Conflict-Driven Clause Learning (CDCL) SAT Solver: Algoritmo Explicado con una Implementación en Python y Ejemplos

1. Introducción a la Satisfacibilidad Booleana (SAT)

El Problema de Satisfacibilidad Booleana (SAT) se erige como un pilar fundamental en los campos de la lógica y la informática. Plantea una pregunta básica: dada una fórmula booleana, ¿existe una asignación de valores de verdad (VERDADERO o FALSO) a sus variables tal que la fórmula completa se evalúe como VERDADERO? Si existe dicha asignación, la fórmula se considera "satisfacible"; de lo contrario, es ïnsatisfacible".[1]

La profunda importancia de SAT proviene de su clasificación como el primer problema demostrado como NP-completo. Este descubrimiento pivotal fue realizado por Stephen Cook en 1971, con trabajo independiente de Leonid Levin en 1973, lo que condujo al teorema de Cook-Levin.[2] Esta equivalencia teórica significa que cualquier método eficiente para resolver SAT podría, en principio, adaptarse para resolver eficientemente cualquier otro problema NP.

A pesar de su estatus NP-completo, que sugiere que no se conoce ningún algoritmo de tiempo polinomial que exista para SAT en el peor caso[3], se han realizado avances prácticos significativos en la resolución de SAT durante las últimas décadas.[4] El desarrollo de algoritmos y heurísticas sofisticados ha permitido a los solucionadores abordar problemas SAT altamente complejos que alguna vez se consideraron intratables.[5] Este progreso subraya un aspecto crucial del diseño de algoritmos para problemas NP-completos: si bien los límites teóricos del peor caso persisten, los esfuerzos prácticos se centran en desarrollar heurísticas y estructuras de datos altamente eficientes que funcionan excepcionalmente bien en instancias típicas o estructuradas del mundo real. Este enfoque navega efectivamente por el vasto espacio de búsqueda exponencial, transformando la resolución de SAT de un desafío puramente teórico en una poderosa herramienta práctica.

Existen varias variantes especializadas de SAT, cada una con sus propios desafíos computacionales únicos. Ejemplos incluyen 3-SAT (donde cada cláusula tiene como máximo tres literales, también NP-completo), MAJ-SAT (preguntando si al menos la mitad de todas las asignaciones satisfacen la fórmula), #SAT (contar asignaciones satisfactorias, que es #P-completo), UNIQUE SAT (determinar si existe exactamente una asignación), MAX-SAT (encontrar el número máximo de cláusulas satisfacibles, que es NP-duro), y WMSAT (satisfacibilidad mínima ponderada para fórmulas monótonas).[6] La diversidad de estas variantes destaca la versatilidad del problema y su amplia aplicabilidad en diferentes dominios computacionales.

[1] Por ejemplo, la fórmula "a Y NO b" es satisfacible, ya que establecer $a = VERDADERO$ y $b = FALSO$ hace que la expresión sea verdadera.

[2] La NP-completitud de SAT implica que todos los problemas dentro de la clase de complejidad NP—una vasta gama de desafíos de decisión y optimización—son a lo sumo tan difíciles de resolver como SAT.

[3] Cook, S. A. (1971). The complexity of theorem-proving procedures. Proceedings of the third annual ACM symposium on Theory of computing.

[4] Marques-Silva, J., & Sakallah, K. A. (1999). GRASP: A search algorithm for propositional satisfiability. IEEE Transactions on Computers.

[5] Moskewicz, M. W., et al. (2001). Chaff: Engineering an efficient SAT solver. Proceedings of the 38th annual Design Automation Conference.

[6] 1

1.1. Forma Normal Conjuntiva (FNC)

Para el procesamiento algorítmico, las fórmulas booleanas generalmente se convierten en una representación estandarizada conocida como Forma Normal Conjuntiva (FNC). Una fórmula en FNC se estructura como una conjunción (AND lógico) de una o más cláusulas, donde cada cláusula es una disyunción (OR lógico) de uno o más literales.[7] Un literal es una variable proposicional (ej. x_1) o su negación (ej. $\neg x_1$).[8] Por ejemplo, la fórmula:

$$(\neg x_1 \vee x_3) \wedge (x_2 \vee x_3) \wedge (\neg x_2 \vee x_3)$$

está en FNC.[9] Cualquier fórmula booleana puede transformarse en una fórmula FNC equisatisfacible, meaning la fórmula original es satisfacible si y sólo si su equivalente FNC es satisfacible.[10]

