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

Introducción **1**

1. Estado del Arte **4**

2. Propuesta **13**

3. Detalles de Implementación y Experimentos **14**

Conclusiones **15**

Recomendaciones **16**

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 o 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. Sin embargo, la eficacia de estos algoritmos depende en gran medida de estrategias de selección de variables, como VSIDS (Variable State Independent Decaying Sum) y DLIS (Dynamic Largest Individual Sum), cuyas ventajas comparativas siguen siendo objeto de debate.

Hoy, los SAT resolvedores (solvers) basados en CDCL dominan aplicaciones críti-

cas, desde la verificación formal de chips hasta la síntesis de programas. No obstante, 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 (solvers), 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 (solvers) 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 (solvers).

Capítulo 1

Estado del Arte

1. Visión de todo el problema de solución de problemas con restricciones. 2. Uso de SAT... hasta DPLL 3. SAT resuelve el problema pero es ineficiente 4. Yo voy a tratar las ineficiencias de DPLL: Al encontrar 1 cláusula conflicto se hace un backtrack cronológico (hacia el anterior nivel decisión) (poner ejemplos) que no necesariamente es el causante del conflicto. Una de las grandes extensiones para combatir este problema, es utilizar CDCL-SAT (cómo funciona este algoritmo, se explica en el 1er capítulo técnicamente DPLL y CDCL para construir un SAT SOLVER). Decir que su objetivo es aprender una cláusula y obtener el nivel de decisión al que debe orientarse nuevamente el proceso de satisfacibilidad, evitando caer nuevamente en el conflicto. Incorpora la cláusula a la base de datos, como una restricción más del problema. En un momento determinado puede ocurrir un restart. El objetivo de esta tesis es, aparte de enlazar una propuesta de CDCL (modificar el programa) como extensión de un SAT-SOLVER, es analizar problemas que han quedado pendientes. Uno de ellos, es el problema de la selección de variables en los niveles de decisión. En esta tesis, se van a analizar algunas heurísticas que han sido propuestas para tratar de hacer aún más eficientes el trabajo de un CDCL-SAT-SOLVER. Se analizan ambas heurísticas, se programan, y se realizan comparaciones para ver la efectividad de las mismas, que pueden estar sujetas al conjunto de cláusulas que tengan las variables.

Visión de todo el problema de solución de problemas con restricciones. El Problema de Satisfacción de Restricciones (CSP) es un problema fundamental en diversas áreas1. Una instancia de un CSP se especifica formalmente mediante una tripla ((V, D, C))1: • V es un conjunto finito de variables1. • (**D**) es un conjunto de dominios, donde cada ($D_i \in D$) es el conjunto de posibles valores para la variable ($v_i \in V$). El conjunto (D_i) se denomina el dominio de (v_i)1. • C es un conjunto finito de restricciones1. Cada restricción en (C) es un par ((scope, relation)), donde (scope) es una lista ordenada de variables involucradas en la restricción, y (relation) es una relación sobre el dominio que define las combinaciones de valores permitidas para esas variables2.

