UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

LARA LETÍCIA DE MORAIS

IMPLEMENTAÇÃO DE UM SOLUCIONADOR SAT BASEADO NO ALGORITMO DPLL COM HEURÍSTICAS DE DECISÃO

TOLEDO

2025

LARA LETÍCIA DE MORAIS

IMPLEMENTAÇÃO DE UM SOLUCIONADOR SAT BASEADO NO ALGORITMO DPLL COM HEURÍSTICAS DE DECISÃO

Implementation of a SAT Solver Based on the DPLL Algorithm with Decision Heuristics

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia de Computação do Curso de Bacharelado em Engenharia de Computação da Universidade Tecnológica Federal do Paraná.
Orientador(a): Prof. Dr. Ricardo Tavares de Oliveira.

TOLEDO

2025

LARA LETÍCIA DE MORAIS

IMPLEMENTAÇÃO DE UM SOLUCIONADOR SAT BASEADO NO ALGORITMO DPLL COM HEURÍSTICAS DE DECISÃO

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia de Computação do Curso de Bacharelado em Engenharia de Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 26 / Junho / 2025

Daniel Cavalcanti Jeronymo
Doutor
Universidade Tecnológica Federal do Paraná - TD

Gustavo Henrique Paetzold
Doutor
Universidade Tecnológica Federal do Paraná - TD

Ricardo Tavares de Oliveira - **Orientador**
Doutor
Universidade Tecnológica Federal do Paraná - TD

TOLEDO

2025

AGRADECIMENTOS

Agradeço, em primeiro lugar, ao meu pai. Seu apoio incondicional, presente de todas as formas desde o primeiro dia desta jornada, foi fundamental para que eu chegasse até aqui. Sem ele, nada disto seria possível.

À minha querida amiga Rafaela Louise, minha companheira fiel ao longo de toda esta trajetória. Agradeço por cada momento compartilhado, pelas alegrias, desafios e pelo apoio mútuo que tornou o caminho mais leve. Sua amizade foi um presente.

A todos os meus professores, minha sincera gratidão. Cada um, à sua maneira, contribuiu para a minha formação, não apenas com seus ensinamentos, mas também com a dedicação que me influenciaram profundamente.

Ao meu orientador, Professor Dr. Ricardo Tavares de Oliveira, agradeço imensamente por ter aceitado o desafio de me guiar neste projeto. Sua presença constante, orientação precisa e sabedoria foram essenciais durante todo o desenvolvimento deste trabalho. Não poderia ter escolhido um orientador melhor.

Por último, mas de forma alguma menos importante, agradeço ao meu marido, Alvaro Barbosa. Que entrou em minha vida no meio desta caminhada e, desde então, tem sido meu maior incentivador. Agradeço por todo o apoio, pela imensa paciência nos momentos difíceis e por acreditar em mim. Celebrar a conclusão deste ciclo ao seu lado, enquanto você também encerra o seu, torna este momento infinitamente mais especial.

RESUMO

Este trabalho apresenta o desenvolvimento de um solucionador para o problema de satisfatibilidade booleana (SAT), baseado no algoritmo de *Davis-Putnam-Logemann-Loveland* (DPLL). A ferramenta implementada realiza desde a verificação da fórmula até sua conversão para a forma normal conjuntiva e o formato padrão de entrada DIMACS, além do algoritmo em si. Para análise comparativa, foram integradas duas heurísticas de decisão: MOM's, de natureza estática, e VSIDS, dinâmica e adaptativa. A avaliação experimental foi conduzida com instâncias de *benchmark* e fórmulas inseridas manualmente. Os resultados indicaram desempenho superior da heurística MOM's na maioria dos testes, enquanto a VSIDS teve vantagem apenas em um caso específico. Conclui-se que a eficácia de uma heurística está ligada ao equilíbrio entre sua complexidade teórica e a forma como é implementada na prática. Os resultados mostram que a vantagem teórica de uma heurística só traz bons resultados práticos quando acompanhada de uma implementação eficiente. Assim, além de disponibilizar uma ferramenta funcional, o estudo contribui para o entendimento prático da interação entre teoria e implementação no contexto de problemas NP-completos.

Palavras-chave: satisfatibilidade booleana; algoritmo DPLL; heurísticas de decisão; MOM's; VSIDS.

ABSTRACT

This work presents the development of a solver for the problem of Boolean satisfiability (SAT), based on the *Davis-Putnam-Logemann-Loveland* algorithm (DPLL). The implemented tool performs formula verification, conversion to conjunctive normal form, and to the DIMACS standard input format, as well as the actual solving process. For comparative analysis, two decision heuristics were integrated: MOM's, a static heuristic, and VSIDS, a dynamic and adaptive heuristic. The experimental evaluation was conducted using benchmark instances and manually inserted formulas. The results indicated superior performance of the MOM's heuristic in most tests, while VSIDS showed advantage only in a specific case. It is concluded that the effectiveness of a heuristic depends on the balance between its theoretical complexity and the way it is implemented in practice. The results show that the theoretical advantage of a heuristic only translates into good practical results when accompanied by an efficient implementation. Hence, besides providing a functional tool, the study contributes to the practical understanding of the interaction between theory and implementation in the context of NP-complete problems.

Keywords: boolean satisfiability; DPLL algorithm; decision heuristics; MOM's; VSIDS.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CDCL	Aprendizado de Cláusulas por Conflito, do inglês *Conflict-Driven Clause Learning*
CNF	Forma Normal Conjuntiva, do inglês *Conjunctive Normal Form*
DIMACS	Centro de Matemática Discreta e Ciência da Computação Teórica, do inglês *Center for Discrete Mathematics and Theoretical Computer Science*
DIMACS Challenge	Desafio organizado pelo *Center for Discrete Mathematics and Theoretical Computer Science* (DIMACS) para avaliar algoritmos em diversas áreas, incluindo problemas de Satisfatibilidade (SAT).
DPLL	Algoritmo Davis-Putnam-Logemann-Loveland
MiniSat	Solucionador de Problemas de Satisfatibilidade Booleana Minimalista, do inglês *Mnimalistic Boolean Satisfiability Problem Solver*
MOM's	Do inglês *Maximum Occurrences in clauses of Minimum Size*
PLE	Eliminação de Literal Puro, do inglês *Pure Literal Elimination*
SAT	Problema de Satisfatibilidade Booleana, do inglês *Boolean Satisfiability Problem*
SAT-solver	Solucionador de Problemas de Satisfatibilidade Booleana, do inglês *Boolean Satisfiability Problem Solver*
UP	Propagação Unitária, do inglês *Unit Propagation*
UPLA	Propagação Unitária *Look-Ahead*, do inglês *Unit Propagation Look-Ahead*
VSIDS	Do inglês *Variable State Independent Decaying Sum*

SUMÁRIO

1 INTRODUÇÃO

O problema de satisfatibilidade booleana (SAT) representa uma base fundamental na ciência da computação teórica e aplicada, com implicações que vão desde a verificação formal de hardware e software até aplicações em inteligência artificial e otimização combinatória. O SAT consiste em determinar se existe uma atribuição de valores (verdadeiro ou falso) para as variáveis de uma fórmula lógica de forma que a expressão seja verdadeira. Demonstrado como NP-completo por (Cook, 1971), o problema SAT é central na teoria da complexidade computacional, pois qualquer problema na classe NP pode ser reduzido a ele em tempo polinomial.

Nos últimos anos, os solucionadores SAT (SAT-solvers) evoluíram de maneira expressiva, possibilitando a resolução eficiente de instâncias com milhares de variáveis e cláusulas. O algoritmo Davis-Putnam-Logemann-Loveland (DPLL) (Davis; Logemann; Loveland, 1962) representa um ponto importante nesse avanço, servindo como fundamento para muitos solucionadores modernos. A escolha adequada de heurísticas de decisão é um fator determinante para a eficiência de algoritmos baseados no DPLL, pois orientam a escolha de variáveis durante o processo de busca, impactando diretamente o tempo de resolução.

Este trabalho tem como objetivo investigar o funcionamento de solucionadores SAT por meio da implementação de uma ferramenta completa. Essa ferramenta realiza desde a verificação e conversão de fórmulas para a Forma Normal Conjuntiva (CNF), utilizando a transformação de Tseitin (Tseitin, 1983) para garantir a eficiência, até a geração da representação no formato padrão DIMACS. O foco deste projeto é o desenvolvimento de um solucionador SAT baseado no algoritmo DPLL, que integra e permite a comparação de duas heurísticas de decisão distintas: a clássica MOM's, baseada na estrutura da fórmula, e a moderna e dinâmica VSIDS, que prioriza variáveis envolvidas em conflitos recentes.

1.1 Objetivos

1.1.1 Objetivo geral

O objetivo geral deste trabalho é estudar e implementar um solucionador para o problema de satisfatibilidade booleana (SAT) baseado no algoritmo DPLL. O projeto aborda desde as etapas de pré-processamento, como a conversão de fórmulas para a Forma Normal Conjuntiva (CNF) e a geração do formato DIMACS, até a implementação e avaliação comparativa das heurísticas de decisão MOM's e VSIDS.

1.1.2 Objetivos específicos

- Implementar um algoritmo para verificar se uma fórmula lógica está na CNF.

- Implementar um conversor de fórmulas lógicas para CNF utilizando a transformação de (Tseitin, 1983), garantindo que o tamanho da fórmula cresça linearmente.
- Desenvolver um módulo para converter fórmulas em CNF para o formato DIMACS.
- Estudar as técnicas de resolução do problema SAT, com ênfase no algoritmo DPLL e em suas otimizações, como propagação unitária e eliminação de literais puros.
- Implementar um solucionador SAT baseado no algoritmo DPLL, integrando as heurísticas de decisão MOM's e VSIDS.
- Avaliar e comparar o desempenho do solucionador com cada uma das heurísticas implementadas, por meio de instâncias de *benchmarks* padronizados e também de fórmulas inseridas manualmente, analisando métricas como o tempo de execução.

