ISSN 1870-4069

Algoritmo para el conteo de modelos en FNC

Pedro Bello López, Guillermo De Ita Luna, Meliza Contreras González, Miguel Rodríguez Hernández

Benemérita Universidad Autónoma de Puebla,
Facultad de Ciencias de la Computación,
México

{pb5pbello, vikax68, mikelrodriguezh}@gmail.com,
comdeita@inaoep.mx,

Resumen. En este documento se presenta un algoritmo para el conteo de modelos de una fórmula especificada en forma normal conjuntiva (FNC), utilizando patrones representados como cadenas de ceros, unos y asteriscos lo que permite un mejor tratamiento computacional. Además, con el algoritmo desarrollado se generó una aplicación web para realizar pruebas con fórmulas en FNC. La estrategia utilizada en la propuesta algorítmica se basa en considerar el mismo conjunto de variables para todas las cláusulas, de esta forma podemos saber el total de modelos, donde el máximo es 2^n. El proceso consiste en aplicar el operador de revisión mediante los patrones mencionados para reducir los modelos, por lo que al final del proceso se obtiene el total de modelos de F.

Palabras clave: Algoritmo, Satisfactibilidad, #SAT, NP-Completo, Sistema.

Algorithm for Counting Models in FNC

Abstract. This document presents an algorithm for the counting of models of a formula specified in conjunctive normal form (FNC), using patterns represented as strings of zeros, ones and asterisks, which allows a better computational treatment. In addition, with the developed algorithm, a web application was generated to perform tests with formulas in FNC. The strategy used in the algorithmic proposal is based on considering the same set of variables for all the clauses, in this way we can know the total number of models, where the maximum is 2n. The process consists of applying the revision operator through the aforementioned patterns to reduce the models, so at the end of the process the total number of F models is obtained.

Keywords: Algorithm, Satisfiability, #SAT, NP-Complete, System.

1. Introducción

El problema de satisfactibilidad (SAT) es considerado el primer problema NP-completo demostrado por Stephen Cook en el año de 1971. El problema SAT es 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. El problema SAT en general es un problema de decisión, lo definimos como: dada una formula booleana F en Forma Normal Conjuntiva (FNC), formada de un conjunto de cláusulas con n variables, se debe encontrar una asignación a dichas variables talque la F sea satisfactible, es decir que exista al menos una asignación de valores que haga verdadera la formula F.

Los problemas NP-completo son los problemas que caracterizan la clase de tal forma que al encontrar una solución del problema SAT se pueden resolver los demás problemas de la clase ya que se puede hacer una transformación en tiempo polinomial al problema correspondiente.

De esta forma el problema SAT consiste en decidir si F es satisfactible. Mientras que el problema de optimización MaxSAT consiste en determinar el número máximo de cláusulas que se pueden satisfacer simultáneamente [1]. Y el problema de conteo #SAT consiste en contar el número de asignaciones que satisfacen a *F*. Nuestra propuesta algorítmica propone un método para #SAT, es decir presentamos un algoritmo y su implementación computacional para contar el número de asignaciones (modelos) de una formula *F* en FNC.

Consideramos para cada cláusula el mismo conjunto de *n* variables y formamos cadenas de ceros, unos y asteriscos (*), el * se usa cuando una variable no aparece en la cláusula, posteriormente aplicamos un proceso de revisión para generar clausulas independientes entre sí, es decir cláusulas que tengan una literal contradictoria. El método es iterativo tomando una a una las cláusulas de la formula F iniciando con el valor de 2^n modelos y se le van restando modelos de acuerdo al número de asteriscos ($2^{\#ast}$) de las cláusulas resultantes al aplicar el proceso de revisión. En la Fig.1 se muestra el proceso completo para la propuesta de conteo de modelos, la transformación de la formula en FNC a su representación de ceros, unos y * se realiza de forma manual por el usuario del sistema.

