Universidad de La Habana
Facultad de Matemática y Computación

Aplicación de la Lógica Computacional en la resolución de problemas

Autor:
Massiel Paz Otaño

Tutores:
Dr. Luciano García Garrido

Trabajo de Diploma
presentado en opción al título de
Licenciada en Ciencia de la Computación

Fecha

github.com/NinaSayers/Application-of-Computational-Logic-in-Problem-Solving.git

Dedicación

Agradecimientos

Agradecimientos

Opinión del tutor

Opiniones de los tutores

Resumen

Resumen en español

Abstract

Resumen en inglés

Índice general

Índice de figuras

Ejemplos de código

Introducción

El desarrollo de la lógica computacional como disciplina se enmarca en la revolución tecnológica del siglo XX, impulsada por la necesidad de resolver problemas complejos en ámbitos como la inteligencia artificial, la verificación de hardware y software, y la optimización industrial. La creciente demanda de sistemas automatizados capaces de procesar restricciones y tomar decisiones eficientes llevó a la comunidad científica a explorar métodos formales para modelar y resolver problemas combinatorios. En este escenario, la teoría de la complejidad computacional emergió como un pilar fundamental, especialmente tras la identificación de la clase NP-Completo por Cook en 1971, que transformó la comprensión de los límites de la computación.

Los problemas con restricciones —aquellos que requieren satisfacer un conjunto de condiciones lógicas— han sido centrales en áreas como la planificación, la criptografía y el diseño de circuitos. El problema de satisfacibilidad booleana (SAT), demostrado por Cook como el primer problema NP-Completo, se convirtió en la piedra angular para estudiar la viabilidad de soluciones eficientes. Aunque los primeros algoritmos para SAT, como el método de Davis-Putnam (DP) y su evolución, Davis-Putnam-Logemann-Loveland (DPLL), sentaron las bases de los resolvedores (solvers), su eficiencia se veía limitada por la explosión combinatoria en instancias complejas. La presencia de cláusulas unitarias, la selección subóptima de variables y el retroceso (backtrack) cronológico exponían claras debilidades, especialmente en problemas con miles de variables.

A pesar de los avances, los SAT resolvedores (solvers) clásicos enfrentaban un desafío crítico: escalar sin sacrificar completitud. Esto motivó la búsqueda de mejoras heurísticas y estratégicas, como el aprendizaje de cláusulas y el retroceso (backtrack) no cronológico, que culminaron en el surgimiento del paradigma Conflict-Driven Clause Learning (CDCL). CDCL no solo optimizó la exploración del espacio de soluciones, sino que introdujo mecanismos para evitar repeticiones de conflictos, marcando un hito en la resolución práctica de problemas NP-Completos.

El núcleo de la eficiencia de los SAT solvers modernos reside, sin lugar a dudas, en su capacidad para reducir el espacio de búsqueda de forma inteligente. Sin embargo, incluso con técnicas como CDCL, un desafío persiste: la selección óptima de variables. Esta elección determina la dirección en la que el algoritmo explora el árbol

de decisiones, y una estrategia subóptima puede llevar a ciclos de conflicto-reparación redundantes, incrementando exponencialmente el tiempo de ejecución. En problemas NP-Completos, donde el número de posibles asignaciones crece como 2^n (con n variables), una heurística de selección inadecuada convierte instancias resolubles en minutos en problemas intratables.

En CDCL, tras cada conflicto, el resolvedor aprende una cláusula nueva para evitar repeticiones. No obstante, la eficacia de este aprendizaje depende de qué variables se eligieron para bifurcar el espacio de soluciones. Si se seleccionan variables irrelevantes o poco conectadas a los conflictos, las cláusulas aprendidas serán débiles o redundantes, limitando su utilidad. Así, la selección de variables no es solo una cuestión de orden, sino de calidad de la exploración.

Dos de las heurísticas de selección de variables son VSIDS (Variable State Independent Decaying Sum) y DLIS (Dynamic Largest Individual Sum). Ambas, son aproximaciones greedy, dado que optimizan localmente (paso a paso) sin garantizar una solución global óptima. Su eficacia depende de cómo la estructura del problema se alinee con sus criterios. Por una parte, VSIDS asigna un puntaje a cada variable, incrementándolo cada vez que aparece en una cláusula involucrada en un conflicto. Periódicamente, estos puntajes se reducen (*decaimiento* exponencial), priorizando variables activas recientemente. Por otra parte, DLIS calcula, para cada literal (variable o su negación), el número de cláusulas no satisfechas donde aparece. Selecciona el literal con mayor frecuencia y asigna su variable correspondiente.