Esta estandarización a FNC ofrece una ventaja significativa: permite que todo el campo de la resolución SAT se concentre en una única representación simplificada del problema. Los solucionadores no necesitan analizar y manejar expresiones booleanas arbitrarias con jerarquías de conectivos complejas. En cambio, operan sobre la estructura directa de las cláusulas. Esta simplificación es fundamental para desarrollar componentes algorítmicos altamente especializados y optimizados, como la propagación de unidades y el análisis de conflictos. La adopción de FNC como formato de entrada universal (ej. DIMACS CNF) ha creado así una interfaz común, haciendo factible desarrollar solucionadores SAT de propósito general que son tanto eficientes como ampliamente aplicables a través de un amplio espectro de problemas que pueden codificarse en SAT.[11]

Dentro de este marco, una cláusula se considera **satisfecha** si al menos uno de sus literales se evalúa como VERDADERO. Por el contrario, una cláusula está **insatisfecha** si todos sus literales están asignados a FALSO. Una fórmula completa se satisface sólo si todas sus cláusulas constituyentes están satisfechas.[12]

1.2. Breve Contexto Histórico: De DPLL a CDCL

Los orígenes de la resolución moderna de SAT se encuentran en el algoritmo Davis-Putnam-Logemann-Loveland (DPLL), un procedimiento de búsqueda completo introducido a principios de la década de 1960. DPLL emplea un enfoque recursivo de retroceso (backtracking) en profundidad para explorar el espacio de búsqueda de las posibles asignaciones de variables.[13] Si bien es fundamental, la eficiencia de DPLL a menudo se veía limitada por su mecanismo de retroceso cronológico, que simplemente deshacía la decisión más reciente al encontrar un conflicto, potentially reexplorando ramas improductivas del árbol de búsqueda.[14]

El paradigma Conflict-Driven Clause Learning (CDCL, Aprendizaje de Cláusulas Impulsado por Conflictos) surgió como una mejora revolucionaria de DPLL, transformando fundamentalmente el panorama de la resolución SAT.[15] Los solucionadores CDCL conservan la búsqueda central con retroceso, pero introducen mecanismos potentes para aprender de los fallos y guiar la búsqueda de manera más inteligente. Las distinciones más notables incluyen el propio aprendizaje de cláusulas impulsado por conflictos, el retroceso no cronológico (backjumping) y heurísticas sofisticadas para la toma de decisiones. Estas innovaciones han permitido a los solucionadores CDCL superar las limitaciones de los enfoques DPLL más simples, conduciendo al rendimiento notable observado en los solucionadores SAT contemporáneos.[16]

2. El algoritmo CDCL: una visión general

El algoritmo CDCL representa un proceso iterativo sofisticado diseñado para determinar la satisfacibilidad de una fórmula booleana. A diferencia de los métodos de retroceso más simples, CDCL se adapta dinámi-

[7] Biere, A., et al. (2009). Handbook of satisfiability. IOS press.
[8] Zhang, L., & Malik, S. (2002). The quest for efficient boolean satisfiability solvers.
[9] Tseitin, G. S. (1968). On the complexity of derivation in propositional calculus.
[10] Plaisted, D. A., & Greenbaum, S. (1986). A structure-preserving clause form translation.
[11] 5
[12] Karp, R. M. (1972). Reducibility among combinatorial problems.
[13] 3
[14] 3
[15] 7
[16] 3

camente en su estrategia de búsqueda aprendiendo de las contradicciones encontradas. Este refinamiento continuo es central para su eficiencia y capacidad para resolver problemas complejos.

2.1. Bucle iterativo central: Decidir, Propagar, Conflicto, Aprender, Retroceder

La operación de un solucionador CDCL se orquesta a través de un bucle iterativo implacable que persiste hasta que la satisfacibilidad de la fórmula de entrada se determina definitivamente. Este bucle integra varios pasos cruciales, cada uno contribuyendo a la capacidad del solucionador para navegar efectivamente el vasto espacio de búsqueda.[17]