1.2 Justificativa

A relevância deste trabalho reside na importância teórica e prática do problema (SAT). Sendo o primeiro problema demonstrado ser NP-completo (Cook, 1971), o SAT é um problema central da teoria da complexidade e um modelo para inúmeros problemas do mundo real em áreas tão diversas quanto verificação de circuitos, planejamento em inteligência artificial e bioinformática. A implementação de um solucionador SAT a partir de seus princípios fundamentais oferece um entendimento aprofundado sobre os desafios e as técnicas que impulsionaram a eficiência desses algoritmos.

A escolha de implementar um solucionador baseado no algoritmo DPLL justifica-se por seu papel fundamental na história dos SAT-solver. Compreender o DPLL é essencial para entender a evolução para arquiteturas mais modernas, como a CDCL. Além disso, o foco na comparação entre as heurísticas MOM's e VSIDS aborda uma questão central no projeto de solucionadores: o impacto da estratégia de decisão. Este estudo permite uma análise prática e comparativa entre uma abordagem estática e estrutural (MOM's) e uma dinâmica e adaptativa (VSIDS), que é a base de solucionadores de ponta como o MiniSat.

Do ponto de vista acadêmico, o projeto integra e aplica conhecimentos de lógica, teoria dos grafos, algoritmos e complexidade computacional. A construção da ferramenta completa - desde a análise da fórmula de entrada com a transformação de Tseitin (Tseitin, 1983), passando pela padronização com o formato DIMACS, até a resolução do problema — consolida o aprendizado teórico e demonstra a aplicação de métodos computacionais para a solução de um problema notoriamente difícil.

1.3 Estrutura do trabalho

Este trabalho está organizado da seguinte forma:

- **Capítulo 1: Introdução** Apresenta o problema SAT, sua relevância na ciência da computação, a justificativa e os objetivos do trabalho.

- **Capítulo 2: Referencial Teórico** Discute os conceitos fundamentais para a compreensão do projeto. Este capítulo está estruturado nos seguintes tópicos:

  - **Lógica Proposicional e Álgebra Booleana:** Introduz os conceitos básicos para a formulação do problema.
  - **Forma Normal Conjuntiva (CNF):** Explica a representação padrão das fórmulas e detalha a **Transformação de Tseitin**.
  - **O Problema SAT e o Algoritmo DPLL:** Descreve o problema de satisfatibilidade, o funcionamento do algoritmo DPLL e suas técnicas de simplificação, como Propagação Unitária e Eliminação de Literal Puro.
  - **Heurísticas de Decisão:** Apresenta em detalhes as heurísticas **MOM's** e **VSIDS**, que são o foco da implementação.
  - **Formato DIMACS:** Explica o padrão utilizado para a entrada de dados no solucionador.

- **Capítulo 3: Materiais e Métodos** Descreve os algoritmos, as estruturas de dados e os procedimentos adotados para implementar a ferramenta, incluindo o verificador de CNF, o conversor de Tseitin, o gerador DIMACS e, principalmente, o solucionador DPLL com os módulos das heurísticas MOM's e VSIDS.

- **Capítulo 4: Resultados e Discussão** Apresenta a análise dos experimentos realizados com o solucionador, detalhando o desempenho das heurísticas MOM's e VSIDS em diferentes instâncias, além de discutir as implicações dos resultados obtidos.

- **Capítulo 5: Conclusão** Reúne as principais realizações do trabalho, enfatizando a criação da ferramenta, a avaliação das heurísticas adotadas e o entendimento adquirido sobre a influência da implementação na efetividade teórica. Apresenta ainda as restrições identificadas e propõe direções para pesquisas futuras.

2 REFERENCIAL TEÓRICO

Este capítulo explora os fundamentos teóricos do problema de satisfatibilidade booleana (SAT). Para contextualizar o problema, são introduzidos conceitos de lógica proposicional, álgebra booleana e a representação em forma normal conjuntiva (CNF), além da transformação de Tseitin (Tseitin, 1983). Em seguida, o capítulo foca nas estratégias de resolução, detalhando o funcionamento de solucionadores (SAT-solver) baseados no algoritmo DPLL, suas otimizações e as heurísticas de decisão que o tornam eficiente. Por fim, é apresentado o formato DIMACS, utilizado para a entrada de dados.

2.1 Lógica proposicional

A lógica proposicional trata das propriedades e relações de proposições declarativas, as quais podem ser verdadeiras ou falsas, mas nunca ambas simultaneamente. Essa característica binária possibilita a análise lógica de proposições simples e compostas, sendo as últimas formadas pela combinação de proposições simples utilizando conectivos lógicos. Esses conectivos – como a negação $(\neg)$, conjunção $(\wedge)$, disjunção $(\vee)$, implicação $(\Rightarrow)$ e bicondicional $(\Leftrightarrow)$ – desempenham papel fundamental na construção de sentenças mais complexas e na representação de ideias abstratas (Klement, 2004).

Segundo (Ribas, 2015), proposições são expressas por fórmulas bem formadas, que seguem regras específicas de construção. Variáveis proposicionais, representadas por símbolos como p, q e r, assumem valores de verdade V (verdadeiro) ou F (falso). Fórmulas compostas, como $(p \wedge q)$, são criadas recursivamente a partir de proposições e conectivos lógicos. Além disso, conceitos como cláusulas, literais e satisfatibilidade são fundamentais para resolver problemas lógicos, sendo amplamente aplicados em algoritmos e métodos computacionais.

Conforme descrito por (Bochenski, 1961), Aristóteles estabeleceu os primeiros fundamentos da lógica formal para analisar a relação entre premissas e conclusões, independentemente do conteúdo específico das proposições. Embora Aristóteles não utilizasse o termo "lógica" como é compreendido hoje, suas obras forneceram as bases para o estudo sistemático das inferências. Ele diferenciava os silogismos em demonstrativos, quando as premissas são verdadeiras, e dialéticos, quando baseados em opiniões geralmente aceitas.

A lógica proposicional, portanto, pode ser entendida como um sistema formal que permite manipular proposições de maneira estruturada e sistemática. Sua relevância supera a filosofia e a matemática, encontrando aplicações em áreas como ciência da computação e engenharia. Problemas como satisfatibilidade e resolução de fórmulas destacam sua importância prática e teórica nos contextos atuais (Ribas, 2015).

2.2 Álgebra booleana

Complementando a estrutura da lógica proposicional, a Álgebra Booleana oferece o formalismo matemático para manipular seus valores de verdade. Introduzida por (Boole, 1854) em seu trabalho *An Investigation of the Laws of Thought*, ele propôs que as operações lógicas poderiam ser tratadas de maneira algébrica, conectando diretamente conceitos de lógica e matemática. A sua estrutura se fundamenta em princípios que possibilitam a representação e manipulação de informações de forma exata e eficaz.

Enquanto a lógica proposicional frequentemente utiliza os símbolos V e F, no contexto da álgebra booleana e da computação, é padrão representar verdadeiro com o valor 1 e falso com 0. Esses valores descrevem o comportamento de variáveis lógicas, que assumem exclusivamente um desses dois estados. Essa notação será adotada a partir deste ponto, especialmente nas tabelas-verdade e na análise de algoritmos.

Além disso, a Álgebra Booleana permite a aplicação de operações lógicas sobre esses valores. Sendo eles:

- Operação **e** (multiplicação lógica ou conjunção - $\wedge$): resulta em 0 se pelo menos uma das variáveis de entrada tiver o valor 0. Por definição, o resultado dessa operação será verdadeiro (1) apenas quando todas as variáveis de entrada forem verdadeiras simultaneamente (Paiva, 2017).

Tabela 1: Tabela verdade da operação E

p	**q**	**p $\wedge$ q**
0	0	0
0	1	0
1	0	0
1	1	1

Fonte: Autoria própria (2025).

Na Tabela 1, é possível observar o resultado da operação p $\wedge$ q. De maneira análoga, pode-se determinar o resultado de expressões envolvendo mais variáveis, como p $\wedge$ q $\wedge$ r. Além disso, as propriedades associativa e comutativa aplicam-se à operação 'E', permitindo que ela seja avaliada em pares e em qualquer ordem, sem alterar o resultado final.

- Operação **ou** (adição lógica ou disjunção - $\vee$): resulta em 1 sempre que pelo menos uma das variáveis de entrada for igual a 1. Como cada variável no contexto da Álgebra Booleana pode assumir apenas os valores 1 ou 0, o resultado da operação também será um desses valores. Para determinar quando a operação é verdadeira (1), basta analisar os casos em que ao menos uma variável de entrada assume o valor 1 (Paiva, 2017).

Tabela 2: Tabela verdade da operação OU

p	**q**	**p** $\vee$ **q**
0	0	0
0	1	1
1	0	1
1	1	1

Fonte: Autoria própria (2025).

Na Tabela 2, estão apresentados os resultados da operação $p \vee q$, considerando duas variáveis. Para expressões envolvendo mais variáveis, como $p \vee q \vee r$, o resultado pode ser obtido de forma similar, avaliando-se pares de variáveis. Por exemplo, é possível calcular o valor de $p \vee q$ e, em seguida, aplicar a operação com r. Essa abordagem é válida devido à propriedade associativa. Além disso, a propriedade comutativa garante que a ordem de avaliação das variáveis não altera o resultado.