Hoy, aunque los SAT resolvedores basados en CDCL dominan aplicaciones críticas, desde la verificación formal de chips hasta la síntesis de programas, su rendimiento varía significativamente según el tipo de problema (p. ej., aleatorios vs. estructurados) y las heurísticas empleadas. Mientras VSIDS prioriza variables recientemente involucradas en conflictos —útil en problemas con alta estructura local—, DLIS enfatiza la frecuencia de aparición de literales, mostrando ventajas en dominios con distribución uniforme de restricciones. Esta dualidad plantea preguntas clave: ¿bajo qué métricas (tiempo de ejecución, memoria, escalabilidad) una estrategia supera a la otra? ¿Cómo influye la naturaleza del problema en su eficiencia?

Esta tesis aporta una comparación sistemática entre VSIDS y DLIS dentro de entornos CDCL, evaluando su desempeño en problemas heterogéneos (industriales, aleatorios y académicos). A diferencia de estudios previos, se integran métricas adaptativas que consideran no solo el tiempo de resolución, sino también el impacto de las cláusulas aprendidas y la distribución de conflictos. Además, se propone un marco teórico para clasificar problemas según su afinidad heurística, contribuyendo a la selección informada de algoritmos en aplicaciones reales.

Teóricamente, este trabajo profundiza en la relación entre estructura de problemas y heurísticas, enriqueciendo la comprensión de CDCL. Prácticamente, ofrece directrices para ingenieros y desarrolladores de resolvedores, optimizando recursos en áreas

como la verificación de software o la logística, donde minutos de mejora equivalen a ahorros millonarios.

Diseño teórico

- **Problema científico:** Ineficiencia de los SAT resolvedores ante problemas con distintas estructuras, asociada a la selección subóptima de variables.
- **Objeto de estudio:** Algoritmos CDCL con estrategias VSIDS y DLIS.
- **Objetivos:**
  - Analizar el impacto de VSIDS y DLIS en el rendimiento de CDCL.
  - Establecer correlaciones entre tipos de problemas y heurísticas eficaces.
- **Campo de acción:** Lógica computacional aplicada a la rersolución de problemas con restricciones.
- **Hipótesis:** El rendimiento de VSIDS y DLIS varía significativamente según la densidad de restricciones, la presencia de patrones locales y el balance entre cláusulas aprendidas y originales.

El documento se organiza en cinco capítulos:

- **Fundamentos de SAT y NP-Completitud:** Revisión teórica de problemas con restricciones y complejidad.
- **Evolución de los SAT resolvedores (solvers):** Desde DP/DPLL hasta CDCL.
- **Heurísticas en CDCL:** VSIDS vs. DLIS, ventajas y limitaciones.
- **Metodología experimental:** Diseño de pruebas, métricas y casos de estudio.
- **Resultados y conclusiones:** Análisis comparativo y recomendaciones prácticas.

Esta investigación busca no solo esclarecer el debate entre VSIDS y DLIS, sino también sentar bases para el diseño de heurísticas adaptativas, impulsando la próxima generación de resolvedores.

Capítulo 1

Marco Teórico

1.1. Fundamentos de SAT y NP-Completitud

El Problema de la Satisfacibilidad Booleana (SAT) es un problema de decisión combinatorio conceptualmente simple 1 y uno de los problemas más fundamentales en la ciencia de la computación 2 y matemáticas 4. Consiste en determinar si, dada una fórmula booleana 5, existe una asignación de valores de verdad a sus variables que haga que la fórmula sea verdadera 6. Tales asignaciones se llaman asignaciones satisfactorias o modelos 8.

La mayoría de los algoritmos y *solvers* de SAT operan con fórmulas representadas en FNC 9. Una fórmula en FNC es una conjunción ($\wedge$) de cláusulas 6, y una cláusula es una disyunción ($\vee$) de literales 6. Por su parte, un literal es una variable booleana (positiva) o su negación (negativa)6, y una asignación mapea cada variable a un valor booleano (verdadero/falso o 1/0) 6.