Un CSP se considera binario si el alcance de cada restricción incluye exactamente dos variables3. Estos pueden representarse gráficamente donde los nodos son variables y las aristas son restricciones entre dos nodos3. Una solución a una instancia de CSP es una asignación de valores de sus respectivos dominios a cada una de las variables en (V), de tal manera que se satisfagan simultáneamente todas las restricciones en (C)3. Una instancia de CSP puede tener una solución, múltiples soluciones, o ninguna solución3. Más allá de encontrar una solución, a veces se busca el número de soluciones o una solución óptima3. La satisfacibilidad en CSPs se refiere precisamente a la existencia de al menos una solución que cumpla con todas las restricciones4. La factibilidad es un término usado en problemas de optimización para describir una solución que no solo satisface las restricciones sino que también cumple con criterios adicionales (como maximización o minimización)4. Se puede concebir la búsqueda de una solución a un problema SAT, y por extensión a cualquier problema NP, como un problema de búsqueda en un Espacio de Búsqueda5. El problema de la satisfacibilidad (SAT), que consiste en determinar si una fórmula de lógica proposicional es o no satisfacible, es un prototipo de los problemas NP-completo6.... SAT en sí mismo puede ser visto como un tipo específico de CSP8. Los problemas de CSP son generalmente decidibles, aunque su complejidad y decidibilidad pueden depender de instancias y restricciones específicas4.... El método de tablas de verdad para SAT tiene una complejidad exponencial (2^n) (donde (n) es el número de variables), lo que lo hace inadecuado para fórmulas grandes7.... Los CSPs encuentran aplicación en una amplia gama de dominios11..., incluyendo: • Gestión de recursos de radio (RRM)11. • Bases de datos (cálculo de joins, actualizaciones de vistas)11. • Razonamiento temporal y espacial11. • Planificación, programación (scheduling), asignación de recursos12. • Diseño y configuración12. • Verificación de hardware e ingeniería de software12. • Diagnóstico cualitativo y razonamiento12. • Modelado de líneas de ensamblaje con capacidad limitada11. • El problema de las n-reinas es un ejemplo clásico de un CSP simple2.... Para resolver un CSP, primero se debe desarrollar un Modelo, el cual luego se entrega a un Solver apropiado14. El modelado requiere un lenguaje declarativo para expresar el problema14. Existen diferentes enfoques o métodos para resolver problemas combinatorios que pueden formularse como CSPs15: • Constraint Programming (CP): Utiliza variables con dominios finitos y combina estrategias de búsqueda con inferencias que reducen el espacio de búsqueda8. • Boolean Satisfiability (SAT): Emplea variables proposicionales y cláusulas como única forma de restricción, utilizando algoritmos dedicados que combinan búsqueda, inferencias lógicas (propagación unitaria) y aprendizaje a partir de fallas8. Los resolvedores SAT modernos, especialmente los basados en CDCL (Conflict-Driven Clause Learning), son motores altamente eficientes capaces de resolver fórmulas con millones de variables y tienen un amplio despliegue industrial9. CDCL extiende el algoritmo DPLL con aprendizaje de cláusulas, reinicios de búsqueda y heurísticas de ramificación16. • Mixed Integer Linear

Programming (MILP): Las variables pueden ser enteras o reales, y las soluciones se encuentran típicamente mediante búsqueda branch and bound, utilizando relajaciones lineales17. Aunque es posible resolver el mismo problema con cualquiera de estos métodos, difieren en cómo se modela el problema y en la metodología de solución15. Las fuentes mencionan que este curso se restringe a métodos determinísticos, completos (exactos) para resolver CSPs, capaces de solucionar problemas con miles e incluso millones de variables12. Uso de SAT... hasta DPLL la evolución de los SAT Solvers y su uso en la resolución de problemas con restricciones ha sido un camino marcado por avances algorítmicos significativos, partiendo de principios teóricos fundamentales hacia motores de resolución altamente eficientes. El punto de partida se encuentra en la Lógica Proposicional (LP) y el problema fundamental de determinar si una fórmula es satisfacible o insatisfacible, conocido como el Problema de la Satisfacibilidad (SAT)1.... Este problema es crucial, ya que fue el primero en ser clasificado como NP-completo1..., lo que teóricamente implica que, bajo la suposición razonable de que $P! = NP$, cualquier algoritmo completo para SAT requeriría tiempo exponencial en el peor de los casos3. Sin embargo, la historia de los SAT Solvers es una "historia de éxito de la ciencia de la computación", desafiando este peor caso y transformándose de una curiosidad académica capaz de resolver cientos de variables en los años 90 a motores que rutinariamente manejan millones de variables y se usan ampliamente en la industria3.... SAT es un problema de decisión decidible; aunque el método de tablas de verdad es un algoritmo completo, su complejidad exponencial ((2^n), donde (n) es el número de variables) lo hace inadecuado para fórmulas grandes2.... La evolución algorítmica para abordar SAT de manera más práctica comenzó con el Principio de Resolución6. La resolución proposicional es un procedimiento de decisión completo para fórmulas en Forma Normal Conjuntiva (FNC), aunque impráctico en la práctica6. A partir de este principio, surgió el algoritmo Davis-Putnam (DP)7. Este fue un primer intento de resolver problemas SAT representados en FNC aplicando la resolución como un componente esencial7. El algoritmo Davis-Putnam-Logemann-Loveland (DPLL)8... representó una mejora crucial sobre el algoritmo DP original9. DPLL aplica una estrategia que elimina la posible memoria exponencial que podría requerir DP9. DPLL utiliza procedimientos introducidos por el algoritmo DP, como la Propagación Unitaria (PU) y la Eliminación de Literales Puros (ELP), aplicándolos prioritariamente siempre que sea posible en cada etapa de la búsqueda9. El algoritmo DPLL es un algoritmo completo que integra búsqueda e inferencia, realizando un proceso de ramificación + propagación unitaria + retroceso (backtracking)10. Cada nodo en el árbol de búsqueda generado por DPLL, denotado por una variable, representa un nivel de decisión10. DPLL constituye el núcleo básico en la construcción de algoritmos resolvedores para Problemas de Satisfacción de Restricciones (PSRs), que realizan esta integración de búsqueda e inferencia10. El algoritmo DPLL comparte similitudes con el procedimiento estándar de búsqueda con retroceso,