Dentro de las múltiples aplicaciones de #SAT, se encuentran las aplicaciones sobre el razonamiento proposicional, el cual constituye una pieza fundamental de la Inteligencia Artificial. Los problemas de satisfactibilidad SAT, MaxSAT y #SAT son problemas completos en las clases de complejidad NP, MAX NP y #P [2]. Un punto importante al trabajar con SAT, MaxSAT y #SAT en nuestro caso, es que, como representantes de sus respectivas clases de complejidad, hallar nuevas estrategias algorítmicas que los resuelven, implica que tales estrategias se pueden extender a los demás problemas de la clase correspondiente.

En este trabajo se presenta en la sección 2 el trabajo relacionado con el tópico del problema SAT sus principales aplicaciones, sus alcances y limitaciones. Posteriormente en la sección 3 se presenta la propuesta algorítmica para el conteo de modelos de una fórmula en FNC, se incluyen algunas consideraciones iniciales para el desarrollo del

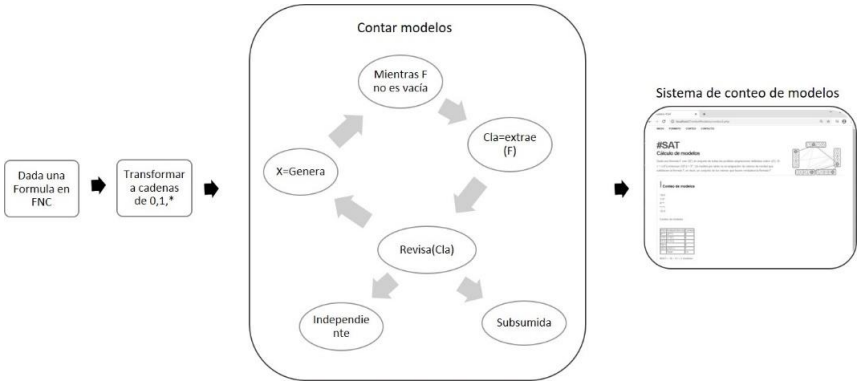

Fig. 1. Proceso general para el conteo de modelos.

algoritmo y a través de ejemplos se describe el algoritmo desarrollado. En la sección 4 se presentan los resultados del algoritmo implementado en una aplicación web y finalmente en las conclusiones se describe la idea general de la complejidad en tiempo de la propuesta del algoritmo para el conteo de modelos en FNC.

2. Trabajo relacionado

Los problemas de satisfactibilidad son estudiados intensivamente en las ciencias computacionales, debido a que son sin duda, los problemas más importantes en el campo de la complejidad computacional [3-4m], y una de las aplicaciones interesantes es cuando contribuyen significativamente a simplificar un problema que por naturaleza incluye procesos exponenciales. Existen diversas líneas de investigación en torno a estos problemas y las aplicaciones son muy variadas.

Dentro de los problemas de satisfactibilidad de restricciones, se encuentra el problema de decisión, que consiste en determinar si una fórmula es satisfactible o no. El problema de conteo asociado, es saber el número de asignaciones que satisfacen a una formula Booleana, conocido como el problema #SAT. Existen muchos problemas no resueltos en su totalidad que se derivan del problema #SAT, por lo que se tiene un constante interés por parte de los investigadores de diversas áreas, por encontrar diferentes formas de resolver el problema #SAT, en parte, debido al impacto que tendría una buena solución de este problema en muchas otras disciplinas y aplicaciones. Sabemos que en su completa generalidad, #SAT es un problema de la clase #P-completo. No se tiene aún demasiado conocimiento sobre las restricciones a #SAT donde su solución sea eficiente, esta es una de las razones por las que muchos investigadores están investigando en esta are de la ciencia.

Dentro de las múltiples aplicaciones de #SAT, se encuentran las aplicaciones sobre el razonamiento proposicional, el cual constituye una pieza fundamental de la Inteligencia Artificial y en los sistemas de agentes y agentes inteligentes que operan de forma robusta ante cambio repentinos impredecibles [4-1m].