Una cláusula vacía () es siempre insatisfacible y representa un conflicto 6. Una fórmula en FNC es satisfacible si existe al menos una asignación que satisface todas sus cláusulas 6. SAT es un problema de decisión, lo que significa que la respuesta es "sí" (satisfacible) o "no" (insatisfacible) 18. Es un problema decidible, ya que existe un algoritmo (por ejemplo, basado en tablas de verdad) para determinar la satisfacibilidad de una fórmula 19.

Respecto a la NP-Completitud, el problema de SAT es el primer problema que se demostró ser NP-completo 21. Fue probado por Cook 2 (y el teorema de Cook-Levin 24) en su trabajo sobre la teoría de la complejidad computacional 2. SAT es considerado el prototipo 18 o el arquetipo 27 de los problemas NP-completos. El teorema de Cook-Levin establece que cualquier problema en la clase de complejidad NP puede reducirse a SAT en tiempo polinomial 26, por tanto que SAT sea NP-completo significa que, si pudiera ser resuelto en tiempo polinomial, entonces todos los problemas en la clase NP también podrían ser resueltos en tiempo polinomial, lo

que implicaría que $P = NP$ 2.

En general, se cree que no existen procedimientos o *solvers* eficientes (en tiempo polinomial) para resolver SAT en el peor caso 3. Todos los métodos completos conocidos tienen una complejidad exponencial en el peor caso 27. A pesar de esta dificultad teórica inherente, los solvers de SAT modernos han logrado un éxito notable y resuelven rutinariamente instancias muy grandes de aplicaciones del mundo real 3. Esto se debe a que explotan la estructura subyacente inherente a estas instancias prácticas 30, aunque instancias pequeñas, aleatorias o "artesanales" pueden ser difíciles 30.

1.1.1. Problemas de Satisfacción de Restricciones

Los Problemas de Satisfacción de Restricciones (CSPs por sus siglas en inglés) son una tecnología exitosa y extendida para resolver una amplia variedad de problemas, como asignación de recursos y planificación 39. Un CSP se define como un problema de decisión cuyo objetivo es determinar si existe una asignación de valores de un dominio finito a un conjunto finito de variables que satisfaga un conjunto dado y finito de restricciones 39. En la práctica, los problemas se especifican inicialmente de forma natural y luego se modelan utilizando lenguajes específicos, y los sistemas de *Constraint Programming* (CP) encuentran soluciones que satisfacen todas las restricciones 40.

La relación entre SAT y CSP es que SAT puede verse como un caso especial del Problema de Satisfacción de Restricciones Discretas Finitas (CSP) 41. En SAT, los dominios de las variables están restringidos a valores booleanos 43, mientras que en los CSPs generales, los dominios pueden ser conjuntos finitos arbitrarios 43. Además, los CSPs permiten relaciones arbitrarias entre subconjuntos de variables como restricciones, a diferencia de SAT donde las restricciones son exclusivamente cláusulas 10.

En cuanto a la complejidad de los CSPs, el Problema de Satisfacción de Restricciones Discretas Finitas es un subproblema NP-completo 43. Existe la posibilidad de reformular o convertir instancias de CSP en instancias de SAT 44, luego los solvers de SAT, altamente optimizados, pueden ser utilizados para resolver problemas formulados originalmente como CSPs 44. Las codificaciones específicas, como la de CSPs binarios a SAT, han demostrado reducir el tiempo de ejecución y el número de nodos de búsqueda para los *solvers* de SAT aplicados a ellos 45. Aunque muchos problemas se modelan más naturalmente como CSPs que como SAT 46, no está claro actualmente si los algoritmos nativos de CSP superan sustancialmente a los algoritmos de SAT altamente optimizados en codificaciones adecuadas 46.

1.2. Evolución de los SAT *solvers*

La evolución de los SAT *solvers* comenzó con algoritmos fundamentales como el procedimiento de Davis-Putnam (DP) y el procedimiento de Davis-Putnam-Logemann-Loveland (DPLL), y más tarde avanzó significativamente con el desarrollo de solucionadores de aprendizaje de cláusulas impulsado por conflictos (CDCL), lo que marcó un gran salto tanto en capacidades teóricas como prácticas.