donde la propagación unitaria se ejecuta después de cada asignación de decisión11. La era moderna de los SAT Solvers está dominada por los Conflict-Driven Clause Learning (CDCL) SAT solvers3.... CDCL es una extensión del algoritmo DPLL y es la principal razón del éxito de los SAT Solvers modernos3.... Aunque implementan la búsqueda con retroceso y la propagación unitaria como DPLL, los CDCL solvers introducen diferencias fundamentales que mejoran drásticamente su eficiencia11...: 1.Aprendizaje de Cláusulas (Clause Learning): Cuando ocurre un conflicto (una asignación parcial que viola las restricciones), el solver analiza la causa del conflicto y aprende una nueva cláusula (cláusula aprendida o lemma) que encapsula esa inconsistencia11.... Esta cláusula se agrega a la base de cláusulas y poda el espacio de búsqueda al prevenir que el solver vuelva a visitar combinaciones de asignaciones conflictivas15.... El aprendizaje de cláusulas es descrito como una característica revolucionaria"que transforma el solver de una herramienta de fuerza bruta a un sistema inteligente capaz de abordar problemas complejos y masivos15.... La relación entre el análisis de conflictos de CDCL y la resolución es estrecha; las cláusulas aprendidas pueden explicarse mediante pasos de resolución18. 2.Retroceso No Cronológico (Non-chronological Backtracking o Backjumping): 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 conflictos11.... Esto evita explorar partes improductivas del árbol de búsqueda relacionadas con decisiones irrelevantes para el conflicto actual13. El backjumping es una razón clave por la que los solvers basados en CDCL son mucho más eficientes que los solvers DPLL simples en muchas instancias13. 3.Reinicios de Búsqueda (Restarts): Periódicamente, el solver puede reiniciar la búsqueda desde el primer nivel de decisión, conservando las cláusulas aprendidas11.... Esto ayuda a escapar de regiones improductivas del espacio de búsqueda ("stagnation")20.... Los reinicios funcionan efectivamente junto con el aprendizaje de cláusulas19.... 4.Heurísticas de Selección de Variables Dirigidas por Conflictos (Conflict-Driven Branching): Priorizan variables que han estado involucradas en conflictos recientes o cláusulas aprendidas, enfocando la búsqueda en las partes más .ªctivas"del problema20.... VSIDS (Variable State Independent Decaying Sum) es una heurística de ramificación de referencia utilizada en la mayoría de los solvers modernos20.... 5.Estructuras de Datos Eficientes: Implementaciones optimizadas, como las "watched literals"(literales observados), mejoran la eficiencia de la propagación unitaria, que es una de las tareas más importantes que realiza un SAT Solver11.... La combinación de estas técnicas (aprendizaje de cláusulas, backjumping, reinicios, heurísticas dirigidas por conflictos y estructuras de datos eficientes) permite a los SAT Solvers modernos, como MiniSat, Glucose y Lingeling, abordar problemas a gran escala con millones de variables y cláusulas5.... En cuanto al uso de los SAT Solvers (especialmente CDCL) para resolver problemas con restricciones, se basa principalmente en la idea de codificar o traducir el problema de restriccio-