En [5-2m] prueban que contar asignaciones falsificantes de lenguajes proposicionales es intratable aún para Horn y fórmulas monótonas, y aun cuando el tamaño de las cláusulas y el número de ocurrencias de las variables son extremadamente limitadas. Esto se debería contrastar con el caso de razonamiento deductivo, donde las teorías de Horn y las teorías con cláusulas binarias son reconocidas por la existencia de algoritmos de satisfactibilidad en tiempo lineal. Lo cual sorprende que aun aproximando el número de asignaciones que satisfacen es intratable para la mayoría de estas teorías restringidas, es decir, "aproximar" al razonamiento aproximado. También identifica algunas clases limitadas de fórmulas proposicionales para las cuales se pueden dar algoritmos eficientes para contar asignaciones satisfactorias.

En [6-3m] muestra cómo se extiende el problema de intratabilidad con el método de Monte Carlo de cadenas de Markov, así también resaltar los límites naturales del método. Utiliza técnicas de acoplamiento antes de introducir "acoplamiento del camino" una nueva técnica, la cual, simplifica radicalmente y mejora sobre métodos previos en el área.

En [5-2m] y [6-3m] se tiene el inconveniente de que no garantizan la complejidad en tiempo lineal, además de que al contar los modelos de Σ, no distinguen entre los modelos en los que una variable *x* toma el valor de 1 de aquellos en el que la misma variable *x* toma el valor de 0.

Se presentan algoritmos para contar modelos proposicionales del problema #SAT [7-5m]. Los algoritmos utilizan descomposiciones arbóreas de ciertos grafos asociados con la formula FNC dada; en particular consideran grafos de incidencia, dual y primordial. Describe el algoritmo coherentemente para una comparación directa con suficiente detalle para realizar una implementación razonablemente fácil. También discute varios aspectos de los algoritmos incluyendo tiempo del caso peor y requerimientos de espacio de almacenamiento. Podemos mencionar que existen algoritmos o técnicas de conteo de modelos. En [8-6m] se dividen las técnicas de conteo práctico de modelos en dos categorías: conteo aproximado y conteo exacto.

En [9-7m] se introduce el problema Max#SAT, una extensión del conteo de modelos (#SAT). Dada una formula sobre el conjunto de variables X, Y y Z, el problema Max#SAT es maximizar sobre las variables X el número de asignaciones a Y que se puede extender a una solución con algunas asignaciones a Z. Muestra también que Max#SAT tiene aplicaciones en muchas áreas mostrando como se puede utilizar para resolver problemas en inferencia probabilística, planeación, síntesis de programas y análisis de flujo de información cuantitativa. También proporciona un algoritmo el cual, haciendo únicamente muchas llamadas polinomialmente a una máquina oráculo NP puede aproximar el conteo máximo dentro de un error multiplicativo válido.

Se desarrolla GANAK, un contador de modelos exacto probabilístico escalable que supera los contadores de modelos aproximados y exactos, ApproxMC3 y #SAT respectivamente. En sus experimentos, el conteo de modelos regresado por GANAK fue igual al conteo exacto de modelos para todos los puntos de referencia [10-8m].

Existen algoritmos exhaustivos de última generación para contar modelos, por ejemplo, un algoritmo DPLL con semántica formal [11], una prueba de la corrección y diseño modular. El diseño modular se base en la separación del núcleo principal del

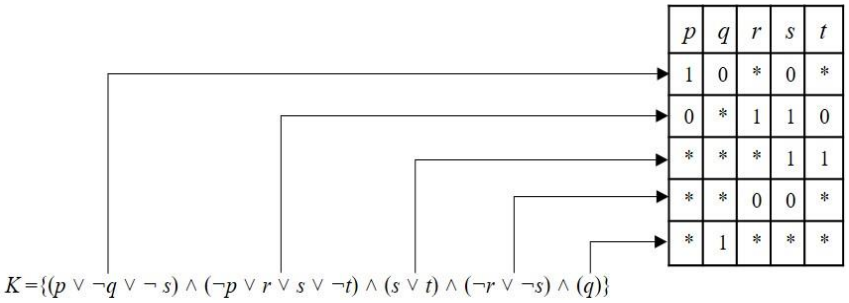

Fig. 2. Transformación de una formula en FNC a cadenas de 0,1 y *.

algoritmo de conteo de modelo de las técnicas de solución de SAT, como se aborda en [12].