- Operação **negação** (complementação lógica - $\neg$): produz como resultado o valor complementar da variável de entrada. Diferentemente das operações 'E' e 'OU', que são binárias, a operação de negação atua sobre uma única variável ou subfórmula, sendo, portanto, um operador unitário (Paiva, 2017).

Tabela 3: Tabela verdade da operação negação

p	$\neg$ **p**
0	1
1	0

Fonte: Autoria própria (2025).

Na Tabela 3, é possível observar os resultados da negação aplicada a variáveis lógicas. O funcionamento simples e direto dessa operação faz dela um elemento fundamental na construção de expressões booleanas mais complexas.

- Operação **implicação** ($\Rightarrow$): Uma implicação $p \Rightarrow q$ é equivalente a $\neg p \vee q$.

Tabela 4: Tabela verdade da operação implicação

p	**q**	**p** $\Rightarrow$ **q**
0	0	1
0	1	1
1	0	0
1	1	1

Fonte: Autoria própria (2025).

A Tabela 4 apresenta a tabela verdade da operação implicação. Como mencionado anteriormente, a equivalência $p \Rightarrow q \equiv \neg p \vee q$ pode ser usada para reescrever implicações em expressões booleanas.

- Operação **bicondicional** ($\Leftrightarrow$): O bicondicional $p \Leftrightarrow q$ é verdadeiro quando p e q possuem o mesmo valor lógico.

Tabela 5: Tabela verdade da operação bicondicional

p	**q**	**p $\Leftrightarrow$ q**
0	0	1
0	1	0
1	0	0
1	1	1

Fonte: Autoria própria (2025).

A Tabela 5 apresenta a tabela verdade da operação bicondicional, que pode ser reescrita como $(p \Rightarrow q) \wedge (q \Rightarrow p)$.

2.3 Forma Normal Conjuntiva - CNF

A Forma Normal Conjuntiva (CNF) é uma representação padrão utilizada na lógica proposicional, amplamente aplicada em problemas de satisfatibilidade (SAT) por reduzir a fórmula a uma estrutura padronizada e simplificada, tornando-a mais adequada para manipulação algorítmica.

Uma expressão está em CNF se for uma conjunção ($\wedge$) de cláusulas, onde cada cláusula é uma disjunção ($\vee$) de literais. Um literal é uma proposição simples (p) ou sua negação ($\neg p$). Se uma cláusula contém um único literal, ele deve assumir o valor verdadeiro. Abaixo, um exemplo de uma expressão na CNF :

$$(p \vee \neg q) \wedge (q \vee r) \wedge (\neg p \vee \neg r)$$

- $(p \vee \neg q)$ é uma cláusula composta pelos literais p e $\neg q$ conectados por um "ou"lógico ($\vee$).
- $(q \vee r)$ é uma cláusula composta pelos literais q e r conectados por um "ou"lógico ($\vee$).
- $(\neg p \vee \neg r)$ é uma cláusula composta pelos literais $\neg p$ e $\neg r$ conectados por um "ou"lógico ($\vee$).

Todas as cláusulas estão conectadas por um e lógico ($\wedge$), formando uma expressão na CNF.

Existem várias formas de tranformar uma expressão na CNF . Esse trabalho irá abordar a conversão por transformação de Tseitin, por ser uma transformação linear e eficiente, conforme descrito na próxima seção.

2.3.1 Tranformação de Tseitin

A transformação de Tseitin, introduzida por (Tseitin, 1983), é uma técnica eficaz para converter fórmulas lógicas gerais para (CNF). A abordagem de Tseitin é particularmente valiosa por evitar o crescimento exponencial no tamanho das fórmulas que frequentemente ocorre em transformações distributivas diretas, que podem resultar em um aumento exponencial, afetando a memória utilizada e no tempo necessário para resolver problemas.

Seu uso garante que a fórmula resultante seja satisfatível se, e somente se, a fórmula original também for, além de crescer linearmente em tamanho, o que a torna prática para aplicação em grandes problemas combinatórios (Tseitin, 1983). A noção de satisfabilidade será apresentada na próxima seção.

Para cada conectivo lógico, uma nova variável representando o resultado do conectivo é introduzida e substitui a subfórmula correspondente. Essas novas variáveis simplificam a fórmula ao mesmo tempo em que preservam sua satisfatibilidade, pois são ligadas às subfórmulas originais por meio de equivalências que podem ser traduzidas para CNF de forma independente (Meert; Vlasselaer; Broeck, 2016).

Considere a seguinte expressão lógica:

$$(p \vee (q \wedge r)) \Rightarrow \neg s$$

A expressão acima não está no formato CNF, mas será transformada aplicando os passos de Tseitin.

- **Passo 1:** reescrever a expressão como equivalente lógico. Uma implicação pode ser reescrita como:

  $$a \Rightarrow b = \neg a \vee b$$

  Portanto a expressão se torna:

  $$\neg(p \vee (q \wedge r)) \vee \neg s$$

- **Passo 2:** utilizar variáveis auxiliares para representar subfórmulas. Para isso, a expressão deve ser dividida em subexpressões.

  Introduzir a para representar $q \wedge r$:

  $$a \Leftrightarrow (q \wedge r)$$

Introduzir b para representar $p \vee a$:

$$b \Leftrightarrow (p \vee a)$$

Finalmente, a expressão se torna:

$$\neg b \vee \neg r$$

- **Passo 3:** reescrever as equivalências ($\Leftrightarrow$) como conjunções de implicações ($\Rightarrow$).

  Para $a \Leftrightarrow (q \wedge r)$:

  $$(a \Rightarrow (q \wedge r)) \wedge ((q \wedge r) \Rightarrow a)$$

  O que é equivalente a:

  $$(\neg a \vee q) \wedge (\neg a \vee r) \wedge (\neg q \vee \neg r \vee a)$$

  Para $b \Leftrightarrow (p \vee a)$:

  $$(b \Rightarrow (p \vee a)) \wedge ((p \vee a) \Rightarrow b)$$

  O equivalente a:

  $$(\neg b \vee p \vee a) \wedge (\neg p \vee b) \wedge (\neg a \vee b)$$

- **Passo 4:** combinar todas as cláusulas geradas.

  Clásulas de $a \Leftrightarrow (q \wedge r)$:

  $$(\neg a \vee q), (\neg a \vee r), (\neg q \vee \neg r \vee a)$$

  Cláusulas de $b \Rightarrow (p \vee a)$

  $$(\neg b \vee p \vee a), (\neg p \vee b), (\neg a \vee b)$$

  Cláusula principal:

  $$(\neg b \vee \neg s)$$

- **Passo 5:** fórmula final na CNF.

$$(\neg a \vee q) \wedge (\neg a \vee r) \wedge (\neg q \vee \neg r \vee a) \wedge (\neg b \vee p \vee a) \wedge (\neg p \vee b) \wedge (\neg a \vee b) \wedge (\neg b \vee \neg s)$$

Seguindo os passos acima, é possível transformar qualquer expressão inicial para CNF sem alterar sua satisfatibilidade.

2.4 SAT-solver

O problema de satisfatibilidade (SAT) é um problema clássico e fundamental da ciência da computação teórica devido à sua capacidade de resolver desafios computacionais. Em termos gerais, o problema de SAT consiste em determinar se existe uma atribuição de valores de verdade (verdadeiro ou falso) para as variáveis de uma fórmula lógica que a torne verdadeira. Esse problema é aplicável em áreas como otimização combinatória, verificação formal de sistemas, inteligência artificial e teoria dos grafos (Klimek, 2018).

Para explicar o conceito, considere os exemplos de expressões lógicas na (CNF):

- **Expressão satisfatível:**

$$(p \vee \neg q) \wedge (\neg p \vee q) \wedge (q \vee r)$$

  Essa expressão é satisfatível porque existe pelo menos uma atribuição de valores verdade para as variáveis p, q e r que a torna verdadeira. A atribuição p = verdadeiro, q = verdadeiro e r = verdadeiro, satisfaz todas as cláusulas.

- **Expressão insatisfatível:**

$$(p \vee q) \wedge (\neg p \vee q) \wedge (p \vee \neg q) \wedge (\neg p \vee \neg q)$$

  Essa expressão é insatisfatível porque não existe nenhuma atribuição de valores verdade para p e q que satisfaça todas as cláusulas simultaneamente. Qualquer tentativa de atribuição resulta em uma contradição, tornando a expressão insatisfatível.

(Cook, 1971), em seu trabalho *The Complexity of Theorem Proving Procedures*, demonstra que o problema de satisfatibilidade booleana (SAT) é NP-Completo. Ele define a classe NP como o conjunto de problemas de decisão cujas soluções podem ser verificadas em tempo polinomial, e prova que qualquer problema em NP pode ser reduzido ao SAT em tempo polinomial. Essa redução é feita por meio da simulação do comportamento de uma Máquina de Turing não determinística, que resolve problemas em NP, utilizando uma fórmula lógica. A partir dessa construção, Cook estabelece que, se existir um algoritmo eficiente (em tempo polinomial) para resolver o SAT, então todos os problemas em NP também poderão ser resolvidos eficientemente. Esse resultado coloca o SAT em uma posição central na teoria da complexidade computacional, sendo o primeiro problema provado como NP-Completo. Não se sabe se é possível construir uma algoritmo polinomial para SAT, sendo esta uma questão ainda em aberto na Computação.

Para resolver instâncias do problema de SAT, utilizam-se algoritmos chamados *SAT-solvers*. Um SAT-solver é um algoritmo desenvolvido para determinar a satisfatibilidade de fórmulas booleanas. Nos últimos anos, os *SAT-solvers* têm evoluído significativamente, permitindo a solução prática de instâncias com milhares de variáveis e cláusulas.