- **Inicialización:** El proceso comienza con la configuración de las estructuras de datos internas. Esto incluye inicializar el mecanismo de literales observados (watched literals), que es vital para la propagación eficiente de restricciones.[18]
- **Verificación de reinicio:** Al comienzo de cada iteración principal, el solucionador evalúa si un "reinicio" es necesario. Si el número de conflictos detectados alcanza un umbral predefinido, el solucionador realiza un reinicio. Esto implica restablecer la asignación parcial actual retrocediendo al nivel de decisión 0, borrando efectivamente el contexto de búsqueda inmediato. Crucialmente, todas las cláusulas aprendidas antes del reinicio se conservan. Esta estrategia dinámica permite al solucionador escapar de regiones improductivas del espacio de búsqueda, evitando que se quede atascado en óptimos locales.[19]
- **Propagación de unidades (UP):** Después de cualquier reinicio o decisión, se realiza la propagación de unidades. Este es un paso determinista y crítico donde el solucionador identifica y propaga las consecuencias lógicas de las asignaciones actuales de variables. Si una cláusula contiene solo un literal no asignado, y todos los demás literales en esa cláusula son actualmente falsos, entonces el literal no asignado debe asignarse como verdadero para satisfacer la cláusula. Este proceso se aplica iterativamente hasta que no se puedan encontrar más cláusulas unitarias o surja un conflicto.[20]
- **Detección de conflictos:** Después de la propagación de unidades, el solucionador verifica si la asignación parcial actual ha llevado a una contradicción, meaning una cláusula se ha vuelto completamente falsa (todos sus literales están asignados como falsos).[21]
- **Manejo de conflictos:** Si se detecta un conflicto:
  - El contador de conflictos se incrementa.
  - Si el conflicto ocurre en el nivel de decisión 0 (meaning no se han tomado decisiones de ramificación, o todas las decisiones se han deshecho, y el conflicto surge de la fórmula inicial o de cláusulas aprendidas), la fórmula es inherentemente insatisfacible (UNSAT), y el solucionador termina, reportando falso.[22]
  - De lo contrario (si el conflicto está en un nivel de decisión mayor que 0), el solucionador inicia un procedimiento de análisis de conflictos. Esta función examina el gráfico de implicación (o el rastro de asignaciones) para entender la causa raíz del conflicto. Luego genera una nueva cláusula aprendida aplicando repetidamente pasos de resolución, típicamente hasta que se alcanza un Punto de Implicación Único (Unique Implication Point, UIP).[23]
  - La nueva cláusula aprendida se agrega a la base de datos de cláusulas. Esta cláusula captura la razón del conflicto y sirve para evitar que el solucionador cometa el mismo error again en futuras búsquedas.[24]

[17] 3
[18] 3
[19] 3
[20] 3
[21] 3
[22] 3
[23] 3
[24] 3

- El análisis de conflictos también determina un nivel de retroceso (backjump level), que es el nivel de decisión que realmente causó el conflicto. El solucionador luego realiza retroceso no cronológico a este nivel calculado, saltando efectivamente sobre decisiones irrelevantes y podando grandes porciones del espacio de búsqueda.[25]
- Finalmente, las puntuaciones de actividad de las variables involucradas en la cláusula aprendida se "incrementan" (aumentan), y todas las actividades de las variables se reducen. Este mecanismo, parte de la heurística VSIDS, prioriza variables que han contribuido recientemente a conflictos, guiando la toma de decisiones futuras.[26]

- **Toma de decisiones:** Si la propagación de unidades no conduce a un conflicto y todavía hay variables no asignadas, el solucionador debe tomar una nueva decisión. Selecciona una variable no asignada y le asigna un valor de verdad (VERDADERO o FALSO). Esta elección está típicamente guiada por una heurística de ramificación, como VSIDS, que aims seleccionar variables que probablemente lleven rápidamente a una solución.[27]

- **Terminación:** El bucle continúa hasta que se encuentre una asignación satisfactoria (todas las variables están asignadas sin conflicto, indicando SAT), o se detecte un conflicto en el nivel de decisión 0 (indicando UNSAT).[28]

La tabla below resume el flujo iterativo central del algoritmo CDCL, destacando el propósito y los componentes clave involucrados en cada paso.

Cuadro 1: Resumen del flujo del algoritmo CDCL

Paso	**Descripción**	**Propósito**	**Componentes clave**
Inicialización	Configura estructuras de datos internas y literales observados.	Prepara el solucionador para operación eficiente.	Literales observados
Verificación de reinicio	Evalúa si el número de conflictos justifica restablecer la búsqueda.	Escapa de áreas improductivas.	Umbral de reinicio
Propagación de unidades	Identifica y propaga asignaciones forzadas de variables.	Extiende asignación parcial; detecta conflictos.	Literales observados, Cláusulas
Detección de conflictos	Verifica si alguna cláusula es falsificada.	Identifica contradicciones.	Cláusulas, Asignaciones
Manejo de conflictos	Analiza conflictos, aprende nuevas cláusulas y retrocede.	Comprende causas de conflictos.	Análisis de conflictos, 1-UIP
Toma de decisiones	Selecciona variable no asignada y le asigna valor.	Extiende asignación cuando no hay propagaciones.	Heurística VSIDS
Terminación	Verifica si todas variables asignadas (SAT) o conflicto en DL0 (UNSAT).	Determina satisfacibilidad final.	Asignación completa

