística converge más rapidamente que el recocido simulado.

Del análisis empírico se infiere que los cocientes de aproximación obtenidos en la práctica, cuando las heurísticas procesan fórmulas aleatorias del tipo 2-FC están por arriba de 0.9 (salvo el caso del algoritmo genético, el cual tuvo cocientes de aproximación entre 0.8 y 0.9). Del análisis realizado concluimos que la aplicación 'pura' del paradigma genético resulta ser una propuesta muy general para MaxSAT, por lo que es indispensable combinar los procesos evolutivos con procesos ya clásicos basados en propiedades lógicas y/o combinatorias de las fórmulas booleanas, y así obtener propuestas competitivas para la resolución de los problemas de satisfactibilidad en general.

4 Propuestas para #SAT

El trabajo de tesis versa fundamentalmente sobre las propuestas algorítmicas diseñadas para resolver el problema #SAT. Los resultados obtenidos son presentados desde la perspectiva de la lógica matemática y de la teoría de la complejidad. En todo el documento se dió especial énfasis en la determinación de la complejidad en tiempo de las propuestas presentadas.

4.1 Aplicando la fórmula de inclusión-exclusión

Se desarrolló un algoritmo para calcular #SAT(F) en donde se aplica una transformación sobre la fórmula F de entrada, construyéndose una nueva fórmula G en donde se mantiene invariante el valor #SAT(F), es decir, #SAT(F) = #SAT(G). La transformación se basa en la propiedad de cláusula subsumida y en la de cláusulas independientes.

Considerando a una cláusula como el conjunto de literales que la componen, tenemos que: Una cláusula C_i *subsume* a C_j si $C_j \subseteq C_i$, y C_i es *independiente* con C_j si $C_i \cup C_j$ es una tautología, es decir, si hay una variable x tal que $x \in C_i \cup C_j$ y $\overline{x} \in C_i \cup C_j$.

Denotemos con $F \in \mathcal{F}(m, n)$ a una fórmula booleana con m cláusulas y en donde aparecen n variables, digamos $F = \{C_1, ..., C_m\}$. Denotemos con $Var(F)$ al conjunto de variables que aparecen en F y con $Lit(F)$ al conjunto de literales, también en F. Una forma de calcular #SAT(F) es:

$$\#\mathrm{SAT}(F) = 2^n - |\bigcup_{i=1}^{m} K(C_i)|. \qquad (5)$$

Donde $K(C)$ denota al conjunto de asignaciones sobre $Var(F)$ que hacen falsa a la cláusula C.
$K(C) = \{S \in \{0,1\}^n | C(S) = Falso\}$.

La propiedad de independencia entre cláusulas y de cláusulas subsumidas permitió decrementar el número de sumandos cuando se aplica la fórmula de inclusión-exclusión para calcular $|\bigcup_{i=1}^{m} K(C_i)|$.

Se realizó el análisis de la complejidad en tiempo del algoritmo resultante, considerando tanto los peores casos como el caso promedio. Nuestra propuesta algorítmica tiene un número esperado de

$$\sum_{t=1}^{\lceil m[(1-k/2n)^k-(k/2n)^{k-1}]\rceil} \binom{m}{t} \qquad (6)$$

operaciones de intersección y/o unión entre conjuntos con t cláusulas.

Esta cota se comparó contra otros factores de complejidad en el caso promedio obtenidos por algoritmos basados también en el principio de inclusión-exclusión, resultando que nuestra cota mejora la complejidad en tiempo para el promedio de los casos reportada por Lozinskii [9].

4.2 Aplicando técnicas Monte Carlo

Se diseñó también un algoritmo basado en las técnicas Monte Carlo para aproximar soluciones al problema #SAT. El algoritmo de aproximación recibe como entrada una fórmula booleana F y los parámetros (ε, δ) y como solución, calcula una estimación z al valor #SAT(F), donde se cumple que:
$\text{Prob}[(1-\varepsilon)\cdot\#\text{SAT}(F) \leq z \leq (1+\varepsilon)\cdot\#\text{SAT}(F)] \geq (1-\delta)$.