Recientemente los modelos probabilísticos para contar modelos han tenido mucha aceptación e importancia como el algoritmo que es inspirado en una estimación estadística para refinar continuamente una estimación probabilística del conteo de modelos para una fórmula, así cada consulta XOR-racionalizada dé como resultado tanta información como sea posible [13].

Los solucionadores modernos SAT también están causando un impacto significativo en los ámbitos de la verificación de software, la resolución de las limitaciones en la inteligencia artificial, la investigación operativa, así como el diagnóstico de errores en circuitos digitales [14] y la variabilidad de los sistemas configurables [15]. Muchos otros problemas de la decisión, como los problemas de coloración de grafos, problemas de planificación y programación de problemas, puede ser codificado en SAT.

3. Algoritmo para el conteo de modelos

Algunas consideraciones básicas para el algoritmo de conteo de modelos son:

- Una formula *F* en FNC, es una conjunción de cláusulas.
- Una cláusula está formada por un conjunto de *n* variables.
- Una literal es una variable con valor *x* o su negación ¬*x*.
- Cada cláusula es mapeada a un conjunto de 0,1 y * como se muestra en la Fig. 2. Cuando una literal no aparece se coloca un asterisco (*) lo que significa que puede cualquiera de dos valores {0,1}.
- Un modelo es una asignación de valores para las literal que al realizar la conjunción de las cláusulas obtenemos un valor de verdadero (satisfactible) para la fórmula *F*.
- El problema #SAT(*F*) entonces es el total de modelos que hacen satisfactible la fórmula *F*.

Cuando transformamos una cláusula *C* en base al mismo conjunto de *n* variables, tenemos que el número de modelos (*Mod*) satisfactibles está dado por $Mod(C) = 2^n - 2^{Ast(C)}$, donde *Ast*(*C*) es el número de asteriscos de la cláusula *C*. Por ejemplo sea *C*=(1***0), donde *n*=5 variables, entonces $Mod(C)=2^5-2^3=32-8=24$.

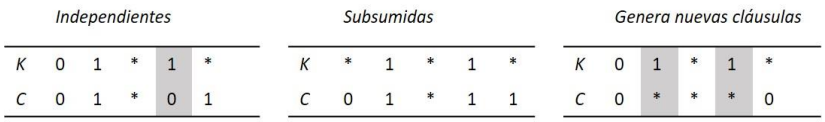

Fig. 3. Operaciones básicas para el algoritmo de conteo de modelos.

Dentro del proceso general del conteo de modelos utilizamos tres operaciones básicas:

- **Clausulas independientes *Ind()*:** dos cláusulas (*K*, *C*) son independientes si tienen variables contrarias para una misma posición en dichas cláusulas.
- **Clausula subsumida *Sub()*:** una cláusula *C* es subsumida en *K* es subsumida si los modelos de *C* están contenidos en *K*.
- **Generación de nuevas cláusulas *Gen()*:** dadas *K* y *C* dos cláusulas se generan *NC* cláusulas si C_i=* y $\mathbf{K}_i$ es una literal con un valor diferente de *.

La Fig. 2 muestra las tres operaciones básicas para el conteo de modelos, la operación de generación de nuevas cláusulas puede generar de 1 a máximo *m-1* cláusulas nuevas. La propuesta de algoritmo de conteo de modelos es:

Entrada:

Sea $F=\{f_1, f_2, \ldots, f_m\}$ un conjunto de *m* cláusulas en FNC, con *n*=número de variables de cada cláusula

Salida:

S,conjunto de cláusulas resultantes del conteo

TotalMod, total de modelos

1: **Inicio**
2: *TotalMod*=$2^n - 2^{Ast(f_1)}$
3: *S*=extraer(*F*);
4: **Para cada** Cláusula f_i **en** *F*, i=2, **hasta** *m* **Hacer**
5: *C*=f_i
6: **Mientras** *S* no este vacío **hacer**
7: *in=Ind(S,C)*; //verifica independencia
8: *nc=Gen(S,C)*; //obtienen # de nuevas cláusulas
9: **Si** (*in*==1 y *vacia*(*F*)) *S=añadir(C)*;
10: **Sino Si** (nc>0)
11: **Para** i←0 hasta nc
12: *F*=añadir(*nueva_clausula_i)*;
13: **FinPara**
14: **Sino** *S*=extraer(*F*); // cláusula subsumida
15: **FinSi**
16: **FinSi**
17: *x=#Ast*(F_i); // $2^{\#\text{ de asteriscos de una clausula}}$
18: *TotalMod=TotalMod-x*;
19: **FinMientras**
20: **FinPara**
21: **Fin**

Las operaciones de *Ind()* y *Gen()* corresponde a ciclos simples de 1 a *n* variables que permite determinar la operación que se debe aplicar, las operaciones son excluyentes para cada cláusula. Cuando se llega al caso de cláusula subsumida se detiene el ciclo para continuar con otra cláusula de la fórmula *F*.

La generación de nuevas cláusulas aumenta la revisión con las nuevas cláusulas generadas y para el caso de independencia al terminar de revisar toda la fórmula *F* se agrega a la salida *S* que representa la fórmula reducida para el conteo de modelos.

3.1 Aplicación del algoritmo

Para ejemplificar la ejecución del algoritmo consideremos el siguiente ejemplo. Sea F= {($p \vee \neg r$) $\wedge$ ($\neg q \vee \neg r \vee s$) $\wedge$ ($\neg p$) $\wedge$ ($q \vee s$) $\wedge$ ($p \vee \neg q \vee r \vee$**s**)} una fórmula en FNC con m=5 cláusulas y n=4 variables. Transformamos cada clausula con el patrón de ceros, unos y asteriscos y obtenemos F = {(1*0*) (*001) (0***) (*1*1) (1011)}.

Aplicando el algoritmo inicia con:

S = {(1*0*)}; S representa el resultado del proceso de revisión de la fórmula almacenada en F.

TotalMod = 16 - 2^2 = 16 - 4 = 12; *TotalMod* inicia con el máximo de modelos posibles de F (16 modelos) menos los modelos de la primera cláusula (4).

En la Tabla 1 se muestra la ejecución del algoritmo de conteo de modelos, se observa que en el paso i=3 se generaron 3 nuevas cláusulas debido a que la cláusula F_3 tiene más asteriscos.

Podemos reducir el número de comparaciones si aplicamos un ordenamiento en base al número de asteriscos de cada cláusula. Consideremos el mismo ejemplo de la Tabla 1. F = {(1*0*) (*001) (0***) (*1*1) (1011)}, aplicando ordenamiento, obtenemos F = {(0***) (1*0*) (*1*1) (*001) (1011)}. Aplicando el algoritmo de conteo obtenemos:

S = {(0***)}; S almacena la primera cláusula de la fórmula F.

TotalMod = 16-2^3 = 16-8 = 8; representa el inicio del conteo.

Tabla 1. Conteo de modelos.

i	***F_i***	***S***							***x***	***TotalMod***
2	*001	1*0* 0001						0001	1	12-1=11
3	0***	1*0* *Ind()*	0001 01** 001* 0000					01** 001* 0000	4 2 1	11-7=4
4	*1*1	1*0* 01*1 1111	0001 *Ind()* *Ind()*	01** *Sub()* *Ind()*	001* *Ind()*	0000 *Ind()*		1111	0 1	4-0=4 4-1=3
5	1011	1*0* *Ind()*	0001 *Ind()*	01** *Ind()*	001* *Ind()*	0000 *Ind()*	1111 *Ind()*	1011	1	3-1=2

S = {(1*0*) (0001) (01**) (001*) (0000) (1111) (1011)}

El total de modelos de la fórmula F = Mod(F) = 2.