1.2.1. Desde Davis-Putnam (DP) hasta Davis-Putnam-Logemann-Loveland (DPLL)

Introducido en 1960, el algoritmo DP fue un método completo temprano para resolver SAT que se basa en el Principio de Resolución (PR) para eliminar variables de manera sistemática. Funciona aplicando recursivamente PR y la eliminación de variables, pero a menudo sufre de un alto costo computacional debido al crecimiento exponencial en el número de cláusulas.

Por su parte, el algoritmo DPLL fue presentado en 1962 como un refinamiento de DP: DPLL aplica una búsqueda de retroceso en el espacio de asignación booleana combinada con propagación unitaria y heurísticas de eliminación de literales puros. Como un procedimiento de búsqueda en profundidad, DPLL asigna recursivamente valores de verdad a las variables y retrocede ante conflictos, podando el espacio de búsqueda de manera efectiva. Se convirtió en la base estándar para la resolución moderna de SAT y sigue estando en el núcleo de muchos *solvers*.

Aunque es más eficiente que DP, el algoritmo clásico DPLL encontró problemas de escalabilidad al enfrentarse a instancias SAT grandes o industriales debido a la exploración redundante de conflictos similares y a la falta de mecanismos de aprendizaje. Esto motivó la integración de estrategias de aprendizaje y heurísticas para guiar la búsqueda de manera más inteligente. Una de las mejoras más significativas fue *Conflict-Driven Clause Learning* (CDCL).

1.2.2. *Conflict-Driven Clause Learning* (CDCL)

CDCL fue introducido en el *solver* GRASP 9 en 1996 5, más de 30 años después del advenimiento de DPLL 12, y ha demostrado ser muy efectivo en problemas SAT del mundo real 12, además de revolucionar la investigación práctica en SAT 11.

Los CDCL SAT *solvers* evolucionaron como una extensión de DPLL al incorporar potentes técnicas de análisis de conflictos y aprendizaje de cláusulas. Los *solvers* de SAT contemporáneos se basan siempre en el framework DPLL extendido con CDCL 7, por lo que, a menudo se les llama CDCL *solvers*, y el término "solver de SAT" se

usa a menudo como sinónimo de "solver CDCL" en entornos prácticos debido a su popularidad y eficiencia 8.

La potencia de CDCL proviene de añadir varias técnicas cruciales al framework básico de DPLL. Una de las más importantes es el *Clause Learning* o Aprendizaje de Cláusulas, es el componente epónimo de CDCL 17. Esta permite aprender nuevas cláusulas al analizar la causa raíz de un conflicto 12. Esta técnica fue propuesta a mediados de los 90 y es una de las contribuciones teóricas más significativas de CDCL 6.

Otra de las técnicas más significativas aplicadas por CDCL es el *Backjumping* o *bactrack* no cronológico. En lugar de simplemente deshacer la última decisión (como el *backtracking* cronológico básico), el *solver* retrocede a un nivel de decisión más alto determinado por el análisis de conflictos. Esto evita explorar partes improductivas del árbol de búsqueda relacionadas con decisiones irrelevantes para el conflicto actual. El *backjumping* es una razón clave por la que los solucionadores basados en CDCL son mucho más eficientes que los *solvers* DPLL simples en muchas instancias.

1.2.3. Mejoras y heurísticas para CDCL

Las mejoras técnicas realizadas a los *solvers* SAT basados en CDCL han sido continuas y abarcan varias áreas clave para mejorar su rendimiento en la resolución de problemas de satisfacibilidad booleanos. Estas mejoras se centran en optimizar sus componentes esenciales: el aprendizaje de cláusulas, las heurísticas de decisión (ramificación), las estrategias de reinicio y la propagación unitaria, entre otros.

Estrategias de Reinicio (*Restarts*)

Los reinicios son características esenciales y muy importantes en los solucionadores CDCL 2. Permiten al *solver* abandonar una ruta de búsqueda improductiva y comenzar de nuevo desde un estado anterior, conservando información valiosa.

