# Resolvedor SAT, basado en procedimientos Davis-Putnam-Longemann-Loveland

Pedro Victor Gabriel Cerna
Director: Robert Nieuwenhuis
Departamento de Lenguajes y Sistemas Informáticos
Universidad Politécnica de Cataluña (UPC)
Barcelona, España

**Resumen**

El problema de satisfación de fórmulas lógicas ($SAT$), es un problema NP-Hard. Una forma de resolverlo es por medio de procedimientos Davis-Putnam-Longemann-Loveland (DPLL), ahora presentamos una implementación de un resolvedor SAT a partir de procedimientos DPLL.

## 1. Introducción

En teoría de la complejidad computacional, el Problema de satisfabilidad booleana (SAT) es un problema perteneciente a la clase de complejidad NP-completo [1,2].

El problema SAT consiste en decidir, cuando una formula booleana es satisfacible. Por su propiedad de ser NP-Hard, este problema en el peor de los casos se piensa que no puede ser resuelto en tiempo polinomial, sin embargo existen muchas algoritmos heurísticos propuestos para dar soluciones.

El problema SAT básicamente es:

Dado un conjunto de literales $L = l_1, ..., l_i, ...l_{i+m}, ..., l_n$ donde $l_n$ es la n-esima literal, un conjunto de cláusulas $C = c_1, c_2, ..., c_i, ...c_n$ formadas por la disjunción de literales o sus negaciones ($c_i = l_1 \bigvee l_i \bigvee ... \bigvee \neg l_{i+m}$), y una asignación de verdad M, el problema consiste en saber si dada una formula F formada de conjunciones ($\bigwedge$) de cláusulas $c$ existe una asignación de verdad que satisfaga a la formula verdadera, de esta forma diremos que es satisfacible en cualquier otro caso le llamaremos no satisfacible.

Las aplicaciones practicas de este problema van desde areas como automatización de diseño electrónico hasta la Inteligencia artificial, En la actualidad existen muchos resolvedores SAT, como los que hallamos en [3,4,5,6,7], algunos de ellos basados en variaciones de procedimientos DPLL ver [8,9,10]

Es importante mencionar que las mejoras en el de manejo de conflictos a través de reglas de resolución para hallar puntos de retorno sobre una asignación para la regla backjump y la $two - watchedliteralapproach$ para la regla de unit propagation a demás de otras mejoras al procedimiento DPLL clásico, permiten alcanzar una mejora de la exploración de los espacios de búsqueda, haciendo posible su uso en la resolución de problemas con un número superior a miles de variables y cláusulas

La meta del presente trabajo es la implementación de un resolvedor SAT, basado en un procedimiento Davis-Putnam-Logemann-Lovelend (DPLL)completo y corregido. Dividiremos este reporte en secciones, la primera definirá el procedimiento DPLL completo y corregido, el manejo de conflictos y formas usadas para realizar la propagación de literales en una asignación M, la segunda presentara el algoritmo de programa y la tercera presentara algunos resultados obtenidos.

# 2. Procedimiento DPLL y sus adaptaciones

## 2.1. Procedimiento DPLL completo y corregido

Dada una formula lógica de la forma CNF: $C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \wedge ... \wedge C_n$ donde C son clausulas de la forma $l_1 \vee l_2 \vee ... \vee l_m$ y $l$ son literales y $\neg l$ denota la negación de l.

Una asignación M es un conjunto de literales tal que $l, \neg l \in M$ para no l. Una literal es *cierta* en M si $l \in L$, y es $falsa$ si $\neg l \in L$, y es indefinida en cualquier otro caso.

Diremos que M es total sobre L si ninguna literal de L es indefinido en M.

Una cláusula es *cierta* en M si todas sus literales lo son en M. Es falsa en M si todas sus literales son falsas en M,y no definido en otro caso.

Una formula F es cierta en M ó satisfecha por M y denotado por $M \models F$, si todas sus cláusulas son ciertas en M.