nes a una instancia de SAT31.... Dado que SAT es el prototipo de los problemas NP-completo, muchos problemas de la clase NP, incluyendo los CSPs (Problemas de Satisfacción de Restricciones), pueden ser formulados o traducidos a SAT1.... Aunque DPLL ya constituía un núcleo para resolvedores de PSRs10, los avances en CDCL han potenciado enormemente esta capacidad. Existe un extenso trabajo en traducir dominios más expresivos, incluyendo CSPs, a SAT para ser resueltos por SAT Solvers CDCL32. Hay herramientas específicas que implementan traducciones de CSP a SAT32. Además de la traducción directa de CSP a SAT, los SAT Solvers CDCL se utilizan como .ºráculos productores de testigos"para la clase NP33.... Esto significa que no solo responden "Sí.º "No.ª la satisfacibilidad, sino que, en caso afirmativo, proporcionan una asignación que satisface la fórmula (la "solución.º "testigo")33.... En caso negativo (insatisfacible), a menudo pueden proporcionar un subconjunto insatisfacible de cláusulas ("núcleo insatisfacible")33. Los SAT Solvers CDCL también han influido fuertemente en el diseño y son a menudo los motores principales de otros resolvedores para problemas relacionados en el área de la resolución de restricciones y optimización, como MaxSAT, QBF, Satisfiability Modulo Theories (SMT), Answer Set Programming (ASP), y Lazy Clause Generation (LCG) en programación con restricciones35.... En LCG, por ejemplo, se utiliza la propagación a través de la generación de cláusulas "lazy"37, lo que recuerda el concepto de aprendizaje de cláusulas en CDCL. En resumen, la evolución desde la Resolución y DP hasta DPLL, y luego la revolución del CDCL, ha transformado el problema teóricamente difícil de SAT en algo manejable para instancias a gran escala. Esta capacidad, combinada con la posibilidad de codificar muchos problemas con restricciones a SAT, ha convertido a los SAT Solvers (particularmente los CDCL) en herramientas poderosas y motores subyacentes para resolver una amplia gama de Problemas de Satisfacción de Restricciones y otros problemas combinatorios en diversas aplicaciones35....

SAT resuelve el problema pero es ineficiente los aspectos relacionados con la propagación unitaria y la selección de variables son cruciales para entender la eficiencia de los solvers SAT, tanto en sus versiones básicas como en las modernas. Procederé a reformular la explicación de los problemas de ineficiencia de los enfoques básicos para resolver SAT (como el DPLL original) incorporando estos puntos, basándome estrictamente en las fuentes y nuestra conversación previa. Aunque SAT es decidible –siempre hay un algoritmo que encuentra la respuesta–1, su complejidad teórica y las limitaciones de los algoritmos iniciales lo hacían ineficiente para instancias grandes. Los solvers básicos de SAT (como el algoritmo Davis-Putnam-Logemann-Loveland - DPLL original2...) enfrentaban, y en su forma pura aún enfrentarían, problemas de eficiencia ligados a tres aspectos principales: 1. La Complejidad Inherente del Problema (Tiempo Exponencial en el peor caso): SAT es un problema NP-completo4. Esto implica que, asumiendo $P! = NP$, cualquier algoritmo completo que resuelva SAT para cualquier instancia necesitará, en el peor de los casos, un tiempo de ejecución

que crece exponencialmente con el tamaño de la entrada, típicamente el número de variables4.... El método fundamental de las tablas de verdad lo demuestra: requiere verificar (2^n) interpretaciones, donde (n) es el número de variables4.... Aunque DPLL utiliza técnicas más sofisticadas como ramificación e inferencia (Propagación Unitaria), sigue teniendo una complejidad temporal exponencial en el peor caso5. Esta limitación inherente significa que, sin mejoras sustanciales, resolver instancias con muchas variables rápidamente es computacionalmente intratable1.... 2. Limitaciones en la Estrategia de Búsqueda y Selección de Variables: El algoritmo DPLL aplica una estrategia de ramificación + propagación unitaria + retroceso3. La ramificación implica seleccionar una variable no asignada (nodo en el árbol de búsqueda) y asignarle un valor de verdad (ramificando)5.... La selección de la variable para ramificar (y el valor a asignar) es una decisión heurística que influye significativamente en la eficiencia8.... Los enfoques básicos de DPLL pueden utilizar estrategias de selección simples o un orden fijo9.... Una selección de variable poco informada puede llevar al solver a explorar secciones improductivas del espacio de búsqueda12. A diferencia de los solvers CDCL modernos que usan heurísticas sofisticadas basadas en la actividad de las variables (como VSIDS), que priorizan variables involucradas en conflictos recientes para dirigir la búsqueda de manera más efectiva13..., los enfoques básicos carecen de esta capacidad de dirigir la búsqueda hacia áreas más críticas o propensas a revelar insatisfacibilidad rápidamente13.... Esta selección de variable menos estratégica contribuye a explorar ineficientemente el gran espacio de búsqueda12. 3. Manejo Ineficiente de Conflictos (Retroceso Cronológico y Falta de Aprendizaje): La propagación unitaria (PU), también conocida como Boolean Constraint Propagation (BCP), es un componente clave del DPLL2.... La PU itera la regla de la cláusula unitaria: si una cláusula tiene todos sus literales menos uno asignados a falso (es una cláusula unitaria), el literal restante debe ser asignado a verdadero para satisfacer la cláusula18.... La aplicación exhaustiva de PU permite deducir asignaciones lógicas y, crucialmente, detectar conflictos cuando una asignación parcial hace que una cláusula sea completamente falsa9.... Sin embargo, el manejo de estos conflictos en el DPLL original es ineficiente12. Utiliza un retroceso (backtracking) cronológico5.... Esto significa que, al encontrar un conflicto, simplemente deshace la última decisión tomada y retrocede al nivel de decisión inmediatamente anterior para probar la alternativa12. Este enfoque tiene dos problemas principales: ◦ No aborda la causa raíz del conflicto: El retroceso cronológico no siempre retrocede al nivel de decisión relevante que originó el conflicto12. Esto puede llevar a la exploración repetida de asignaciones parciales que ya se sabe que son inconsistentes12. ◦ Falta de aprendizaje: Los algoritmos DPLL básicos no aprenden de los conflictos22. No analizan por qué ocurrió el conflicto ni derivan nuevas cláusulas (cláusulas aprendidas o "lemmas"16...) que resuman la causa de la inconsistencia22.... Esto significa que el solver puede volver a tropezar con la misma combinación de asignaciones inconsistentes más adelante en