Las mejoras se centran en cuándo y cómo reiniciar. Se han explorado diversas políticas de reinicio, desde intervalos fijos hasta series geométricas y políticas más sofisticadas como las basadas en el LBD (*Literal Block Distance*) de las cláusulas aprendidas recientemente, similar a cómo lo hace Glucose.

Existen técnicas de reinicio avanzadas que utilizan información del estado del *solver* para decidir el momento óptimo, e incluso se ha investigado el uso de Aprendizaje por Refuerzo (*Reinforcement Learning - RL*) para determinar las políticas de reinicio, mostrando un rendimiento superior en algunos *benchmarks* 5.

Es posible hacer la distinción entre reinicios completos (*full reset*) y reinicios parciales (*partial reset*) que retienen un número constante de actividades de variables a través de los límites del reinicio 6. También se puede hacer una distinción entre reini-

cios "cálidos" (*warm restarts*) que preservan estado relevante (cláusulas aprendidas, actividad, fases) y "fríos" (*cold restarts*) que re-aleatorizan u olvidan más información.

Las investigaciones empíricas evalúan el impacto de diferentes políticas de reinicio en el rendimiento de los *solvers* 7.

Gestión de Cláusulas Aprendidas (*Learned Clause Management*)

Mantener y gestionar un gran número de cláusulas aprendidas es crucial para la eficiencia, ya que tener demasiadas cláusulas puede ser perjudicial para el rendimiento 12.

La calidad de las cláusulas aprendidas se mide a menudo por el LBD (*Literal Block Distance*)14. Las cláusulas con un LBD bajo (por ejemplo, LBD=2, llamadas "glue clauses") se consideran de alta calidad y son importantes para conservar 14. Las mejoras incluyen políticas avanzadas de eliminación de cláusulas 16, que a menudo se basan en métricas como el LBD para decidir qué cláusulas son menos útiles y deben ser eliminadas 14. Algunos solvers, como Kissat, implementan reciclaje de cláusulas ("glue clause recycling") 17.

Una técnica *in-processing* propuesta es la minimización de cláusulas aprendidas, que elimina literales redundantes de las cláusulas aprendidas 21. Esta técnica se activa antes de ciertos reinicios y mejora la calidad de las cláusulas, teniendo un impacto positivo en el rendimiento 21.

Heurísticas de Decisión (*Branching Heuristics*)

Los *solvers* CDCL de última generación utilizan heurísticas de decisión dinámicas 2 para seleccionar la siguiente variable a asignar y su valor. VSIDS es una heurística clásica 2, pero existen variantes 2. Otras heurísticas incluyen CHB y LRB 18. Las mejoras buscan hacer la selección de variables más efectiva. Una técnica es utilizar la frecuencia de conflictos de las variables observada durante la búsqueda local para dirigir las heurísticas de ramificación en el CDCL 1. Identificar y priorizar variables que aparecen en cláusulas de alta calidad también puede guiar las decisiones, ya que estas variables tienden a ser más eficientes en términos de inferencia y conflicto 15. Se ha desarrollado un esquema de "*variable bumping*" basado en la aparición en cláusulas "glue" 15.

El *phase saving* es una técnica para recordar la última asignación exitosa de una variable y usarla en decisiones futuras 16. Las *target phases* extienden esta idea, buscando mantener y extender asignaciones prometedoras 1.

Propagación Unitaria (Unit Propagation - UP / BCP)

La propagación unitaria es un procedimiento fundamental y a menudo el más costoso en tiempo de ejecución en CDCL. Su eficiencia es crítica.

La implementación eficiente se basa en el esquema de los dos literales vigilados (*Two-Watched Literals - TWL*) 29, que reduce drásticamente el número de cláusulas a revisar cuando se asigna un literal.

Las mejoras más recientes se centran en priorizar el orden en que se examinan las cláusulas durante la propagación unitaria para encontrar conflictos o implicaciones más rápidamente 96. Ejemplos incluyen *Core First Unit Propagation* (CFUP), que prioriza cláusulas "*core*" (por ejemplo, con LBD <= 7) 96, y *Priority Propagation* (PriPro).

Una nueva técnica propone usar la noción de literales "estables" (literales que tienden a ser satisfechos por largos períodos) para seleccionar los literales vigilados, mejorando CaDiCaL en algunos *benchmarks* 30.