El análisis del algoritmo permitió determinar una cota superior del número de ejemplos requeridos en el muestreo, de tal forma que la solución obtenida cumpla los factores de aproximación (ε, δ) requeridos. El número de ejemplos (iteraciones del algoritmo) multiplicado por el valor de la complejidad en tiempo invertido en cada iteración, determinó una complejidad en tiempo de $O(nm^2 \ln(2\delta^{-1})\varepsilon^{-2})$ para dada como entrada una fórmula $F \in \mathcal{F}(m,n)$.

El algoritmo de aproximación propuesto es un ejemplo de como explotar, dada una fórmula booleana F, la relación (5), y sugiere una línea amplia de experimentación de la aplicación de las técnicas Monte Carlo en la resolución del problema #SAT, lo que puede incidir en la determinación de mejores factores de aproximación para resolver (aproximar) problemas de la clase #P.

Ambas propuestas: aplicación de la fórmula de inclusión-exclusión para resolver de manera exacta a #SAT y la aplicación de técnicas Monte Carlo para hallar soluciones aproximadas de #SAT, son propuestas alternativas a otras basadas en los mismos principios [3,7,9].

5 Algoritmo propuesto para #2-SAT

La contribución principal de esta investigación estriba en el diseño de un algoritmo de ramificación y acotamiento para #SAT(F), cuando F es una fórmula booleana en 2-FC. El algoritmo propuesto construye un árbol de enumeración. A cada nodo del árbol se le asocia una pareja (S, F'), siendo S una asignación parcial, $F' \subseteq F$, y se cumple que: $S \cap Lit(F') = \emptyset$.

Para una cláusula C, diremos que $N(C) = \{\bar{l} \mid l \in C\}$ es su *contrapuesta*. La cláusula C es satisfecha por una asignación S si: $C \cap S \neq \emptyset$. C es falsificada por S si $N(C) \subseteq S$, o similarmente, si $N(C)$ - $S = \emptyset$.

Dado un nodo del árbol con pareja (S, F') asociada, el esquema de ramificación se basa en la selección de una cláusula $C \in F'$, a la que sin que pérdida de generalidad denotaremos por $\{l_1, l_2\}$. Hay tres asignaciones que extienden a S y satisfacen a C, las cuales son:
$S_1 = S \cup \{l_1, \bar{l}_2\}, S_2 = S \cup \{\bar{l}_1, l_2\}$ y $S_3 = S \cup \{l_1, l_2\}$.

Nótese que la unión de estas tres nuevas asignaciones representa la combinatoria en signos de las variables que aparecen en C y que conforman las diferentes maneras posibles de satisfacer a C.

Construidas las tres asignaciones, el nodo actual se ramifica con tres nodos hijos, y a cada nodo hijo ν_i se le asocia la pareja (S_i, F'), $i = 1, 2, 3$, y se procede a simplificar la fórmula F' y a extender la asignación S_i, según el procedimiento siguiente:

1. Si $C \in F'$ es tal que $S_i \cap C \neq \emptyset$ entonces no se modifica S_i.

2. Si $N(C) - S_i = \emptyset$ entonces el correspondiente nodo de la asignación S_i se declara como nodo hoja.

3. Si $N(C) - S_i = \{\bar{l}\}$ entonces $S_i = S_i \cup \{l\}$.

   En cualquiera de los tres casos se elimina a C de F' y se continua aplicando el procedimiento hasta que $Var(S_i) \cap Var(F') = \emptyset$.

Después de aplicar el procedimiento, en cada nodo hijo ν_i se obtiene una nueva pareja (S'_i, F_i), siendo S'_i la extensión de la asignación S_i y F_i la subfórmula resultante de F' por la eliminación de cláusulas según el procedimiento.

Como función de acotamiento el algoritmo usa un procedimiento de conteo que permite detener el proceso de ramificación de nodos, al inferirse el número de asignaciones parciales que satisfacen a F sin tener que construirlas explícitamente. La función de corte evita tener que hacer una enumeración exhaustiva de todas las posibles asignaciones que tiene la fórmula de entrada.

En el algoritmo se reconocen cuatro casos (llamados casos base) que permiten calcular en tiempo polinomial (especificamente, a lo más en tiempo lineal) el número de asignaciones parciales que extienden la asignación actual y que satisfacen a F.