Definiremos cada procedimiento DPLL por medio de un conjunto de estados con a una relación binaria $\Longrightarrow$ sobre estos estados, llamado relación de transición, llamaremos transmisión de $S$ a $S'$ a $S \Longrightarrow S'$, además llamaremos derivación a transiciones de la forma $S_1 \Longrightarrow S_2$, $S_2 \Longrightarrow S_3$ y subderivación a cualquier subsecuencia se S.

Dado un estado S, una regla de transición define si hay una transición desde S a partir de esta regla al estado S', denotaremos esto un *pasodeaplicacin* de la regla.

un sistema de transiciones es un conjunto de reglas definidas sobre un conjunto de estados. Dado un sistema de transiciones R, definiremos $\Longrightarrow_R$ como la relación de transición. si no existen transiciones de S por $\Longrightarrow_R$. diremos que S es final con respecto a R.

Los estados en el sistema de transiciones de un procedimiento DPLL, son Failstate, o un par de la forma $M \| F$, donde F es la formula en forma normal conjuntiva (CNF siglas en inglés) y M una asignación.

M nunca contendrá una literal y su negada al mismo tiempo, marcaremos con un bit especial una literal decisional y sin el cuando no lo es, de tal forma que la denotaremos como $l^d$, denotaremos una secuencia vacía de literales por $\emptyset$.

Diremos que una cláusula esta en conflicto en el estado $M \| F, C$ si $M \models \neg C$ (para más información ver [10]).

Es posible utilizar un resolvedor DPLL para saber si existe alguna asignación $M$ que haga a la formula satisfacible o no, pero para ello definiremos el procedimiento DPLL completo y corregido.

El procedimiento DPLL completo y corregido consta de las siguientes reglas.

1. Unit propagation
2. Backjump
3. Decide
4. Fail
5. Learn
6. Forget

Definiremos un DPLL completo y corregido como un sistema de transiciones CI consistente de las 6 reglas anteriores, en este sistema las literales añadidas a M excepto Decide son marcadas como literales no desicionales.

En la implementación del procedimiento DPLL, se encontraron algunos casos extraordinarios los cuales mencionaremos más adelante, básicamente los casos y sus soluciones fueron aplicados a la regla backjump. Para comprender mejor esto presentamos la descripción de las reglas y posteriormente una explicación sobre su implementación.

UNITPROPAGATE

$$M \| F, C \vee l \Longrightarrow Ml \parallel F, C \vee l \, if \begin{cases} M \models \neg C \\ l \; es \; indefinida \; en \; M \end{cases} \tag{1}$$

Para poder hacer satisfacible una formula lógica en la forma CNF, es necesario hallar una asignación de verdad para sus literales dada una asignación $M$, en donde existen n literales

que van de $l_1, ..., l_n$, esta regla expande $M$ para lograr que una asignación sea verdadera.

DECIDE

$$M \| F, \Longrightarrow M l^d \parallel F if \begin{cases} o \neg\ l\ ocurre\ en\ una\ clausula\ de\ F \\ l\ es\ indefinida\ en\ M \end{cases} \tag{2}$$

Esta regla escoge una literal l de F, y la añade a M, esta literal es marcada como una literal decisional, la cual denota que si Ml no pude ser extendida al modelo F entonces la extensión $M \neg l$ debe ser considerada.

FAIL

$$M \| F, C \Longrightarrow FailState if \begin{cases} M \models \neg C \\ M\ contiene\ litareles\ no\ decicionales \end{cases} \tag{3}$$

Esta regla detecta una cláusula de conflicto C y produce un estado de falla cuando M contiene literales no decicionales.

BACKJUMP

$$M^d, N \parallel F, C \Longrightarrow M\ l' \parallel F, C if \begin{cases} M\ l^d\ N \models \neg C, y\ existe \\ alguna\ clausula\ C' \vee\ l'\ tal\ que: \\ F,\ C\ \models C' \vee\ l'\ y\ M \models \neg C', \\ l'\ no\ esta\ definida\ en\ M\ y \\ l'\ o \neg l'\ ocurre\ en\ F\ o\ en M l^d N \end{cases} \tag{4}$$

Esta regla es una regla que una vez hallado un conflicto, y no se puede aplicar la regla fail, hace el manejo de conflicto, para detectar a que punto de la asignación M se debe volver y regresa al mejor estado de M existente.