Historicamente, um dos primeiros algoritmos fundamentais para resolver o problema de SAT foi o algoritmo de Davis-Putnam-Logemann-Loveland (DPLL), desenvolvido na década de 1960. Esse algoritmo, descrito por (Davis; Logemann; Loveland, 1962), utiliza um método de busca baseado em *backtracking*, uma técnica de busca que explora soluções recursivamente, desfazendo escolhas quando um caminho não leva à solução, e serve como base para a maioria dos *SAT-solvers* modernos. Entre as principais técnicas introduzidas pelo DPLL estão a propagação unitária, que permite simplificar a fórmula ao atribuir valores a variáveis determinadas unicamente, e a heurística para escolha de variáveis. Além disso, a técnica de literais vigiados, mais tarde explorada por pesquisas como as de (Eén; Sörensson, 2003), contribui para otimizar a verificação de cláusulas durante a execução do algoritmo.

2.5 O Algoritmo DPLL

O algoritmo Davis-Putnam-Logemann-Loveland (DPLL) representa um avanço fundamental na criação de SAT-solver. Mesmo tendo sido introduzido há mais de cinco décadas, o DPLL continua sendo a base de grande parte dos *SAT-solvers* (Fredrikson; Platzer, 2023).

(Nieuwenhuis; Oliveras; Tinelli, 2005) observam que a maioria dos *SAT-solvers* avançados atuais utilizam variações do procedimento DPLL para determinar a satisfatibilidade de fórmulas proposicionais na CNF . Proposto originalmente por (Davis; Putnam, 1960) e consolidado por (Davis; Logemann; Loveland, 1962), o DPLL é um algoritmo completo, que utiliza uma técnica de busca com retrocesso (*backtracking*) para determinar se uma dada fórmula proposicional na (CNF) é satisfatível. A eficiência do DPLL vem da sua abordagem estruturada para explorar o espaço de possíveis atribuições de verdade, combinada com o uso de regras de inferência eficazes. Essas regras, como a propagação unitária e a eliminação de literais puros, reduzem significativamente o número de combinações possíveis a serem consideradas. O algoritmo opera sobre uma fórmula em CNF de maneira recursiva e, a cada passo, busca simplificar a fórmula. Quando necessário, seleciona uma variável para atribuir um valor de verdade, analisando as implicações dessa escolha.

O processo segue uma série de etapas que envolvem simplificação da fórmula, tomada de decisões e retrocesso quando necessário.

- **Simplificação da Fórmula:**

  Inicialmente, o DPLL tenta reduzir a complexidade da fórmula por meio da aplicação de regras determinísticas. Entre essas regras estão a propagação unitária, que permite inferir valores de variáveis a partir de cláusulas unitárias, e a eliminação de literais puros, que remove variáveis cuja ocorrência na fórmula não gera conflito (Fredrikson; Platzer, 2023; Nieuwenhuis; Oliveras; Tinelli, 2005). Ambas as técnicas contribuem significativamente para diminuir o espaço de busca e serão abordadas com mais detalhes nas seções seguintes.

- **Verificação de Término:**

Após a simplificação, o algoritmo verifica se a fórmula foi completamente resolvida. Se todas as cláusulas foram satisfeitas, considera-se que a fórmula é satisfatível e uma atribuição válida foi encontrada. Por outro lado, se alguma cláusula se torna vazia — ou seja, todos os seus literais foram atribuídos como falsos — ocorre uma contradição, indicando que a atribuição atual é inválida e deve ser revista (Fredrikson; Platzer, 2023; Davis; Logemann; Loveland, 1962).

- **Escolha do Literal:**

  Se a fórmula não foi completamente resolvida, o algoritmo precisa fazer uma escolha para prosseguir com a busca. Uma variável ainda não atribuída é selecionada, frequentemente com base em alguma heurística, e um valor de verdade é atribuído a ela. O algoritmo então prossegue recursivamente com esta nova atribuição, retornando ao passo de simplificação (Fredrikson; Platzer, 2023; Gong; Zhou, 2017).

- **Retrocesso (*backtracking*):**

  Se uma escolha e suas consequências levam a uma cláusula vazia, o algoritmo realiza um retrocesso, desfazendo a última decisão e tentando a atribuição oposta para a variável. Se ambas as atribuições (Verdadeira e Falsa) resultarem em contradições, conclui-se que a subfórmula, sob as decisões anteriores, é insatisfatível. Nesse caso, o algoritmo retrocede mais um nível na árvore de busca (Davis; Logemann; Loveland, 1962; Nieuwenhuis; Oliveras; Tinelli, 2005).

Esse processo iterativo de tentativa, simplificação, detecção de conflitos e retrocesso compõe o núcleo do DPLL. A abordagem de *backtracking* adotada pelo algoritmo permite explorar de forma inteligente o espaço de soluções, percorrendo todas as alternativas válidas ao mesmo tempo em que evita redundâncias (Fredrikson; Platzer, 2023).

2.5.1 Eliminação de Literal Puro

A Eliminação de Literal Puro (PLE) é uma técnica clássica de simplificação utilizada em algoritmos para o problema SAT. Seu objetivo é identificar literais que aparecem com apenas uma polaridade em toda a fórmula — ou seja, que não possuem seu complemento — e atribuir valores de forma a satisfazer automaticamente todas as cláusulas em que ocorrem. Isso permite a remoção dessas cláusulas, reduzindo o espaço de busca sem comprometer a satisfatibilidade da fórmula original.

No contexto de fórmulas na CNF, um literal a é considerado puro quando seu complemento $\neg a$ não aparece em nenhuma cláusula. Como discutido por (Johannsen, 2005), tais literais podem ser atribuídos como verdadeiros (ou seus complementos como falso), permitindo a eliminação das cláusulas em que aparecem.

Uma característica importante da PLE é sua aplicação iterativa: a remoção de cláusulas pode tornar outros literais puros, permitindo novas simplificações. O processo se repete até que

não haja mais literais puros na fórmula, resultando em uma versão logicamente equivalente, porém simplificada (Johannsen, 2005; Kwon; Park; Ryu, 2008).

Essa técnica é frequentemente aplicada como uma etapa inicial no algoritmo DPLL, contribuindo para a redução do espaço de busca antes da escolha de variáveis e propagação de implicações.

2.5.2 Propagação Unitária

A Propagação Unitária (UP) é uma das técnicas fundamentais utilizadas em algoritmos para o problema (SAT). Seu objetivo é identificar cláusulas unitárias — aquelas compostas por um único literal não atribuído — e deduzir automaticamente o valor que esse literal deve assumir para satisfazer a cláusula. Esse processo permite simplificar a fórmula, reduzindo tanto o número de decisões quanto o tamanho do espaço de busca.

No contexto de fórmulas na CNF, uma cláusula se torna unitária quando todos os seus literais, exceto um, foram atribuídos como falsos. Por exemplo, em uma cláusula como (a), se os demais literais já foram eliminados ou atribuídos como falsos, a única forma de satisfazê-la é atribuindo o valor **verdadeiro** ao literal restante — neste caso, à variável a. Essa atribuição forçada, conhecida como propagação unitária, não apenas satisfaz a cláusula original, mas também simplifica a fórmula: cláusulas contendo a são satisfeitas e podem ser removidas, enquanto $\neg a$ pode ser descartado das cláusulas em que aparece. Como discutido por (Zhang; Stickely, 1996; Biere; Heule; Maaren, 2009; Marques-silva; Lynce; Malik, 2009), esse processo é aplicado iterativamente, gerando novas cláusulas unitárias sempre que possível, até que nenhuma nova inferência seja realizada, ou até que uma contradição, como uma cláusula vazia, seja encontrada.

A propagação unitária é aplicada de forma recorrente ao longo da execução do algoritmo DPLL. Estudos como os de (Anbulagan, 2004) e (Le Berre, 2001) ressaltam que a implementação eficiente da propagação unitária é crucial para o desempenho geral dos *SAT-solvers*. Além disso, essa técnica é a base para métodos mais avançados, como o *Unit Propagation Look-Ahead* (UPLA), que analisa os efeitos de atribuições provisórias em variáveis ainda indefinidas. Ao antecipar as consequências dessas atribuições, o UPLA torna as decisões de ramificação mais precisas e ainda permite detectar conflitos de forma antecipada.

Essa técnica é considerada fundamental para o sucesso dos algoritmos baseados em DPLL, servindo também como base para solucionadores modernos que incorporam aprendizado de cláusulas e outras estratégias de inferência. Sua aplicação contribui significativamente para a redução da complexidade das instâncias e para a diminuição do tempo necessário para encontrar uma solução.

Desde sua formulação original, o algoritmo DPLL serviu como base para o desenvolvimento de *SAT-solvers* modernos. Com o tempo, foram incorporadas técnicas adicionais como aprendizado de cláusulas e retrocesso não cronológico, o que levou o surgimento da abordagem conhecida como aprendizado dirigido por conflito (CDCL). Essa evolução deu origem a

solucionadores altamente eficientes, como o **MiniSat**, que tornou-se uma referência devido à sua simplicidade e performance, utilizando heurísticas como VSIDS para seleção de variáveis (Eén; Sörensson, 2004). Derivado dele, o **Glucose** aprimora o processo de aprendizado por conflito utilizando a métrica LBD (Literal Block Distance), priorizando cláusulas mais úteis para decisões futuras (Audemard; Simon, 2009). Já o **MapleSAT** introduz novas estratégias de reinicialização e aprendizado adaptativo, com o objetivo de equilibrar a exploração do espaço de busca (Liang *et al.*, 2016). Esses solucionadores demonstram como os princípios fundamentais do DPLL continuam a orientar o desenvolvimento de técnicas modernas na resolução do problema de satisfatibilidade.

A eficiência desses algoritmos, no entanto, depende crucialmente da forma como as variáveis são escolhidas durante a busca, uma tarefa feita pelas heurísticas de decisão.