Un nodo se marca como terminal (nodo hoja del árbol) si su pareja asociada $(S,\, F')$ es un caso base. Una pareja (S, F') es un caso base, si cumple alguna de las condiciones siguientes:

1. $S = \emptyset$
2. $F' = \emptyset$
3. $\mid Var(F') \mid \leq 2$
4. F' es una fórmula del tipo 2,2-FC

La complejidad en tiempo del algoritmo depende directamente de la altura del árbol y, esto a su vez, depende del reconocimiento de los casos base. Una línea de investigación futura es la identificación de más casos base. Enfasis especial será hallar instancias más generales que las fórmulas F del tipo 2,2-FC donde el valor #SAT(F) pueda computarse eficientemente, lo que incidirá en una reducción sustancial de la complejidad en tiempo del algoritmo de ramificación y acotamiento.

Dada como entrada al algoritmo una fórmula $F \in \mathcal{F}(m,n)$ en 2-FC, el análisis detallado de la complejidad en tiempo mostró que en los peores casos, la complejidad es $\mathrm{O}(nmr^n)$, siendo r la raíz real positiva del polinomio $p(r) = r^4 - r^2 - r - 1$ y la cual, estimada con una precisión de siete dígitos decimales, es: $r \simeq 1.4655713$. Esto establece una nueva cota inferior a la complejidad en tiempo de los algoritmos que resuelven #2-SAT.

5.1 El problema #2,2-SAT

Considerando el problema #SAT, existen muy pocos resultados positivos, en el sentido de conocer para que tipo de fórmulas booleanas #SAT pueda resolverse eficientemente. Actualmente, en la literatura sólo aparece que el caso #1-SAT es trivialmente resoluble.

Una de las contribuciones principales de este trabajo fue la determinación y caracterización de subclases específicas de fórmulas booleanas donde el problema #SAT se resuelve eficientemente. Esto es de suma importancia ya que puede incidir en el esclarecimiento del borde entre problemas tratables e intratables.

Como resultado de nuestras investigaciones, se diseñó un algoritmo de tiempo lineal sobre la fórmula de entrada que resuelve el problema #2,2-SAT. Estableciéndose entonces que para toda fórmula booleana F del tipo 2-FC donde toda variable aparece a lo más dos veces en la fórmula, el problema de conteo de satisfactibilidad asociado (#2,2-SAT) se resuelve eficientemente, y de hecho, se resuelve por un algoritmo de complejidad lineal en tiempo. Un punto importante a investigar a futuro, es la extensión de la técnica utilizada en este algoritmo, tal que se pueda aplicar a la resolución de #SAT, considerando como entrada instancias más generales que las fórmulas en 2,2-FC.

Bibliografía

[1] De Ita G., Morales G., Propuestas algorítmicas en el tratamiento de los problemas de satisfactibilidad, Reporte técnico serie verde No. 54, Dpto. de Ingeniería Eléctrica, CINVESTAV-IPN, 1996.

[2] De Ita G., Pinto D., Nuño M., Heuristics for improving the non-oblivious local search for MaxSAT, *Lecture Notes in Computer Science 1484*, 219-229, 1998.

[3] Dubois Olivier, Counting the number of solutions for instances of satisfiability, *Theor. Comp. Sc.* 81, (1991), 49-64.

[4] Garey M.R. and Johnson D.S., *Computers and Intractability: A Guide to the theory of NP-Completeness*, W.H. Freeman and Co., New York, 1979.

[5] Gu J., Purdom P., Franco J., Wah B., *Algorithms for the satisfactibility (SAT) Problem*, Handbook of Combinatorial Optimization D.-Z. Du and P.M. Pardalos Eds., 1998.

[6] Hansen P., Jaumard B., Algorithms for the Maximum Satisfiability Problem, *Computing 44*, 279-303, 1990.

[7] Karp R., Luby M., Madras N., Monte-Carlo Approximation Algorithms for Enumeration Problems, *Journal of Algorithms 10*, 429-448, 1989.

[8] Khanna S., Motwani R., Sudan M., Vazirani U., On syntactic versus computational views of approximability, TR95-023 ECCC Elect. Colloq. on Comp. Complexity, 1995.

[9] Lozinskii E., Counting propositional models, *Inf. Proc. Letters 41*, (1992), 327-332.

[10] Valiant L.G., The complexity of enumeration and reliability problems, *SIAM J. Comput.* Vol.8, No.3, (1979), 410-421.