[25] 3
[26] 3
[27] 3
[28] 3

2.2. Diferencias clave y ventajas sobre DPLL tradicional

Los solucionadores CDCL representan una evolución significativa del algoritmo DPLL tradicional, principalmente al transformar una búsqueda simple de retroceso en profundidad en una exploración dinámica impulsada por el conocimiento. Este cambio de paradigma explica las notables mejoras de rendimiento observadas en los solucionadores SAT modernos.[29]

Una de las distinciones más notables es la implementación del retroceso no cronológico (backjumping) en CDCL. A diferencia de DPLL, que retrocede estrictamente al nivel de decisión inmediatamente anterior ante un conflicto, CDCL analiza el conflicto para determinar la causa real y salta directamente al nivel de decisión relevante para resolver ese conflicto, saltando potencialmente muchas decisiones intermedias irrelevantes.[30] Esto permite al solucionador podar vastas porciones futiles del espacio de búsqueda que DPLL exploraría exhaustivamente.

Central al poder de CDCL es el aprendizaje de cláusulas, el componente epónimo del algoritmo. Cuando ocurre un conflicto, el solucionador no simplemente retrocede; analiza las razones subyacentes de la contradicción y sintetiza una nueva cláusula, conocida como çláusula aprendida.º "lema".[31] Esta cláusula aprendida es una consecuencia lógica de la fórmula original y se agrega a la base de datos de cláusulas. Su propósito es "prohibir"la combinación específica de asignaciones que llevó al conflicto para que no ocurra again, evitando efectivamente que el solucionador re-explore partes similares del árbol de búsqueda.[32] Este mecanismo transforma la búsqueda de un recorrido pasivo en un proceso activo de acumulación de conocimiento.

El algoritmo CDCL puede verse como un sistema de aprendizaje iterativo. Refina continuamente su comprensión del problema y su estrategia de búsqueda basándose en fallos pasados. Este proceso es análogo a los sistemas de control de aprendizaje iterativo (ILC) en ingeniería, donde la información de .[en]sayosçompletados se utiliza para mejorar el rendimiento en iteraciones posteriores.[33] La naturaleza ïmpulsada por conflictos"garantiza que este aprendizaje siempre se centre en superar obstáculos, haciendo que el proceso sea altamente eficiente y adaptable a la estructura específica de una instancia SAT dada.

Además, los solucionadores CDCL emplean heurísticas inteligentes para la toma de decisiones, como Variable State Independent Decaying Sum (VSIDS), para guiar la selección de la siguiente variable sobre la cual ramificar. Estas heurísticas priorizan variables que han estado frecuentemente involucradas en conflictos recientes, dirigiendo la búsqueda hacia las partes más restringidas o "problemáticas"de la fórmula.[34]

Otra mejora crucial es el uso estratégico de reinicios. Periódicamente, el solucionador puede restablecer su asignación parcial actual y retroceder al nivel de decisión 0, mientras preserva todas las cláusulas aprendidas.[35] Esto permite al solucionador escapar de óptimos locales y explorar nuevas regiones del espacio de búsqueda, lo que es particularmente beneficioso para instancias difíciles.

Un aspecto sutil pero crítico del diseño de CDCL es la interacción entre aprender y olvidar cláusulas. Si bien el aprendizaje de cláusulas es esencial para podar el espacio de búsqueda, una base de datos en constante crecimiento de cláusulas aprendidas puede paradójicamente degradar el rendimiento. El exceso de cláusulas puede ralentizar la propagación de unidades y consumir memoria significativa.[36] La investigación ha demostrado que el aprendizaje de cláusulas sin un mecanismo para descartar cláusulas menos útiles puede deteriorar dramáticamente el proceso de solución.[37] Esto destaca la necesidad de una gestión dinámica de memoria y relevancia del conocimiento dentro del solucionador, donde las cláusulas más antiguas o menos efectivas se eliminan periódicamente para mantener la eficiencia. Este equilibrio garantiza que la base de conocimiento del solucionador se mantenga compacta y relevante, optimizando su capacidad para progresar.

[29] 11
[30] 3
[31] 3
[32] 11
[33] 14
[34] 3
[35] 3
[36] 11
[37] 11

3. Componentes centrales de un solucionador CDCL

La eficiencia y robustez de los solucionadores CDCL se atribuyen a la interacción sofisticada de varios componentes centrales. Cada componente aborda un desafío específico al navegar el vasto espacio de búsqueda de problemas SAT.