2.6 Heurísticas de Decisão

Durante a execução do algoritmo DPLL, uma das etapas mais importantes é a escolha de qual variável será atribuída em cada ponto da busca. Essa escolha pode acelerar significativamente a resolução do problema. Para isso, são empregadas heurísticas de decisão, que analisam características da fórmula atual com o objetivo de selecionar variáveis que maximizem o potencial de simplificação ou detecção de conflitos. Nesta seção, serão abordadas duas heurísticas implementadas neste trabalho: a heurística MOM's de natureza estrutural, e a VSIDS, de natureza dinâmica e amplamente utilizada em solucionadores modernos.

2.6.1 Heurística MOM's

A heurística MOM's, introduzida por (Freeman, 1995), é uma estratégia clássica de seleção de variáveis utilizada em algoritmos baseados no DPLL. A MOM's foi desenvolvida com o objetivo de tornar o processo de decisão mais eficiente, priorizando variáveis que ocorrem com maior frequência em cláusulas pequenas da fórmula — geralmente aquelas com exatamente dois ou três literais. Essa abordagem parte do princípio de que variáveis presentes nessa clásulas estão mais propensas a gerar conflitos, favorecendo a simplificação da fórmula e acelerando o processo de propagação (Lynce, 2004).

O cálculo da pontuação de cada variável segue a seguinte expressão:

$$\text{MOM}(x) = (n_x + 1) \times (n_{\neg x} + 1),$$

em que n_x representa o número de ocorrências do literal x nas cláusulas de menor tamanho e $n_{\neg x}$ o número de ocorrências de seu complemento $\neg x$ nas mesmas cláusulas. O incremento de 1 em cada termo serve para evitar produtos nulos e para favorecer variáveis com presença balanceada entre formas positiva e negativa (Freeman, 1995; Li; Anbulagan, 1997).

A eficácia de heurísticas baseadas na análise estrutural da fórmula, como a MOM's, é frequentemente comparada a abordagens mais avançadas de *look-ahead*. Um exemplo é o solucionador **Satz**, proposto por (Li; Anbulagan, 1997), que não utiliza a MOM's, mas adota uma estratégia de propagação unitária antecipada, avaliando o impacto de atribuições provisórias antes de tomar uma decisão. Embora essa técnica envolva um custo computacional mais elevado, ela permite uma exploração mais profunda da fórmula, sendo mais eficaz em reduzir o espaço de busca por meio de inferência local.

Heurísticas como a MOM's, como apontado por (Dequen; Dubois, 2006), contribuem para a redução da árvore de busca ao priorizar decisões com maior impacto estrutural na fórmula. Embora heurísticas mais dinâmicas, como a VSIDS — abordada na próxima seção — tenham se tornado padrão em solucionadores modernos, a MOM's ainda é relevante tanto como base teórica quanto como na prática, em estágios iniciais da resolução ou em contextos com restrições computacionais. Sua influência permanece visível nos princípios de heurísticas derivadas e em solucionadores que combinam abordagens estruturais com técnicas de inferência.

Estudos, como o de (Biere; Heule; Maaren, 2009) reforçam a importância dessa e de outras heurísticas estruturais no contexto de solucionadores baseados em DPLL, especialmente quando associadas a técnicas de simplificação como propagação unitária.

2.6.2 Heurística VSIDS

A heurística VSIDS representa uma das abordagens mais influentes e eficazes na seleção de variáveis em solucionadores baseados em CDCL. Apesar de a VSIDS ser tradicionalmente associada a esses solucionadores, sua aplicação pode ser estendida a implementações baseadas no DPLL clássico, fornecendo uma forma dinâmica e eficiente de guiar a escolha de variáveis mesmo na ausência do aprendizado de cláusulas.

Introduzida inicialmente por (Moskewicz *et al.*, 2001) e amplamente adotada por solucionadores modernos como o MiniSat, a VSIDS baseia-se na atribuição de uma pontuação (*score*) para cada variável, a qual é atualizada dinamicamente ao longo do processo de resolução. Essa pontuação reflete a relevância recente da variável com base na sua participação em conflitos.

O funcionamento da VSIDS pode ser descrito em quatro etapas principais que interagem para orientar a seleção de variáveis:

- **Inicialização:** As pontuações de atividade de todas as variáveis são inicializadas com o valor zero. Isso implica que as primeiras decisões realizadas pelo solucionador são essencialmente aleatórias. Essa limitação motivou pesquisas que propõem estratégias alternativas de inicialização com base em conhecimento do domínio ou na estrutura da fórmula (Lenssen; Lonsing; Biere, 2023; Shacham; Zarpas, 2003).

- **Incremento por Conflito (*bumping*):** Quando um conflito é identificado, as variáveis presentes na cláusula de conflito têm suas pontuações aumentadas. Esse incremento favorece a seleção futura dessas variáveis, por estarem diretamente associadas às

causas dos conflitos recentes (Du; Wang, 2017; Shacham; Zarpas, 2003; Lenssen; Lonsing; Biere, 2023).

- **Atenuação Periódica (*decay*):** Após um número definido de conflitos, todas as pontuações são multiplicadas por um fator de decaimento *decay*, onde $0 < \textit{decay} < 1$. Essa operação reduz progressivamente a influência de variáveis que participaram de conflitos antigos, permitindo que o foco da busca se adapte dinamicamente à atividade recente (Du; Wang, 2017; Shacham; Zarpas, 2003).

- **Seleção:** A cada nova decisão, a heurística escolhe a variável ainda não valorada com maior pontuação de atividade no momento. Essa escolha guia o solucionador em direção a áreas da fórmula com maior concentração de conflitos, onde há maior potencial para novas deduções (Du; Wang, 2017; Shacham; Zarpas, 2003; Lenssen; Lonsing; Biere, 2023).

As regras de atualização podem ser formalizadas da seguinte maneira: seja *score*(v) a pontuação da variável v, inc o valor do incremento aplicado após conflitos, e *decay* o fator de atenuação:

- Após um conflito, para cada variável v na cláusula de conflito:

$$\textit{score}(v) \leftarrow \textit{score}(v) + \text{inc}$$

- Periodicamente, para todas as variáveis v:

$$\textit{score}(v) \leftarrow \textit{score}(v) \times \textit{decay}$$

A combinação entre incrementos localizados e atenuações periódicas torna a VSIDS uma heurística altamente responsiva às dinâmicas do processo de resolução. Conforme argumentado por (Lynce, 2004), uma das principais vantagens dessa abordagem está em seu baixo custo computacional, já que as atualizações de pontuação são independentes do estado atual das variáveis e não exigem reavaliações complexas da estrutura da fórmula.

A seleção da heurística de decisão é um elemento fundamental no desempenho de SAT-solver, influenciando diretamente a forma como o espaço de busca é explorado. A comparação entre a abordagem estrutural da MOM's e a estratégia dinâmica da VSIDS mostra diferentes formas de abordagens na estrutura do problema. Enquanto a MOM's atua de maneira estática, priorizando padrões fixos da fórmula, a VSIDS responde de forma adaptativa à evolução dos conflitos durante a resolução. A implementação dessas duas heurísticas neste trabalho permite uma análise prática do comportamentos e da forma como cada decisão afeta a eficiência do desempenho do solucionador.

2.6.3 Aprendizado dirigido por Conflito (CDCL)

Com o avanço dos *SAT-solvers*, novas abordagens mais eficientes foram desenvolvidas, sendo uma das mais notáveis o aprendizado dirigido por conflito (CDCL). Segundo (Eén; Mishchenko; Sörensson, 2007), o CDCL expande as capacidades do DPLL ao introduzir um mecanismo de aprendizado, permitindo a dedução de novas cláusulas a partir dos conflitos encontrados durante a busca. Isso reduz significativamente o espaço de busca, tornando os *SAT-solvers* muito mais eficientes na resolução de instâncias grandes e complexas.

Além disso, *SAT-solvers* adotam o formato padrão de entrada de dados conhecido como DIMACS. Esse formato padroniza a representação de fórmulas lógicas na CNF, tornando mais acessível a análise e o processamento de dados de entrada (Klimek, 2018).

2.6.4 DIMACS

O formato DIMACS é um padrão amplamente utilizado para representar problemas de satisfatibilidade booleana (SAT) em um formato legível por máquina. Ele foi desenvolvido como parte do DIMACS Challenge, uma iniciativa que visa promover a pesquisa e o desenvolvimento de algoritmos eficientes para problemas NP-Difíceis, incluindo o problema SAT (Cockburn *et al.*, 2006).

Ele consiste em um arquivo de texto que descreve uma fórmula booleana na CNF e é composto por duas partes principais: o cabeçalho e as cláusulas. O cabeçalho começa com a linha p cnf, seguida pelo número de variáveis e o número de cláusulas na fórmula. Por exemplo, `p cnf 5 3` indica que a fórmula possui `5` variáveis e `3` cláusulas. Após o cabeçalho, as cláusulas são listadas, onde cada número inteiro representa um literal (positivo para variáveis e negativo para suas negações), e o número `0` marca o término de cada cláusula. Como exemplo, considere o arquivo:

```
p cnf 5 3
1 -2 0
3 0
-4 -5 0
```

Este arquivo indica a fórmula:

$$(p \vee \neg q) \wedge (r) \wedge (\neg s \vee \neg t)$$

Esse formato foi projetado para ser simples e eficiente, permitindo que *SAT-solvers* leiam e processem rapidamente instâncias de problemas. Além disso, o formato é flexível o suficiente para representar problemas de grande escala, tornando-o uma escolha popular. O MiniSat, por exemplo, lê o arquivo DIMACS, converte a fórmula em uma estrutura interna e aplica técnicas

avançadas, como propagação unitária e CDCL, para determinar a satisfatibilidade da fórmula (Eén; Sörensson, 2004).