Table 2. Conteo de modelos aplicando ordenamiento.

i	***F_i***	***S***				***x***	***TotalMod***
2	1*0*	0***					
		Ind()			1*0*	4	8-4=4
3	*1*1	0***	1*0*				
		11*1	1111		1111	1	4-1=3
4	*001	0***	1*0*	1111			
		1001	*Sub()*			0	3-0=3
5	1011	0***	1*0*	1111			
		Ind()	*Ind()*	*Ind()*	1011	1	3-1=2

S = {(0***) (1*0*) (1111) (1011)}
El total de modelos de la fórmula $F = Mod(F) = 2$.

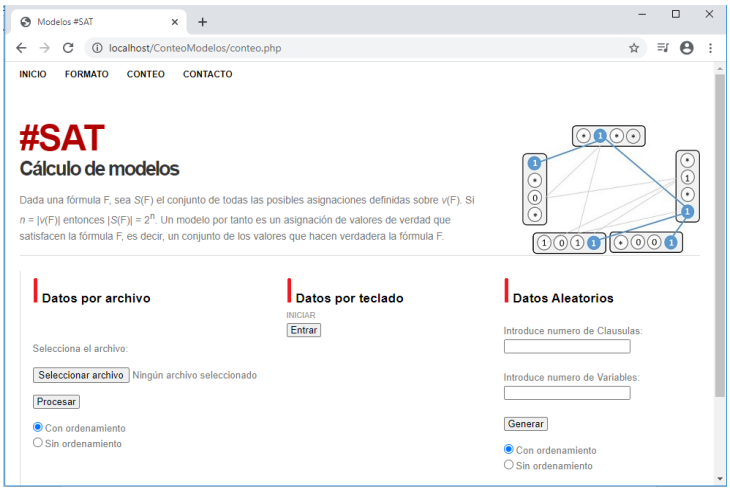

Fig. 4. Pantalla para el conteo de modelos.

La Tabla 2 muestra la ejecución del algoritmo de conteo de modelos aplicando un ordenamiento descendente de acuerdo al número de asteriscos de cada cláusula. Se reduce el número de comparaciones debido a que tomamos como cláusula inicial (0***) que elimina más modelos al iniciar el proceso de revisión.

4. Resultados

Como resultado del diseño del algoritmo de conteo de modelos se cuenta con un sistema en web implementado en el lenguaje de programación PHP. El sistema permite realizar pruebas del conteo de modelos a través de archivos de texto, de forma manual o generando entradas aleatorias ver Fig. 4. El sistema se ejecuta en un servidor local.