3.1. Toma de decisiones y heurísticas de ramificación (VSIDS)

Cuando la propagación de unidades ya no puede deducir más asignaciones, el solucionador CDCL debe hacer una elección arbitraria: "decideün valor de verdad para una variable no asignada. Esto se conoce como un .estado de decisión".[38] La efectividad del solucionador depende en gran medida de qué variable se elige para este paso de ramificación. Los solucionadores CDCL modernos dependen predominantemente de la heurística *Variable State Independent Decaying Sum* (VSIDS) para tomar estas decisiones críticas.[39]

El mecanismo VSIDS opera asignando una "puntuación de actividad"(un número de punto flotante) a cada variable en la fórmula booleana, típicamente inicializada en cero.[40] Su naturaleza dinámica se rige por dos operaciones clave:

- **Incremento aditivo (Bumping):** Cuando ocurre un conflicto, las puntualiza las actuaciones de actividad de las variables involucradas en ese conflicto—específicamente, aquellos literales dentro de la cláusula aprendida o aquellos resueltos durante el análisis de conflictos—son ïncrementadas.º aumentadas aditivamente, usualmente por una cantidad fija como 1.[41] Este aumento inmediato resalta su participación reciente en asignaciones problemáticas.
- **Decrecimiento multiplicativo (Decay):** A intervalos regulares durante la ejecución del solucionador, las puntuaciones de actividad de todas las variables se multiplican por un factor de decrecimiento constante, α, donde $0 < \alpha < 1$.[42] Este decrecimiento reduce gradualmente la influencia de conflictos más antiguos, asegurando que la heurística se mantenga enfocada en las partes más recientes y relevantes del problema.

El solucionador luego selecciona la variable no asignada con la puntuación de actividad actual más alta para ramificar.[43] La efectividad de VSIDS surge de su comportamiento como una media móvil exponencial (EMA). El decrecimiento multiplicativo hace que priorice variables que aparecen persistentemente en conflictos (representando una "señal") sobre aquellas que aparecen intermitentemente (consideradas ruido").[44] Este enfoque dinámico permite al solucionador concentrar sus esfuerzos de búsqueda en las variables más .ªctivas.º restringidas", que a menudo son las que impulsan las dificultades actuales para satisfacer la fórmula. VSIDS fue un gran avance cuando se introdujo por primera vez como parte del solucionador SAT Chaff, y sus principios continúan sustentando muchas heurísticas de ramificación de alto rendimiento en la actualidad.[45] Este enfoque adaptativo permite al solucionador concentrar el esfuerzo computacional donde es más probable que produzca progreso, en lugar de tomar decisiones arbitrarias.

3.2. Propagación de unidades (Propagación de restricciones booleanas - BCP)

La propagación de unidades, también conocida como Propagación de Restricciones Booleanas (BCP), es un proceso fundamental y determinista dentro de los solucionadores CDCL.[46] Se aplica después de cada decisión o implicación para deducir asignaciones forzadas. La regla es simple: si una cláusula contiene solo un literal no asignado, y todos los demás literales en esa cláusula están actualmente asignados a FALSO, entonces el único literal no asignado debe asignarse a VERDADERO para satisfacer la cláusula.[47] Este

[38] 3
[39] 3
[40] 13
[41] 3
[42] 13
[43] 13
[44] 13
[45] 13
[46] 7
[47] Silva, J. P. M., & Sakallah, K. A. (1996). GRASP-A new search algorithm for satisfiability.

proceso continúa iterativamente hasta que no se puedan encontrar más cláusulas unitarias o se detecte una contradicción.[48]

Para implementar eficientemente la propagación de unidades, especialmente en solucionadores que manejan millones de cláusulas (incluyendo un número creciente de cláusulas aprendidas), el .[es]quema de dos literales observados"(two-literal watching scheme) es ampliamente adoptado.[49] Esta optimización reduce drásticamente la carga computacional en comparación con verificar ingenuamente cada cláusula después de cada asignación.

- Para cada cláusula, el solucionador designa dos literales para ser "observados". Estos literales observados se eligen de tal manera que no estén actualmente asignados a FALSO.[50]
- El principio clave es que una cláusula solo necesita ser inspeccionada cuando uno de sus dos literales observados es asignado a FALSO. Si esto ocurre, el solucionador intenta encontrar otro literal no FALSO, no observado dentro de la misma cláusula para reemplazar el literal observado recién falsificado.[51]
- Si no se puede encontrar tal reemplazo, una de dos condiciones debe cumplirse:
  - La cláusula se ha convertido en una cláusula unitaria: el otro literal observado no está asignado, y todos los demás literales (incluyendo el que acaba de ser falsificado) son FALSO. El literal observado no asignado es entonces implicado como VERDADERO.[52]
  - La cláusula se ha convertido en una cláusula de conflicto: ambos literales observados y todos los demás literales en la cláusula son ahora FALSO. Esto significa una contradicción.[53]