3 MATERIAIS E MÉTODOS

Este capítulo descreve os recursos, ferramentas e procedimentos adotados para a implementação manual dos algoritmos de verificação de CNF, transformação por Tseitin, conversão para o formato DIMACS e a resolução do problema SAT, incluindo a lógica central do solucionador baseada no algoritmo DPLL com as heurísticas MOM's e VSIDS. Todo o desenvolvimento é realizado em *Python* e executado no ambiente interativo do *Google Colab*.

3.1 Materiais

Para a implementação e experimentação do projeto, são utilizados os seguintes materiais e ferramentas:

- **Ambiente de Desenvolvimento:**
  - ***Google Colab*:** Plataforma utilizada para a execução interativa dos notebooks *Python*, facilitando testes, visualização dos resultados e o compartilhamento do código.
  - **Navegador Web:** Necessário para acessar o *Google Colab*.
- **Linguagem de Programação:**
  - ***Python 3*:** Empregada para implementar os algoritmos de análise de expressões lógicas, transformação por Tseitin, conversão para o formato DIMACS, implementação do DPLL e das heurísticas de decisão.
- **Bibliotecas e Módulos:**
  - *ast.literal_eval*: Utilizada para converter a entrada do usuário (em forma de *string*) em uma estrutura de dados (listas e *strings*) que representa a fórmula lógica.
  - *pprint*: Utilizada para exibir as estruturas de dados de forma legível e organizada.
  - *time*: Utilizada para medir o desempenho temporal das funções e algoritmos implementados, permitindo a análise da eficiência das heurísticas adotadas.
  - *matplotlib.pyplot*: Biblioteca utilizada para a geração de gráficos, possibilitando a visualização comparativa dos resultados experimentais das heurísticas.
  - *copy*: Utilizado para realizar cópias das estruturas de dados, garantindo a preservação das versões originais durante transformações e manipulações.

  - Módulos padrão do *Python*: Empregados para a manipulação de listas e *strings*, fundamentais para representar a árvore de derivação das expressões lógicas.

  Adicionalmente às bibliotecas, são utilizados arquivos no formato DIMACS obtidos a partir de *benchmarks*, amplamente adotados em pesquisas sobre SAT-solver. Esses arquivos servem como entrada para a execução dos testes de desempenho das heurísticas.

- **Recursos de Referência:**
  - Estudos e artigos científicos que abordam Lógica Proposicional, Forma Normal Conjuntiva (CNF) e Transformação de Tseitin (Tseitin, 1983).
  - Documentação do padrão DIMACS para representação de fórmulas SAT, como (Cockburn *et al.*, 2006).
  - Estudos e artigos científicos que abordam *SAT-solvers*, com foco no algoritmo DPLL (Davis; Putnam, 1960; Davis; Logemann; Loveland, 1962).
  - Estudos e artigos científicos que abordam heurísticas de decisão, com foco nas heurísticas MOM's e VSIDS (Freeman, 1995; Moskewicz *et al.*, 2001; Lynce, 2004).

3.2 Métodos

A abordagem metodológica desenvolvida consiste na implementação e integração de módulos responsáveis pelas seguintes etapas:

3.2.1 Formato de Entrada da Fórmula

A expressão lógica é fornecida utilizando a notação prefixa, na qual o operador aparece antes dos operandos.

Exemplo:

['AND', ['OR', 'A', ['NOT', 'B']], 'C']

Nesse exemplo:

- O operador **AND** indica que a operação principal é a conjunção.
- A estrutura hierárquica da expressão é definida pela ordem dos elementos na lista.

Observação: São utilizados exclusivamente os operadores **AND**, **OR** e **NOT** neste trabalho. Embora operadores como **implicação** e **bicondicional** também façam parte da Lógica Proposicional, os três operadores mencionados são suficientes para expressar qualquer fórmula lógica, conforme discutido na seção 2.1.

3.2.2 Verificação da Estrutura CNF

Objetivo: Verificar se a expressão lógica já se encontra na Forma Normal Conjuntiva (CNF), onde cada cláusula é uma disjunção de literais e a expressão uma conjunção dessas cláusulas.

Procedimento: A análise é realizada de forma recursiva:

- Literais (representados por *strings*) são considerados válidos.
- Para o operador **NOT**, garantiu-se que o único operando é um literal.
- Para o operador **OR**, verifica-se se todos os elementos são literais ou negações válidas.
- Para o operador **AND**, cada subexpressão deve representar uma cláusula válida.

3.2.3 Transformação de Tseitin

Objetivo: Converter expressões que não estão em CNF para uma forma equivalente que preserve a satisfatibilidade, evitando o crescimento exponencial do tamanho da fórmula.

Procedimento: A transformação de Tseitin é aplicada de forma recursiva:

- Cada subfórmula é associada a uma variável auxiliar.
- São geradas cláusulas que relacionam essa variável auxiliar à subfórmula original.
- A expressão final garante a equivalência com a expressão original.

3.2.4 Conversão para o Formato DIMACS

Objetivo: Traduzir a expressão lógica, já em CNF, para o padrão DIMACS, utilizado por *SAT-solvers*.

Procedimento:

- **Mapeamento dos Literais:** Cada literal recebe um identificador numérico único, facilitando sua representação em DIMACS.
- **Processamento das Cláusulas:** Cada cláusula é transformada em uma sequência de números inteiros (com literais positivos para variáveis e negativos para negações), terminando com o número zero.
- **Geração do Arquivo:** é gerado um cabeçalho indicando o número total de variáveis e de cláusulas, seguido pelas cláusulas convertidas.

3.2.5 Leitura do arquivo DIMACS

Objetivo: Extrair o número de variáveis e as cláusulas de um arquivo no formato DIMACS para processamento pelo SAT-solver.

Procedimento:

- **Inicialização:** Uma lista de cláusulas vazia é criada para amazenar as cláusulas e o número de variáveis é inicializado com zero.
- **Processamento do arquivo:** Todas as linhas são lidas e para cada linha:
  - Espaços em brancos são removidos.
  - Linhas vazias ou comentários, iniciadas com *c*, são ignorados.
  - Linhas de cabeçalho, iniciadas com *p* são processadas para extrair o número de variáveis.
  - Demais linhas são convertidas em cláusulas: valores são separados e convertidos para inteiros; o último elemento (*0*) é removido, pois indica o término da cláusula; a cláusula é adicionada à lista de cláusulas.
- **Retorno:** A função retorna o número de variáveis e a lista de cláusulas.

3.2.6 Simplificação da Fórmula

Objetivo: Reduzir a fórmula lógica ao aplicar uma nova atribuição de valor a uma variável, eliminando cláusulas já satisfeitas e removendo literais falsificados, para facilitar a busca e garantir que a fórmula reduzida seja satisfatível se a fórmula original também for.

Procedimento: A simplificação da fórmula é realizada conforme descrito abixo:

- Para cada cláusula na fórmula original, é iniciada uma nova cláusula vazia que armazenaria os literais que permanecem após a simplificação.
- A função percorre cada literal da cláusula original, aplicando as seguintes regras:
  - Se o literal corresponde à variável atribuída com valor *True*, a cláusula é considerada satisfeita e descartada.
  - Se o literal é a negação da variável atribuída com valor *False*, a cláusula também é considerada satisfeita e descartada.
  - Se o literal é a negação da variável atribuída com valor *True*, esse literal é removido da cláusula.
  - Se o literal corresponde à variável atribuída com valor *False*, esse literal é removido da cláusula.

  - Literais que não se relacionam à variável atribuída são mantidos.

- Cláusulas já satisfeitas são removidas da nova fórmula.
- Caso alguma cláusula fique vazia após a remoção de literais (indicando um conflito), a função retorna *False* para sinalizar insatisfatibilidade na atribuição atual.
- As cláusulas simplificadas restantes são reunidas formando a nova fórmula simplificada.

3.2.7 Eliminação de Literais Puros

Objetivo: Identificar variáveis que aparecem exclusivamente com o mesmo sinal (positivo ou negativo) na fórmula e atribuir valores que as satisfaçam diretamente. Essa técnica tem como propósito simplificar a fórmula e reduzir o espaço de busca, uma vez que tais variáveis não causam conflitos.

Procedimento: A eliminação de literais puros é realizada por meio da análise dos sinais com que cada variável aparece nas cláusulas.

- É criado um dicionário para mapear cada variável ao conjunto de sinais (*True* para positivo e *False* para negativo) com os quais ela aparece na fórmula. Também é feita uma cópia da atribuição atual, a fim de preservar o estado original.
- A função percorre todas as cláusulas da fórmula, coletando os sinais dos literais e organizando-os por variável.
- Em seguida, é verificado, para cada variável, se ela aparece com apenas um sinal (ou sempre positiva, ou sempre negativa) e se ainda não há atribuição feita.
- Quando uma variável é considerada pura, ela recebe uma atribuição de acordo com o sinal observado.
- A fórmula é simplificada com base nessa nova atribuição.
- Caso a simplificação resulte em um conflito (em uma fórmula insatisfatível), o processo é interrompido com falha.
- Ao final, a função retorna a fórmula simplificada e o conjunto de atribuições atualizado.

3.2.8 Propagação Unitária

Objetivo: Automatizar a atribuição de valores a variáveis por meio da identificação e resolução de cláusulas unitárias, simplificando a fórmula e reduzindo o espaço de busca.