LEARN

$$M \parallel F \Longrightarrow M \parallel F, C if \begin{cases} todos\ los\ atomos\ de\ C\ ocurren\ en\ F \\ F \models C \end{cases} \tag{5}$$

Regla que aprende las cláusulas generadas en el manejo de conflicto por la regla backjump, y que evitan repetición de errores en la asignación.

FORGET

$$M \parallel F\ C \Longrightarrow M \parallel F, if \{\ F \models C \tag{6}$$

Esta regla olvida las cláusulas aprendidas, pero por medio de un análisis de las cláusulas menos usadas, con el fin de no saturar la memoria con cláusulas aprendidas.

## 2.2. Mejoras para el mejor funcionamiento de un procedimiento PPLL

Las mejoras realizadas al procedimiento DPLL que usamos son:

**esquema two − watched − literal**.

En el esquema llamado two-watched-literal, se escogen dos literales no-falsas (inicialmente no asignados), las cuales se observarán en cada cláusula. Una cláusula se procesa sólo cuando uno de sus two-watched-literales se asigna como falsa; tal asignación provoca la búsqueda para que otra literal no asignada reemplace la que se convierte falsa. La cláusula se convierte unitaria si el único literal que se encuentre es la otra literal observada, por lo cual debe ser asignada a verdad para satisfacer la cláusula[11].

**manejodecoflicto**

Una vez hallado un conflicto en una cláusula, y teniendo una asignación M, la forma de decidir a que nivel de decisión volver, y que literal debe ser marcada como $l'$, definiremos como Unique Implication Point (UIP), a la negación de la literal que actuará como $l'$, la cual es hallada por medio del uso la regla de resolución sobre la cláusula de conflicto, y cada una de las cláusulas que propagan a las literales en dicha cláusula, hasta hallar una sola literal del nivel de decisión actual.

Para entender esto mejor supongamos que tenemos un conflicto en la cláusula $5 \bigvee 7 \bigvee \neg 7$, y además sabemos que:

| Literal | Cláusula propagada |
|---|---|
| 3 | $1 \bigvee \neg 2 \bigvee 3$ |
| 2 | $5 \bigvee 7 \bigvee 1 \bigvee \neg 2$ |
| 1 | $\neg 4 \bigvee 5 \bigvee 7 \bigvee 1$ |
| 4 | $5 \bigvee 7 \bigwedge \neg 4$ |

M es de la forma:...6 ...¬7 ..,9 ¬8 ¬5 4 ¬1 2 ¬3

Ultima literal decicional: 9

Aplicando la regla de resolución para encontrar el UIP tendrá como resultado:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{5 \bigvee 7 \bigvee \neg 3 \qquad 1 \bigvee \neg 2 \bigvee 3}{5 \bigvee 7 \bigvee 1 \bigvee \neg 2}}{\neg 4 \bigvee 5 \bigvee 2} \qquad \dfrac{\neg 4 \bigvee 5 \bigvee 7 \bigvee 1}{\neg 4 \bigvee 1}}{\neg 6 \bigvee 8 \bigvee 4} \qquad \dfrac{4 \bigvee 7 \bigwedge \neg 4}{\neg 6 \bigvee 8 \bigwedge 7 \bigvee \neg 5}}{8 \bigvee 7 \bigvee \neg 5}}{8 \bigvee 7 \bigvee \neg 6}$$

En donde la la cláusula donde hallamos el UIP es: $8 \bigvee 7 \bigvee \neg 6$, para este ejemplo el UIP es la literal 8.

Los casos a consideras ahora son: cuando todas la literales son negativas, la regla backjump simula a la siguiente regla:

BACKTRACK

$$M^d, N \parallel F, C \Longrightarrow M \neg l \parallel F, C \; if \begin{cases} M \; l^d \; N \models \neg C \\ N \; contiene \; literales \; no \; decicionales \end{cases} \tag{7}$$

En la cual se limita a negar el valor de la ultima literal decisional y marcarla como $l'$. borrando todas las literales propagadas en el presente nivel de decisión.