Los beneficios del esquema de dos literales observados son sustanciales. Reduce significativamente las verificaciones innecesarias, ya que las cláusulas solo se visitan cuando están cerca de convertirse en cláusulas unitarias o de conflicto.[54] Además, permite la desasignación en tiempo constante durante el retroceso, ya que no es necesario modificar los literales observados en la base de datos de cláusulas cuando se deshace una asignación. Esta optimización también reduce el número total de accesos a memoria, que a menudo es un cuello de botella principal para las implementaciones de solucionadores SAT debido a las altas tasas de fallos de caché de datos.[55] Este diseño inteligente de estructura de datos sirve como una çaché.[ef]ectiva para la inferencia lógica, asegurando que la operación más frecuente—verificar cláusulas unitarias—se realice con máxima eficiencia.

3.3. Gráfico de implicaciones y gestión del trail

Si bien los solucionadores SAT completos normalmente no construyen una estructura de datos de gráfico explícita para cada asignación, las relaciones entre las asignaciones de variables se representan conceptualmente y se rastrean implícitamente a través de un "trail.º "pila de asignaciones".[56] Este trail es un registro secuencial de todos los literales asignados con un valor de verdad durante la ruta de búsqueda actual.

Cada entrada en el trail está anotada con información crucial[57]:

- El literal que fue asignado (ej. x_1 o $\neg x_2$).
- El nivel de decisión en el que ocurrió la asignación. Los niveles de decisión aumentan con cada nueva decisión arbitraria tomada por el solucionador.
- La cláusula antecedente que forzó esta asignación, si fue una implicación (ej. C_1). Para variables de decisión (aquellas elegidas arbitrariamente), el antecedente típicamente se marca como vacío o NIL.[58]

[48] 10
[49] 3
[50] 3
[51] 3
[52] 3
[53] 3
[54] 3
[55] 3
[56] 3
[57] 3
[58] 3

- Un indicador de si la asignación fue una decisión o una propagación.[59]

El nivel de decisión asociado con una variable refleja su profundidad dentro del árbol de búsqueda actual. Las variables de decisión están en el nivel de decisión actual, mientras que las variables implicadas heredan el nivel de decisión más alto entre los literales en su cláusula antecedente, o nivel 0 si son implicadas por una cláusula unitaria en ese nivel.[60]

El gráfico de implicaciones es un gráfico acíclico dirigido conceptual que ayuda a visualizar estas dependencias.[61] En este modelo conceptual, los vértices representan literales asignados, y las aristas dirigidas indican implicaciones derivadas de cláusulas. Por ejemplo, si la cláusula $(\neg A \vee B)$ implica B cuando A es verdadero, una arista iría de A a B. Cuando ocurre un conflicto, se agrega un "vértice de conflicto.[es]pecial al gráfico, con aristas que llegan a él desde las negaciones de los literales en la cláusula falsificada. Este gráfico permite rastrear la cadena de asignaciones e implicaciones que llevaron a una contradicción, lo cual es crucial para el análisis de conflictos. El trail, como representación lineal, captura eficientemente estas relaciones, permitiendo un recorrido rápido durante el análisis de conflictos sin la sobrecarga de estructuras de datos de gráficos explícitas.

3.4. Análisis de conflictos y aprendizaje de cláusulas

Cuando la asignación parcial actual conduce a una contradicción—es decir, una cláusula se vuelve completamente falsificada—el solucionador CDCL entra en su fase crítica de análisis de conflictos.[62] El objetivo es comprender la causa raíz del conflicto y generar nuevo conocimiento para prevenir su recurrencia.

El proceso de análisis de conflictos implica recorrer el gráfico de implicaciones (o el trail) hacia atrás desde la cláusula conflictiva.[63] Este recorrido identifica un çonjunto de razones"de literales que, si todos fueran verdaderos, conducirían inevitablemente al conflicto observado.[64] El mecanismo central para derivar este conjunto de razones es el principio de resolución. Comenzando con la cláusula conflictiva, el solucionador la resuelve repetidamente con las cláusulas antecedentes de variables del nivel de decisión actual. Este proceso continúa hasta que se alcanza un Punto de Implicación Único (Unique Implication Point, UIP).[65]

Los Puntos de Implicación Únicos (UIPs) son centrales para un aprendizaje efectivo de cláusulas. Un UIP es un literal en el gráfico de implicaciones tal que todos los caminos desde el literal de decisión más reciente (en el nivel de decisión actual) al vértice de conflicto deben pasar a través de él.[66] Si bien pueden existir múltiples UIPs, los solucionadores CDCL modernos casi universalmente emplean el esquema 1-UIP. Este esquema se enfoca en identificar el primer UIP encontrado al recorrer hacia atrás desde el conflicto, que es el UIP más cercano al vértice de conflicto.[67]