la búsqueda12..., desperdiciando esfuerzo. Los solvers CDCL superan esto mediante el análisis de conflictos para aprender nuevas cláusulas y usando retroceso no cronológico (backjumping) para saltar directamente a un nivel de decisión anterior que pueda resolver la causa del conflicto16.... En resumen, mientras que la Propagación Unitaria es una técnica de inferencia eficiente en sí misma (en tiempo lineal32) que es fundamental para detectar conflictos18..., los problemas de eficiencia en los solvers básicos como el DPLL original residían en la complejidad exponencial inherente al problema SAT, una estrategia de búsqueda y selección de variables menos sofisticada que no dirigía eficientemente la exploración del espacio, y sobre todo, un manejo ineficiente de los conflictos caracterizado por el retroceso cronológico y la ausencia de aprendizaje5.... Las técnicas introducidas en los solvers CDCL, como el aprendizaje de cláusulas, el retroceso no cronológico (backjumping), y las heurísticas de ramificación dirigidas por conflictos, se desarrollaron específicamente para mitigar estos problemas13.... Yo voy a tratar las ineficiencias de DPLL podemos explicar por qué las extensiones modernas como CDCL y otras heurísticas son fundamentales para superar la ineficiencia de los algoritmos SAT básicos, sin entrar en detalles técnicos de su implementación. Como hemos visto, el problema de la Satisfacibilidad (SAT) es intrínsecamente difícil, siendo NP-completo, lo que implica un tiempo de ejecución exponencial en el peor caso para cualquier algoritmo completo1.... Los algoritmos básicos como el Davis-Putnam-Logemann-Loveland (DPLL) original, aunque decidibles4, sufren de ineficiencias clave como el retroceso cronológico y la falta de aprendizaje, lo que limita severamente su rendimiento en instancias grandes5.... Las técnicas modernas, particularmente aquellas agrupadas bajo el paraguas de Conflict-Driven Clause Learning (CDCL), representan la razón principal por la que los solvers SAT pasaron de ser una çuriosidad académica.[en] los 90, capaces de manejar solo unos pocos cientos de variables, a "motores altamente eficientes"que rutinariamente resuelven fórmulas con millones de variables hoy en día1.... CDCL es un potente avance sobre el DPLL básico7. La fortaleza de CDCL radica principalmente en su capacidad para aprender de los conflictos7.... Cuando se encuentra una inconsistencia (un conflicto), el solver CDCL analiza el conflicto para derivar una nueva cláusula, llamada cláusula aprendida (o "lemma")8.... Esta cláusula aprendida encapsula la causa raíz del conflicto y se agrega a la base de cláusulas8.... El impacto de este aprendizaje es significativo porque: 1.Poda el espacio de búsqueda: Las cláusulas aprendidas impiden que el solver vuelva a visitar combinaciones de asignaciones que ya se sabe que conducen a conflictos8.... Esto reduce drásticamente la probabilidad de conflictos redundantes y permite que el solver se enfoque en áreas más prometedoras del espacio de búsqueda, acelerando la convergencia9.... 2.Mejora la Propagación Unitaria: Las cláusulas aprendidas enriquecen la fórmula, haciendo que la Propagación Unitaria (PU) sea más efectiva11.... A medida que se hacen asignaciones, estas nuevas cláusulas pueden volverse unitarias más fácilmente, forzando asignaciones lógicas y reduciendo la