# 3. Implementación

A continuación se presenta el algoritmo del programa

```
Data: Formula F
Result: Asignación M valida o estado de falla
initialization M a vacio;
decide;
while no estado de falla o l no asignadas do
    switch verifica do
        case hay literales por propagar
            unitpropagation;
        end
        case M ya no pude ser extendida
            decide;
        end
        case existe un conflicto
            if Fail no se aplica then
                backjump
        end
        otherwise
            reinicia el programa;
        end
    end
end
```

**Algorithm 1**: resolvedor SAT usando DPLL

Las funciones principales del algoritmo son las siguientes:

1. Unitpropagation
2. Backjump
3. Decide
4. Fail
5. Verifica

Presentamos a continuación una descripción de cada una de ellas y algunas de sus subfunciones.

**Decide**
En esta función, se elige la literal decisional que se ha de elegir para continuar la expansión de M, cuando no es posible expandir la asignación M a partir de alguna otra regla.

La heurística usada para la selección de una literal $l$, es que al principio del programa se genera una lista de todas las literales existentes en F, y se crea un listado de por ordenado de mayor a manor por orden de aparición en M.

Una vez obtenida esta lista la función decide, hace un recorrido por ella y selecciona la literal que tenga un índice de incidencia superior y que además no este asignada en M.

**Fail**
Esta función verifica si se ha alcanzado el estado de falla (fail state). si es así termina el programa.

**Unitpropagation**
Esta función propaga literales en M usando al heurística de two-watched literal.

**Verifica**
Esta función verifica si existe un conflicto de M sobre alguna cláusula perteneciente, si hay M puede ser extendido por que existe literales de propagación pendientes. Además contiene actualizado el listado de que cláusulas aprendidas aparecen el casos de conflicto, y el listado de las literales que también aparecen el casos de conflicto, de este último para evitar números muy grandes cada 200 ejecuciones de la función divide todos los contadores por 25.

**Backjump**
La función backjump una vez alcanzado un conflicto, y no pudiéndose aplicar la regla $fail$, genera el tratamiento de conflicto para determinar a que punto debe volver la asignación M.

**Data**: conflicto en F
**Result**: M regresado a un punto especifico con un $l'$ añadido
initialization M a vació;
**if** *lit-neg* **then**
  regresa-l;
**else**
  calculus;
  **if** *existe-espacio* **then**
    learn ;
  **else**
    forget;
    learn;
regresa;

**Algorithm 2**: función Backjump

En donde:

**Learn** es la función que aprende las cláusulas generadas por por la función calculus, y evitar conflictos posteriores.
**Forget** Esta función, borra algunas de las cláusulas aprendidas por la función learn, mirando el índice de cláusulas menos usadas por la función verifica y eliminándolas.
**regresa-l** Esta función, regresa al literal decisional penúltima y borra las literales propagadas en el presente nivel de decisión agregando la última literal decisional como $l'$.
**calculus**. Esta función calcula el UIP, sobre el cual se debo volver, mediante la aplicación de la regla de resolución.
**existe-espacio**. Esta función verifica si existe espacio para aprender, si es así retorna TRUE, en caso contrario false.

# 4. Resultados

Para poder comparar el rendimiento de nuestro programa, es necesario adaptarse al estándar establecido por los benchmarks existentes, provenientes de diversas áreas, el cual es:

Archivo.cnf
línea 1 p cnf 5525 96480
línea 2 5318 0
línea n ....

En la primera línea del archivo tenemos las indicaciones que las cláusulas están en forma normal conjuntiva, seguido del número de literales que aparecen en F, y finamente el numero de cláusulas que tiene F.

A partir de la segunda línea aparecen las cláusulas, en donde el operador $\bigvee$ es sustituido por un espacio. y la separación entre cláusulas es un salto de línea.

El ordenador en donde se ejecuto el set de pruebas es una Toshiba Satellite m50, con un procesador Intel Centrino a 1.8 GHz, 2Mb de cache, 1Gb de memoria RAM y Linux Debian 3.1. El tiempo de los resultados se presenta en segundos usados para decidir la satisfabilidad o no de cada problema.