Procedimento: O procedimento é implementado como um ciclo que identifica e resolve iterativamente as cláusulas unitárias presentes na fórmula:

- A função verifica continuamente a presença de cláusulas unitárias na fórmula.
- Antes de aplicar a atribuição, é realizada uma verificação para detectar possíveis conflitos com valores já atribuídos. Em caso de conflito, a propagação é interrompida com falha.
- Quando não há conflito, a fórmula é simplificada com base na nova atribuição.
- Caso a simplificação gere uma cláusula vazia, é indicado um conflito lógico, e o processo é encerrado.
- Esse procedimento se repete enquanto ainda restarem cláusulas unitárias a serem tratadas.
- Ao final, a função retorna a fórmula simplificada e o conjunto de atribuições atualizado ou, em caso de conflito, um indicativo de falha.

3.2.9 Heurística de Decisão MOM's

Objetivo: Selecionar, dentre as variáveis não atribuídas, aquela que maximiza a chance de levar a uma decisão relevante, com base na frequência com que aparece em cláusulas de menor tamanho. Essa escolha visa acelerar a identificação de conflitos ou soluções.

Procedimento: A heurística MOM's é aplicada da seguinte forma:

- É determinada a menor cláusula da fórmula, em termos de quantidade de literais.
- Em seguida, são filtradas todas as cláusulas que possuem esse mesmo tamanho mínimo.
- A função percorre essas cláusulas, contando quantas vezes cada variável não atribuída aparece com sinal positivo e com sinal negativo.
- Para cada variável, é calculado um *score* utilizando a fórmula $(p + 1) \times (n + 1)$, onde p representa a quantidade de ocorrências positivas e n, as negativas.
- A variável com maior *score* é escolhida como prioridade, por apresentar maior potencial de impacto nas próximas decisões do algoritmo.
- Ao final, a função retorna essa variável para que seja utilizada como próxima decisão no algoritmo DPLL.

3.2.10 Heurística de Decisão VSIDS

Objetivo: Priorizar a escolha de variáveis que estejam recentemente envolvidas em conflitos utilizando um decaimento exponencial ao longo do tempo. A finalidade é direcionar

o algoritmo para áreas do espaço de busca com maior probabilidade de solução, tornando o processo de decisão mais eficiente.

Procedimento: A heurística VSIDS é implementada por meio da manutenção de uma pontuação para cada variável, atualizada ao longo da execução. O processo segue os seguintes passos:

- Um dicionário é utilizado para armazenar os *scores* das variáveis, representando sua relevância na resolução de conflitos.
- Sempre que uma cláusula conflitante é detectada, a variável responsável por essa decisão tem seu *score* incrementado..
- Após cada conflito é aplicada uma operação de decaimento exponencial (fator constante igual a 0.95) aos *scores* de todas as variáveis, reduzindo a influência de conflitos antigos.
- Para a seleção da próxima variável de decisão, a função percorre as cláusulas da fórmula em busca da variável não atribuída com maior pontuação acumulada.
- A variável escolhida é retornada para ser testada com os valores *True* e *False* pelo algoritmo DPLL.

3.2.11 Algoritmo DPLL com Heurísticas MOM's e VSIDS

Objetivo: Implementar manualmente um solucionador SAT baseado no algoritmo DPLL, integrando as heurísticas de decisão MOM's e VSIDS para avaliar o comportamento de ambas no processo de busca.

Procedimento:A implementação do solucionador é estruturada de forma modular, permitindo a escolha da heurística utilizada em tempo de execução. Para fins de análise comparativa, o processo é executado sequencialmente duas vezes sobre a mesma fórmula: uma com cada heurística. O procedimento para cada uma dessas execuções consiste nas seguintes etapas principais:

- A busca começa com uma pilha contendo o estado inicial (a fórmula original e um conjunto vazio de atribuições).
- Para cada estado retirado da pilha, é realizado um ciclo de simplificações utilizando propagação unitária e eliminação de literais puros. A implementação dessas simplificações varia conforme a heurística selecionada:
  - Com a heurística MOM's, as funções de propagação unitária e eliminação de literal puro preservam o estado original das estruturas, gerando cópias a cada iteração.

  - Com a heurística VSIDS, as funções são adaptadas para modificar diretamente as estruturas de dados, permitindo o acompanhamento dinâmico das pontuações das variáveis durante a execução.

- Se a fórmula é simplificada até se tornar vazia, isso indicava que todas as cláusulas foram satisfeitas e a atribuição é retornada como solução.
- Caso seja detectado um conflito (cláusula vazia), o sistema retoma a pilha para explorar outras possibilidades (*backtrack*).
- A escolha da próxima variável a ser testada é feita com base na heurística selecionada:
  - Com MOM's, a variável de maior pontuação é escolhida.
  - Com VSIDS, é escolhida a variável com maior *score* acumulado entre as não atribuídas.
- Para a heurística VSIDS, a cada detecção de conflito é realizado:
  - O incremento da pontuação da variável que causa o conflito.
  - O decaimento das pontuações de todas as variáveis, aplicando um fator multiplicativo.

A função final retorna *True* e a atribuição satisfatória caso a fórmula sea solucionável, ou *False* caso contrário.

3.2.12 Integração e Execução

Objetivo: Integrar os módulos de verificação, transformação, conversão e resolução para possibilitar a execução interativa e completa da expressão lógica inserida pelo usuário ou lida de um arquivo.

A arquitetura geral da implementação, que integra os diversos módulos em um fluxo completo desde a entrada de dados até a resolução final, é visualmente representada no fluxograma da Figura 1.

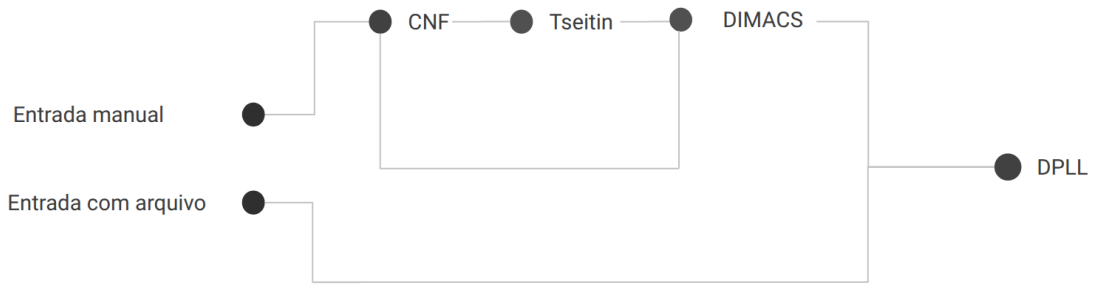

Figura 1 – Fluxograma geral da implementação
Fonte: Autoria própria (2025)

Procedimento: O sistema é desenvolvido com um menu interativo que permite ao usuário:

- Escolher entre inserir manualmente uma expressão lógica ou carregar um arquivo no formato DIMACS.
- Em caso de entrada manual:
  - Recebe a expressão em notação prefixa.
  - Verifica se a expressão está em CNF.
  - Aplica a transformação de Tseitin, se necessário.
  - Converte a expressão para o formato DIMACS.
  - Gera um mapa de literais para facilitar a leitura da saída do solucionador.
  - Salva a saída gerada em um arquivo DIMACS.
- Em caso de entrada por arquivo:
  - Lê o conteúdo do arquivo DIMACS informado.
  - Interpreta suas cláusulas para execução.
- Após a leitura da fórmula:
  - Executa o solucionador DPLL com duas heurísticas de decisão: MOM's e VSIDS.
  - Apresenta se a fórmula é satisfatível ou insatisfatível.
  - Caso seja satisfatível, exibe as atribuições das variáveis, através do mapa de literais.
  - Exibe o tempo de execução total de cada heurística.
  - Gera um gráfico comparativo dos tempos acumulados das duas execuções.

3.2.13 Testes e Validação

Objetivo: Testar o funcionamento dos algoritmos em diferentes cenários de entrada, incluindo a transformação para CNF, a conversão para o formato DIMACS e a resolução de fórmulas lógicas utilizando o algoritmo DPLL com heurísticas de decisão.

Procedimentos:

- São utilizadas expressões lógicas já na CNF para validar a conversão para DIMACS.
- Realiza-se testes com expressões que não estão na CNF para confirmar a eficácia da transformação de Tseitin.

- Os resultados são analisados por meio da inspeção dos mapas de literais e da estrutura das cláusulas geradas.

Para validar o solucionador, o resultado final de cada teste (SAT/UNSAT) foi também verificado em relação à resposta correta e documentada para cada benchmark utilizado, assegurando a precisão da ferramenta.

A Figura 2 mostra uma expressão em sua forma normal conjuntiva e o retorno do formato DIMACS:

```
Escolha o modo de entrada:
1 - Inserir expressão lógica manualmente
2 - Usar arquivo DIMACS existente
Opção (1 ou 2): 1

Insira uma expressão lógica iniciando com o operador:
['AND', ['OR', 'A', ['NOT', 'B']], 'C']

Expressão já está em CNF.

Formato DIMACS:
p cnf 3 2
1 -2 0
3 0


Mapa de literais:
{'A': 1, 'B': 2, 'C': 3}
```

Figura 2 – Transformação de uma expressão lógica na CNF para o formato DIMACS
Fonte: Autoria própria (2025)

Neste exemplo, a primeira linha contém um comentário que orienta a inserção da expressão lógica. Em seguida, a segunda linha exibe a expressão inserida, que já está na CNF. A terceira linha, também um comentário, informa que o formato DIMACS é gerado como saída. A linha *p cnf 3 2* indica que a fórmula é convertida para o formato DIMACS, onde:

- **3** representa o número de variáveis distintas na expressão lógica. Neste caso, as variáveis são A, B e C.
- **2** representa o número de cláusulas presentes na fórmula.