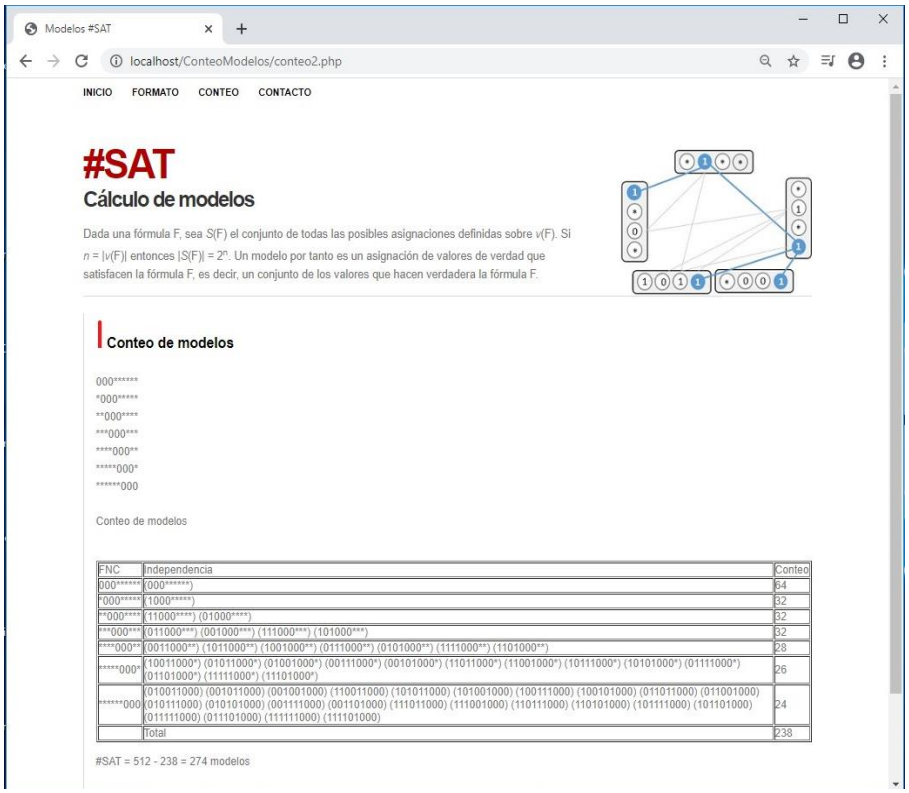

Fig. 5. Ejemplo de la salida del conteo de modelos.

Para realizar el cálculo utilizando la entrada de archivo, se debe tener un archivo de texto, donde en la primera línea debe haber el número de cláusulas y el número de variables por cláusula, posteriormente se coloca una cláusula por línea usando la representación de 0,1 y asteriscos. Por ejemplo, considere el archivo entrada9.txt con 7 cláusulas y 9 variables:

```
7 9
000******
*000*****
**000****
***000***
****000**
*****000*
******000
```

La Fig. 5 muestra el cálculo del conteo de modelos para el archivo entrada9.txt, se muestra la entrada original y en la tabla el conjunto de cláusulas que se van generando, se va reduciendo paso a paso el número de modelos. El sistema web de forma automática crea un archivo de texto (salida.txt) para guardar los resultados obtenidos de cada ejecución.

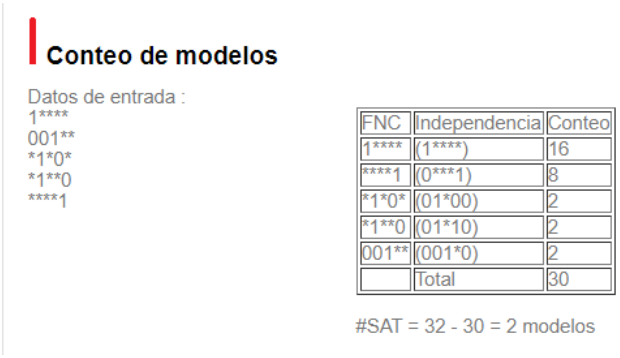

Fig. 6. Conteo de modelos del ejemplo del asistente virtual *A*.

Tabla 3. Ejemplos de ejecución de algoritmos para entradas aleatorias.

Cláusulas	Variables	Modelos
10	10	1024 - 128 = 896
20	10	1024 - 231 = 793
50	20	1048576 - 11741 = 1036835
100	20	1048576 - 20243 = 1028333
200	30	1073741824 - 1065208 = 1072676616
300	40	1099511627776 - 41099096 = 1099470528680

En la Tabla 3 se describe varios ejemplos de ejecución del algoritmo de conteo de modelos en forma aleatoria. Para estos ejemplos el tiempo de ejecución no es significativo ya que se utilizó en promedio 1.5 segundos.

En el área de razonamiento automático se modelan problemas con lógica proposicional, por ejemplo, dado el siguiente conjunto de proposiciones de un asistente virtual *A*:

p : comprar pañales

q: pañales de la marca

r: marca blanca

s: cantidad requerida

t: paquete individual

Formamos el conjunto de instrucciones del asistente virtual, *A*={comprar pañales, no hay pañales o no hay pañales de la marca o hay marca blanca, no hay la cantidad requerida o hay pañales de la marca, no hay paquete individual o hay pañales de marca, hay paquete individual}, trasformando *A* en su FNC, tenemos que $A = \{(p) \wedge (\neg p \vee \neg q \vee r) \wedge (\neg s \vee q) \wedge (\neg t \vee q) (t)\}$, convertimos *A* en su patrón correspondiente de 0,1 y *, obteniendo *A*={(1****) (001**) (*1*0*) (*1**0) (****1)}. La Fig. 6 muestra el conteo de modelos de la formula A utilizando la entrada por teclado.