# 5. Conclusiones

Los problemas SAT, tiene una gran gama de aplicaciones tanto en el mundo científico como en el mundo industrial, el poder hallar soluciones que permitan saber cuando un problema es satisfacible o no en tiempos razonables resulta ser de gran importancia.

| Archivo | Estado | DPLL | ZCHAFF | SIEGE | MSAT | MSAT mnimize | MSAT nrand mnimize | MSAT mnimize a little | BCLT |
|---|---|---|---|---|---|---|---|---|---|
| 4pipe_1_ooo | u | 4.98 Ă | 12.40 | 4.29 | 56.80 | 8.97 | 14.00 | 3.82 | 18.34 |
| 4pipe_2_ooo | u | 26.15 Ă | 20.20 | 6.15 | 23.60 | 11.60 | 15.00 | 11.70 | 16.58 |
| 4pipe_3_ooo | u | 14.17 Ă | 17.10 | 14.10 | 218.00 | 42.60 | 44.00 | 39.90 | 17.60 |
| 4pipe_4_ooo | u | 32.45 Ă | 19.40 | 9.08 | 115.00 | 14.50 | 163.00 | 60.20 | 19.24 |
| 4pipe | u | 5.00 Ă | 9.83 | 4.90 | 337.00 | 453.00 | 219.00 | 359.00 | 44.14 |

Cuadro 1: Comparación de tiempos, de nuestro programa DPLL comparado con otros programas existentes.

EL uso de resolvedores SAT basado en procedimiento DPLL son un gran avance en el tratamiento de este tipo de problemas, pues permiten resolver problemas cada vez más grandes. Las mejoras en el manejo de conflictos así como de como las heurística de como decidir la forma en que se propagan la literales en M, resultan ser de gran ayuda para aumentar el rendimiento del programa. Concluimos entonces como resultado final que la implementación de del resolvedor SAT basado en el procedimiento DPLL completo y corregido dio como resultado en algunos casos, la majora en los tiempos necesarios para determinar la satisfabilidad de una formula lógica, y encontrar el modelo M que cumple dicha condición ó en caso contrario para mostrar que no se puede alcanzar el estado anterior.

# 6. Referencias


[1]M. Motoki, On the Maximum Satisfiability of Random 3-CNF Formulae, Research Report C-141, Dept. of Math. and Computing Sciences, Tokyo Inst. of Tech. (2000).

[2] Garey M., and Johnson D. Computers and Intractability: a Guide to the Theory of NP-completeness. Freeman, San Francisco, California, 1979.

[3] Inês Lynce, João Marques Silva. Efficient Haplotype Inference with Boolean Satisfiability.American Association for Artificial Intelligence.2006.

[4]Matthew W. Moskewicz, Conor F. Madigan,Ying Zhao, Lintao Zhang, Sharad Malik. Chaff: Engineering an Efficient SAT Solver. Proceedings of the 38th Design Automation Conference (DAC'01).

[5]Evgueni Goldberg, Yakov Novikov. BerkMin: a Fast and Robust Sat-Solver (2002).

[6]Miroslav N. Velev, Randal E. Bryant. Effective Use of Boolean Satisfiability Procedures in the Formal Verification of Superscalar and VLIW Microprocessors (2001), Proceedings of the 38th Conference on Design Automation Conference 2001.

[7]Henry Kautz and Bart Selman. Unifying SAT-based and Graph-based Planning. Proc. IJCAI-99, Stockholm, 1999.

[8]Evgueni Goldberg, Yakov Novikov. BerkMin: a Fast and Robust Sat-Solver (2002)

[9]J. P. Marques-Silva and K. A. Sakallah, "GRASP: a search algorithm for propositional satisfiability,"IEEE Trans. on Computers, vol. 48, no. 5, pp. 506–521, 1999.

[10]Robert Nieuwenhuis, Albert Oliveras, Cesare Tinelli. Abstract DPLL and Abstract DPLL Modulo Theories. LPAR 2004: 36-50

[11]Hossein M. Sheini, Karem A. Sakallah. Pueblo:A Modern Pseudo-Boolean SAT Solver. Design, Automation, and Test in Europe archive Proceedings of the conference on Design, Automation and Test in Europe - Volume 2.