Integración con Búsqueda Local (*Local Search*)

Una línea de mejora reciente y significativa es la integración estrecha de CDCL con algoritmos de búsqueda local 1. Esto implica técnicas como: (1) explorar ramas prometedoras utilizando búsqueda local para encontrar asignaciones parciales que puedan extenderse a soluciones (técnica rx)26; (2) usar búsqueda local para determinar las fases de las variables (*re-phasing* basado en búsqueda local - rp)26; y (3) dirigir las heurísticas de ramificación del CDCL utilizando la frecuencia con la que las variables aparecen en conflictos durante la búsqueda local (cf)1.

Estas técnicas combinadas han demostrado mejoras significativas en el rendimiento de *solvers* como Glucose y Maple, especialmente en instancias satisfacibles 1, y se considera que contribuyeron al aumento del rendimiento de los mejores *solvers* en competiciones recientes 1.

Frameworks de Optimización y Meta-Heurísticas

Se están explorando enfoques que utilizan técnicas de *Machine Learning* para ajustar y optimizar heurísticas automáticamente 35. AutoSAT es un ejemplo que usa *Large Language Models* (LLMs) para modificar heurísticas, logrando resultados competitivos 35.

El *framework Multi-Armed Bandit* (MAB) se utiliza para combinar y seleccionar dinámicamente diferentes heurísticas (como VSIDS y CHB), a menudo orquestado por reinicios 37. *Kissat MAB*, que utiliza MAB, ganó la pista principal de *SAT Competition 2021* 37. Estas mejoras se han implementado en solvers de última generación

como Glucose, CaDiCaL, Kissat, MapleSAT y sus variantes, que son los principales contendientes en las competiciones de SAT 6. MiniSat a menudo sirve como una base clásica para la investigación y comparación 4. El desarrollo continúa, buscando nuevas formas de optimizar el proceso de búsqueda, el aprendizaje y la gestión de información para resolver instancias SAT cada vez más difíciles.

1.3. Tipos de problemas o *benchmarks*

Históricamente, existían heurísticas como DLIS (*Dynamic Largest Individual Sum*), que se basaba en la frecuencia con la que una variable aparecía en cláusulas insatisfechas, lo que se relaciona con la centralidad de grado en el grafo de incidencia. Sin embargo, los *solvers* CDCL modernos dependen en gran medida de heurísticas dinámicas que se adaptan al progreso de la búsqueda 1.

Para evaluar la eficiencia de estas heurísticas como VSIDS, se utilizan conjuntos de problemas (benchmarks) con características variadas. Los más comunes provienen de las competiciones anuales de SAT (*SAT Competition*) 15.

Los *benchmarks* de las *SAT Competition* se clasifican típicamente en varias categorías 17:

Instancias de Aplicación (*Application instances*)

Son problemas derivados de aplicaciones del mundo real, como verificación de *software* y *hardware*, planificación, criptografía, etc. 11. Estas instancias suelen tener una estructura subyacente que puede ser explotada por los *solvers* CDCL 20. La estructura, como la modularidad o la existencia de "*backdoors*" pequeños, puede correlacionar con el tiempo de resolución 21.

Instancias Combinatorias Dificultosas (*Hard Combinatorial instances / Crafted instances*)

Son problemas construidos artificialmente o codificaciones de problemas combinatorios o matemáticos (como problemas de coloración, o el principio del palomar) diseñados específicamente para ser difíciles para los *solvers* 16. A menudo son más pequeñas que las instancias de aplicación pero pueden ser muy difíciles de resolver 19. Tienden a tener menos estructura que las de aplicación 21.

Instancias Aleatorias (*Random instances*)

Generadas aleatoriamente, como las instancias Random k-SAT 17. Los *solvers* CDCL son, en general, notablemente pobres resolviendo este tipo de instancias 17.

Su falta de estructura discernible las hace difíciles para las técnicas de aprendizaje de cláusulas basadas en conflictos 21.

Instancias Ágiles (*Agile instances*)

Un tipo específico de instancias *bit-blasted* (resultantes de convertir problemas de lógicas superiores a SAT) utilizadas en algunas competiciones 19.