La cláusula aprendida derivada de un corte 1-UIP posee una propiedad crucial: contiene exactamente un literal del nivel de decisión más reciente antes del retroceso no cronológico.[68] Tal cláusula se denomina çláusula assertiva". Los beneficios del esquema 1-UIP son significativos: produce cláusulas assertivas que guían efectivamente el retroceso, y genera cláusulas con la Distancia de Bloque Literal (Literal Block Distance, LBD) mínima entre todas las cláusulas assertivas posibles. LBD, que mide el número de niveles de decisión distintos en una cláusula, se correlaciona con la fuerza semántica, haciendo que estas cláusulas sean altamente efectivas para podar el espacio de búsqueda.[69]

Una vez derivada, esta nueva cláusula (lema) se agrega a la base de datos de cláusulas de la fórmula.[70] Esta cláusula aprendida es una consecuencia lógica de la fórmula original, meaning su adición no altera la satisfacibilidad del problema. Su propósito es "prohibir.[ex]plícitamente la combinación específica de asignaciones que condujo al conflicto, evitando así que el solucionador re-explore esa ruta improductiva en el futuro.

[59] 3
[60] 7
[61] 3
[62] 3
[63] 3
[64] 3
[65] 3
[66] 3
[67] 10
[68] 10
[69] 16
[70] 3

Las cláusulas aprendidas cumplen un doble papel: actúan como restricciones negativas, podando ramas que conducen a conflictos, y como guía positiva, dirigiendo al solucionador a una parte más prometedora del espacio de búsqueda a través de la posterior propagación de unidades después del retroceso no cronológico. Esta funcionalidad dual es fundamental para la eficiencia de CDCL, transformando una búsqueda pasiva de retroceso en una exploración activa impulsada por el conocimiento.

3.5. Retroceso no cronológico (Backjumping)

Tras la generación de una cláusula aprendida, CDCL ejecuta el retroceso no cronológico (backjumping), un diferenciador clave respecto al DPLL tradicional.[71] En lugar de simplemente deshacer la decisión más reciente, el backjumping aprovecha la información contenida en la nueva cláusula aprendida para determinar el nivel de decisión más relevante al que regresar.

El mecanismo para el backjumping es preciso: el nivel de retroceso se identifica como el segundo nivel de decisión más grande entre los literales de la cláusula aprendida. Si la cláusula aprendida contiene solo un literal (una cláusula unitaria), el nivel de retroceso se establece en 0.[72] Todos los literales asignados en niveles de decisión mayores que este nivel de retroceso calculado se eliminan del trail, deshaciendo efectivamente todas las decisiones y propagaciones que son irrelevantes para el conflicto.

Este salto estratégico permite al solucionador omitir muchas decisiones intermedias irrelevantes, podando significativamente el espacio de búsqueda.[73] Una propiedad crítica de las cláusulas aprendidas mediante el esquema 1-UIP es que garantizan exactamente un literal en el nivel de decisión más reciente antes del backjumping. Esto asegura que, inmediatamente después del backjumping, la nueva cláusula aprendida se convierta en una cláusula unitaria, permitiendo una nueva propagación de unidades. Esta propagación forzada impulsa la búsqueda desde un estado más informado, evitando que el solucionador re-explore los mismos subproblemas conflictivos.[74]

Para ilustrar la naturaleza dinámica del gráfico de implicaciones y el proceso de análisis de conflictos y backjumping, considere el siguiente rastro simplificado, basado en un ejemplo común en la literatura de CDCL[75]:

[71] 3
[72] 10
[73] 10
[74] 10
[75] 3

Cuadro 2: Rastro de ejecución del algoritmo CDCL

Paso/Acción	**Literal**	**ND**	**Cláusula**	**Notas**
Estado Inicial	-	0	-	Trail vacío.
Decisión 1	$x_1 = \text{V}$	1	DECISIÓN	Primera decisión.[76]
Propagación 1.1	$x_2 = \text{F}$	1	$(\neg x_1 \vee \neg x_2)$	x_2 forzado.[77]
Propagación 1.2	$x_3 = \text{V}$	1	$(\neg x_1 \vee x_3)$	x_3 forzado.[78]
Propagación 1.3	$x_4 = \text{F}$	1	$(\neg x_3 \vee \neg x_4)$	x_4 forzado.[79]
Propagación 1.4	$x_5 = \text{V}$	1	$(x_2 \vee x_4 \vee x_5)$	x_5 forzado.[80]
Decisión 2	$x_6 = \text{F}$	2	DECISIÓN	Segunda decisión.[81]
Propagación 2.1	$x_7 = \text{F}$	2	$(\neg x_5 \vee x_6 \vee \neg x_7)$	x_7 forzado.[82]
Propagación 2.2	$x_8 = \text{V}$	2	$(x_2 \vee x_7 \vee x_8)$	x_8 forzado.[83]
Propagación 2.3	$x_9 = \text{F}$	2	$(\neg x_8 \vee \neg x_9)$	x_9 forzado.[84]
Propagación 2.4	$x_{10} = \text{V}$	2	$(\neg x_8 \vee x_{10})$	x_{10} forzado.[85]
Propagación 2.5	$x_{11} = \text{V}$	2	$(x_9 \vee \neg x_{10} \vee x_{11})$	x_{11} forzado.[86]
Propagación 2.6	$x_{12} = \text{F}$	2	$(\neg x_{10} \vee \neg x_{12})$	x_{12} forzado.[87]
Conflicto!	$(\neg x_{11} \vee x_{12})$	2	CONFLICTO	Cláusula falsificada.[88]
Análisis	$(\neg x_8)$	-	1-UIP	Se aprende $(\neg x_8)$.[89]
Backjump	$x_8 = \text{F}$	0	Aprendida	Retrocede a ND=0.[90]

Este rastro detallado ilustra cómo CDCL aprovecha el gráfico de implicaciones para identificar la fuente de un conflicto, aprender una nueva cláusula que encapsula este conocimiento y luego retroceder no cronológicamente a un nivel de decisión relevante, podando efectivamente el espacio de búsqueda y guiando al solucionador hacia una solución de manera más eficiente.[91]

[91] 10

Referencias Bibliográficas

1. Boolean satisfiability problem - Wikipedia, consultado el 14 de agosto de 2025, https://en.wikipedia.org/wiki/Boolean_satisfiability_problem
2. Satisfiability: Algorithms, Applications and Extensions, consultado el 14 de agosto de 2025, https://sat.inesc-id.pt/~ines/sac10.pdf
3. Beyond VSIDS: Implementing CDCL, consultado el 14 de agosto de 2025, https://medium.com/@ishan-akhouri/beyond-vsids-implementing-cdcl-3e1214cfb8d4
4. Implementation of CDCL SAT solvers, consultado el 14 de agosto de 2025, http://ssa-school-2016.it.uu.se/wp-content/uploads/2016/06/LaurentSimon.pdf
5. LeowWB/cdcl-sat: CDCL sat solver - GitHub, consultado el 14 de agosto de 2025, https://github.com/LeowWB/cdcl-sat
6. thtran97/CDCL-based-SAT-Solver - GitHub, consultado el 14 de agosto de 2025, https://github.com/thtran97/CDCL-based-SAT-Solver
7. Conflict-Driven Clause Learning SAT Solvers, consultado el 14 de agosto de 2025, https://www.cs.princeton.edu/~zkincaid/courses/fall18/readings/SATHandbook-CDCL.pdf
8. Conflict Driven Clause Learning (CDCL) - GeeksforGeeks, consultado el 14 de agosto de 2025, https://www.geeksforgeeks.org/theory-of-computation/conflict-driven-clause-learning-cdcl/
9. Conflict-driven clause learning - Wikipedia, consultado el 14 de agosto de 2025, https://en.wikipedia.org/wiki/Conflict-driven_clause_learning
10. Conflict-driven clause learning (CDCL) SAT solvers, consultado el 14 de agosto de 2025, https://users.aalto.fi/~tjunttil/2020-DP-AUT/notes-sat/cdcl.html
11. Too much information: Why CDCL solvers need to forget learned clauses - PMC, consultado el 14 de agosto de 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC9417043/
12. Conflict-Driven Clause Learning SAT Solvers - ResearchGate, consultado el 14 de agosto de 2025, https://www.researchgate.net/publication/255409904_Conflict-Driven_Clause_Learning_SAT_Solvers
13. Understanding VSIDS Branching Heuristics, consultado el 14 de agosto de 2025, https://mk.cs.msu.ru/images/1/1f/SAT_SMT_Vijay_Ganesh_HVC2015.pdf
14. Iterative Learning Control — Algorithms, Applications and Future Research Directions, consultado el 14 de agosto de 2025, https://research.tue.nl/files/339814887/RogersChuMooOomTan2024.pdf
15. Hard Examples for Common Variable Decision Heuristics, consultado el 14 de agosto de 2025, https://www.csc.kth.se/~vinyals/research/papers/HardExamplesVsids.pdf
16. Clause Size Reduction with all-UIP Learning - PMC, consultado el 14 de agosto de 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC7326470/