necesidad de tomar decisiones arbitrarias12.... Esto hace que PU sea más potente11. Además del aprendizaje, CDCL utiliza el retroceso no cronológico (backjumping)7.... En lugar de simplemente deshacer la última decisión al encontrar un conflicto, el solver calcula un nivel de backjump basado en el análisis del conflicto12. Esto le permite saltar directamente a un nivel de decisión anterior que es relevante para resolver la causa del conflicto, evitando explorar partes irrelevantes del árbol de búsqueda12.... Más allá de la dupla aprendizaje-backjumping (que son ingredientes clave que funcionan juntos11), la eficiencia de los solvers modernos se potencia con otras estrategias y heurísticas: • Heurísticas de Selección de Variables: La elección de qué variable ramificar y qué valor asignarle es crucial6.... Los enfoques básicos pueden ser menos informados6. Los solvers CDCL modernos emplean heurísticas dinámicas y dirigidas por conflictos11.... Un ejemplo destacado es VSIDS (Variable State Independent Decaying Sum)18.... Aunque no entraremos en cómo calcula el puntaje exactamente, la idea es que prioriza variables que han estado involucradas en conflictos recientes20.... Esto dirige la búsqueda hacia las partes más .ªctivas"del problema, donde es más probable encontrar una solución o una refutación rápidamente11.... Históricamente, heurísticas como DLIS (Dynamic Largest Individual Sum) y DLCS (Dynamic Largest Combined Sum) sentaron las bases para esta idea de explotación dinámica de la actividad de los literales18. Estas heurísticas guían las decisiones de manera más inteligente con el tiempo24. • Implementación Eficiente de la Propagación Unitaria: La PU es una parte fundamental del proceso DPLL/CDCL y puede consumir una gran parte del tiempo de ejecución25. Técnicas como las Literales Vigías (Watched Literals)8... son esenciales para la eficiencia moderna14. Permiten que el solver evite escanear cada cláusula cada vez que se asigna una variable8..., reduciendo drásticamente la sobrecarga de PU y acelerando el proceso8.... • Estrategias de Reinicio (Restarts): Los solvers CDCL modernos a menudo reinician periódicamente la búsqueda19.... La razón es que un solver puede estancarse explorando una parte improductiva del espacio de búsqueda ("fenómeno de cola pesada"29). Los reinicios ayudan a escapar de estas regiones29.... Crucialmente, los solvers retienen las cláusulas aprendidas a través de los reinicios19..., lo que significa que el solver no pierde el conocimiento adquirido y comienza la nueva búsqueda con información adicional que puede guiarla de manera más efectiva19. • Gestión de la Base de Cláusulas Aprendidas: Aprender muchas cláusulas es beneficioso, pero la base de datos de cláusulas puede volverse muy grande, lo que ralentiza PU y consume mucha memoria31. Por lo tanto, se han desarrollado estrategias sofisticadas para gestionar y eliminar periódicamente las cláusulas aprendidas de baja calidad o redundantes11.... Métricas como la Distancia de Bloque Literal (LBD) ayudan a determinar qué cláusulas eliminar28.... Esto mantiene un equilibrio entre retener conocimiento útil y mantener la eficiencia de la búsqueda11.... En conclusión, si bien la complejidad teórica de SAT sigue siendo exponencial en el peor caso1..., las técnicas de CDCL —principalmente el aprendizaje de cláusulas y

el retroceso no cronológico—, combinadas con heurísticas sofisticadas de selección de variables (como VSIDS/DLIS) y optimizaciones en la propagación unitaria (como Literales Vigías), así como las estrategias de reinicio y gestión de cláusulas, transforman los solvers de herramientas de búsqueda básica a sistemas inteligentes que aprenden de sus errores34, podan eficientemente el espacio de búsqueda y escalan a problemas de gran magnitud20.... Estos son los factores que explican su éxito en aplicaciones prácticas1....

Capítulo 2

Propuesta

Capítulo 3

Detalles de Implementación y Experimentos

Conclusiones

Conclusiones

Recomendaciones

Recomendaciones