5. Conclusiones

Se presenta una propuesta algorítmica para el conteo de modelos de una formula en FNC utilizando una representación de cadenas de 0,1 y *'s. Esta representación nos permite mejorar el proceso de conteo de modelos de manera algorítmica.

Con el objetivo de proveer un mecanismo de pruebas, se generó una versión en web del algoritmo que permite hacer cálculos con datos de teclado, de archivo o de forma aleatoria.

La complejidad en tiempo del algoritmo viene dada por el número de *m* cláusulas y el número de *n* variables de cada cláusula, se procesa cada cláusula una a una y se generan a lo más *n-1* nuevas cláusulas que también son revisadas para generar independencia, en el peor caso se procesan *m***n* *(*n*-1, *n*-2, *n*-3, ... 1) operaciones, debido a que una cláusula puede tener *n-1* asteriscos (para cláusulas unitarias), y cada una de estas nuevas cláusulas pueden generar n-2 nuevas cláusulas a procesar. Al implementar el sistema utilizamos 3 ciclos anidados de orden *m***n*. La complejidad algorítmica en general del problema de satisfactibilidad para fórmulas a partir de 3 variables es exponencial, sin embargo, nuestro algoritmo, aunque sigue siendo exponencial puede ser acotado por un parámetro en términos del número de cláusulas y el número de variables de cada clausulas.

Este tipo de algoritmos de conteo de modelos puede ser utilizado en el área de razonamiento automático, en bases de conocimiento en la revisión de creencias y otras aplicaciones de la inteligencia artificial.

Referencias

1. De-Ita, G., Gómez, H., Merino, B.: Algorithm to count the number of signed paths in an electrical network via boolean formulas. Acta Universitaria, 22, pp. 69−74, Universidad de Guanajuato (2012)
2. Garey, M.R., Johnson, D.S.: Computers and intractability: A Guide to the theory of NP-Completeness, W.H. Freeman and Co., New York (1979)
3. Papadimitriou, C.H.: Computational complexity. Addison Wesley (1994)
4. Guillen, C., López, A., De-Ita G.: Diseño de algoritmos combinatorios para #SAT y su aplicación al razonamiento proposicional, Reporte Técnico No. CCC-05-005 (2005)
5. Roth, D.: On the hardness of approximate reasoning, Artificial Intelligence, pp.273−302 (1996)
6. Russ, B.: Randomized algorithms: Approximation, generation, and counting. Springer-Verlag London Berlin Heidelberg (2001)
7. Samer, M., Szeider, S.: Algorithms for propositional model counting. Elsevier (2009)
8. Gomes, C., Sabharwal, A., Selman, B.: Model counting, handbook of satisfiability. 20, IOS Press (2008)
9. Fremont, D., Rabe, M., Seshia, S.: Maximum model counting. In: Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI), pp. 3885–3892 (2017)
10. Sharma, S., Roy, S., Soos, M., Meel, K.S.: GANAK: A scalable probabilistic exact model counter. In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI´19), pp. 1169−1179 (2019)

11. Silva, F., Neves, J., Dias, S., Zárate, L., Song, M.: A hybrid approach to solve SAT and UNSAT problems. IEEE Latin America Transactions, 18(4) (2020)
12. Oztok, U., Darwiche, A.: An exhaustive DPLL algorithm for model counting. Journal of Artificial Intelligence Research, 62, pp 1−32 (2018)
13. Kim, S., McCamant, S.: Bit-vector model counting using statistical estimation, tools and algorithms for the construction and analysis of systems. 10805, pp. 133−151 (2018)
14. Gaber, L., Hussein, A. Mahmoud, H. Mourad. M.: Mohammed moness computation of minimal unsatisfiable subformulas for SAT-based digital circuit error diagnosis. Journal of Ambient Intelligence and Humanized Computing Springer-Verlag (2020)
15. Sundermann, C., Thüm, T., Schaefer, I.: Evaluating #SAT solvers on industrial feature models. In: ACM International Conference Proceeding Series, a3 (2020)