VSIDS y otras heurísticas modernas se evalúan principalmente en instancias de Aplicación y Combinatorias Dificultosas, ya que son los tipos de problemas en los que los *solvers* CDCL son más eficientes 17. Aunque VSIDS es la heurística dominante, se ha demostrado que diferentes heurísticas pueden rendir mejor en distintas familias de instancias 30.

Además de la clasificación por origen (Aplicación, Combinatorio, Aleatorio, Ágil), las instancias se pueden clasificar y analizar según otros criterios:

Satisfacibilidad (SAT vs UNSAT)

Si la instancia tiene al menos una asignación que satisface la fórmula (SAT) o si es inherentemente insatisfacible (UNSAT) 17. El comportamiento del *solver* y la efectividad de las heurísticas pueden diferir significativamente dependiendo de si la instancia es SAT o UNSAT 41.

Propiedades Estructurales

Características de la fórmula vistas como un grafo (grafo de incidencia variable/-cláusula). Estas incluyen 20:

- **Centralidad:** Qué tan conectadas están las variables o cláusulas.
- **Modularidad/Estructura Comunitaria:** Si las variables pueden agruparse en "comunidades" fuertemente conectadas internamente pero débilmente conectadas entre sí 20. Instancias de aplicación tienden a alta modularidad 21.
- ***Backdoors***: Pequeños conjuntos de variables cuya asignación correcta hace que el resto de la fórmula sea fácil de resolver 22. Se sugiere que las instancias industriales a menudo tienen "*backdoors*" pequeños 23.
- **Backbones:** Variables que deben tener un valor específico en cualquier asignación satisfacible 22.
- ***Treewidth* (Ancho de árbol):** Una medida de la "parecido a un árbol" de la estructura de la fórmula. Estas propiedades estructurales se utilizan para

entender por qué algunas instancias son difíciles y cómo se comportan los *solvers* en ellas 20.

- **Relación Cantidad de Cláusulas vs. Cantidad de Variables (Densidad)** La relación entre el número de cláusulas y el número de variables en una fórmula en FNC es un criterio importante, a menudo denominado densidad 46. Para las instancias aleatorias k-SAT, la densidad es un factor crítico. Existe un fenómeno de transición de fase alrededor de una densidad específica (aproximadamente 4.27 para 3-SAT) donde las instancias son más difíciles de resolver 25. Por debajo de esta densidad, la mayoría de las instancias son SAT y relativamente fáciles; por encima, la mayoría son UNSAT y también más fáciles que las de la zona de transición 25. En cambio, para las instancias de aplicación o combinatorias, la relación no es tan simple. A veces, codificaciones más compactas (con menos variables o cláusulas, pero quizás más complejas) pueden ser más difíciles de resolver que codificaciones más dispersas 48. Sin embargo, la densidad sigue siendo una característica utilizada para describir y clasificar instancias 46.

Capítulo 2

Propuesta

Capítulo 3

Detalles de Implementación y Experimentos

Conclusiones

El dilema en CDCL CDCL mitiga parcialmente este problema mediante el aprendizaje de cláusulas, pero no elimina la dependencia de la selección inicial de variables. Por ejemplo:

Si VSIDS elige variables periféricas en un problema con núcleos críticos (ej: PHP), el solver gastará recursos en regiones irrelevantes.

Si DLIS prioriza literales frecuentes en problemas con restricciones jerárquicas (ej: scheduling), perderá la capacidad de explotar correlaciones locales.

Esta interdependencia entre heurísticas y estructura del problema explica por qué, a pesar de los avances en CDCL, no existe una estrategia universalmente óptima. La selección de variables sigue siendo un cuello de botella teórico y práctico, especialmente al escalar a miles de variables con relaciones complejas.

La introducción de CDCL marcó un avance al reemplazar el retroceso (backtrack) cronológico con uno dirigido por conflictos, pero su éxito está ligado a la sinergia entre aprendizaje y selección de variables. Mientras las cláusulas aprendidas reducen el espacio de búsqueda, las heurísticas de selección determinan cómo se navega en él. Un desbalance entre estos componentes condena al solver a un rendimiento subóptimo, perpetuando la necesidad de estudios comparativos como el propuesto en esta tesis.

Recomendaciones

Recomendaciones