As linhas seguintes correspondem às cláusulas da expressão lógica, onde cada número representa uma variável de acordo com o mapa de literais:

- *1 -2 0* corresponde à cláusula $(A \vee \neg B)$. Aqui, *1* representa a variável A, *-2* representa a negação de B, e o *0* indica o fim da cláusula.
- *3 0* corresponde à cláusula (C), onde *3* representa a variável C e *0* marca o término da cláusula.

O mapa de literais, mostrado na última parte, associa cada variável lógica a um número inteiro:

$$\{ A : 1, B : 2, C : 3 \}$$

A Figura 3 mostra o resultado da execução do algoritmo DPLL sobre a expressão lógica apresentada anteriormente na Figura 2, que já se encontra na CNF.

```
Número de variáveis: 3
Cláusulas:
[1, -2]
[3]
SAT com MOM's
Atribuições: {'A': True, 'B': False, 'C': True}
Tempo total MOM's: 0.0001 segundos
SAT com VSIDS
Atribuições: {'A': True, 'B': False, 'C': True}
Tempo total VSIDS: 0.0000 segundos
```

Figura 3 – Execução do DPLL com aplicação das heurísticas MOM's e VSIDS em uma fórmula SAT
Fonte: Autoria própria (2025)

Neste exemplo, a primeira linha indica que a fórmula possui 3 variáveis e em seguida as cláusulas, que foram descritas na Figura 2.

- A heurística MOM's encontra uma atribuição satisfatória em 0.0001 segundos com as seguintes atribuições:

  {A: True, B: False, C: True}

- A heurística VSIDS também retorna a mesma solução satisfatória, com as mesmas atribições:

  {A: True, B: False, C: True}, em 0.0000 segundos.

Ambas as heurísticas indicam que a fórmula é **satisfatível**, ou seja, existe uma combinação de valores lógicos para as variáveis que torna todas as cláusulas verdadeiras.

Adicionalmente, para validar a funcionalidade de entrada de dados por arquivo, o mesmo problema lógico é salvo como um arquivo DIMACS e executado através da segunda opção do menu interativo. A Figura 4 ilustra essa execução.

```
Escolha o modo de entrada:
1 - Inserir expressão lógica manualmente
2 - Usar arquivo DIMACS existente
Opção (1 ou 2): 2

Digite o caminho do arquivo DIMACS: /content/teste_sat.txt

Número de variáveis: 3
Cláusulas:
[1, -2]
[3]
SAT com MOM's
Atribuições: {1: True, 2: False, 3: True}
Tempo total MOM's: 0.0001 segundos
SAT com VSIDS
Atribuições: {1: True, 2: False, 3: True}
Tempo total VSIDS: 0.0000 segundos
```

Figura 4 – Execução do DPLL a partir de um arquivo DIMACS existente
Fonte: Autoria própria (2025)

Como observado na Figura 4, o sistema lê corretamente o arquivo, identifica as variáveis e as cláusulas. O resultado da execução é idêntico ao obtido pela entrada manual (Figura 3), com ambas as heurísticas MOM's e VSIDS confirmando que a fórmula é satisfatível com as mesmas atribuições. Este teste confirma a implementação correta da leitura e interpretação de arquivos no formato DIMACS.

A Figura 5 mostra que a expressão de entrada não está em sua forma normal conjuntiva. Então é aplicada a transformação de Tseitin, e retornada a expressão na CNF juntamente com o formato DIMACS.

```
Escolha o modo de entrada:
1 - Inserir expressão lógica manualmente
2 - Usar arquivo DIMACS existente
Opção (1 ou 2): 1

Insira uma expressão lógica iniciando com o operador:
['OR', ['AND', 'A', ['NOT', 'A',]], ['AND', 'B', ['NOT', 'B']]]

Expressão transformada em CNF:
['AND',
 ['OR', ['NOT', 'aux_1'], ['NOT', 'A']],
 ['OR', 'aux_1', 'A'],
 ['OR', ['NOT', 'aux_2'], 'A'],
 ['OR', ['NOT', 'aux_2'], 'aux_1'],
 ['OR', 'aux_2', ['NOT', 'A'], ['NOT', 'aux_1']],
 ['OR', ['NOT', 'aux_3'], ['NOT', 'B']],
 ['OR', 'aux_3', 'B'],
 ['OR', ['NOT', 'aux_4'], 'B'],
 ['OR', ['NOT', 'aux_4'], 'aux_3'],
 ['OR', 'aux_4', ['NOT', 'B'], ['NOT', 'aux_3']],
 ['OR', ['NOT', 'aux_5'], 'aux_2', 'aux_4'],
 ['OR', 'aux_5', ['NOT', 'aux_2']],
 ['OR', 'aux_5', ['NOT', 'aux_4']],
 ['OR', 'aux_5']]

Formato DIMACS:
p cnf 7 14
-1 -2 0
1 2 0
-3 2 0
-3 1 0
3 -2 -1 0
-4 -5 0
4 5 0
-6 5 0
-6 4 0
6 -5 -4 0
-7 3 6 0
7 -3 0
7 -6 0
7 0


Mapa de literais:
{'aux_1': 1, 'A': 2, 'aux_2': 3, 'aux_3': 4, 'B': 5, 'aux_4': 6, 'aux_5': 7}
```

Figura 5 – Transformação de uma expressão lógica para CNF e para o formato DIMACS
Fonte: Autoria própria (2025)

Neste exemplo, a primeira linha também contém um comentário que orienta a inserção da expressão lógica. Em seguida, a segunda linha exibe a expressão inserida, que ainda não está na CNF.

A terceira linha informa que a expressão é transformada para a forma normal conjuntiva (CNF) utilizando a transformação de Tseitin.

A linha *p cnf 7 14* indica que a fórmula é convertida para o formato DIMACS, onde:

- **7** representa o número de variáveis distintas na expressão lógica..
- **14** representa o número de cláusulas presentes na fórmula.

As linhas seguintes correspondem às cláusulas da expressão lógica, onde cada número representa uma variável de acordo com o mapa de literais:

- *-1 -2 0* corresponde à cláusula $(\neg aux\_1 \vee \neg A)$.
- *1 2 0* corresponde à cláusula $(aux\_1 \vee A)$.
- *-3 2 0* corresponde à cláusula $(\neg aux\_2 \vee A)$.

- *-3 1 0* corresponde à cláusula $(\neg aux\_2 \vee aux\_1)$.
- *3 -2 -1 0* corresponde à cláusula $(aux\_2 \vee \neg A \vee \neg aux\_1)$.
- *-4 -5 0* corresponde à cláusula $(\neg aux\_3 \vee \neg B)$.
- *4 5 0* corresponde à cláusula $(aux\_3 \vee B)$.
- *-6 5 0* corresponde à cláusula $(\neg aux\_4 \vee B)$.
- *-6 4 0* corresponde à cláusula $(\neg aux\_4 \vee aux\_3)$.
- *6 -5 -4 0* corresponde à cláusula $(aux\_4 \vee \neg B \vee \neg aux\_3)$.
- *-7 3 6 0* corresponde à cláusula $(\neg aux\_5 \vee aux\_2 \vee aux\_4)$.
- *7 -3 0* corresponde à cláusula $(aux\_5 \vee \neg aux\_2)$.
- *7 -6 0* corresponde à cláusula $(aux\_5 \vee \neg aux\_4)$.
- *7 0* corresponde à cláusula (aux_5).

O mapa de literais, mostrado na última parte, associa cada variável lógica a um número inteiro:

$$\{\, aux\_1 : 1,\ A : 2,\ aux\_2 : 3,\ aux\_3 : 4,\ B : 5,\ aux\_4 : 6,\ aux\_5 : 7 \,\}$$

- O algoritmo DPLL, incluindo as heurísticas MOM's e VSIDS, é testado com fórmulas satisfatíveis e insatisfatíveis, garantindo o resultado correto.

A Figura 6 mostra o resultado da execução do algoritmo DPLL sobre a expressão lógica apresentada anteriormente na Figura 5, que não se encontra na CNF.

```
Número de variáveis: 7
Cláusulas:
[-1, -2]
[1, 2]
[-3, 2]
[-3, 1]
[3, -2, -1]
[-4, -5]
[4, 5]
[-6, 5]
[-6, 4]
[6, -5, -4]
[-7, 3, 6]
[7, -3]
[7, -6]
[7]
UNSAT com MOM's.
Tempo total MOM's: 0.0004 segundos
UNSAT com VSIDS.
Tempo total VSIDS: 0.0001 segundos
```

Figura 6 – Execução do DPLL com aplicação das heurísticas MOM's e VSIDS em uma fórmula UNSAT
Fonte: Autoria própria (2025)

Neste exemplo, a primeira linha indica que a fórmula possui 7 variáveis e em seguida as cláusulas, que foram descritas na Figura 5.

- A heurística MOM's não encontrou uma atribuição que satisfaz a fórmula.
- A heurística VSIDS também não encontrou uma atribuição que satisfaz a fórmula.

A ausência de uma atribuição satisfatória confirma que a fórmula convertida por Tseitin mantém a equivalência lógica com a expressão original, e que de fato não existe uma combinação de valores que satisfaça simultaneamente todas as cláusulas.

Esses exemplos demonstram a resolução de uma fórmula simples em CNF, além de permitir uma comparação inicial do desempenho entre as heurísticas utilizadas e também a capacidade do algoritmo e das heurísticas de reconhecer fórmulas logicamente inconsistentes, mesmo após o processo de transformação de Tseitin.

A implementação deste trabalho pode ser encontrada em (Morais, 2025).

4 RESULTADOS E DISCUSSÃO

Este capítulo apresenta os resultados experimentais obtidos com o SAT-solver desenvolvido, com foco na análise comparativa de desempenho entre as heurísticas de decisão MOM's e VSIDS. Conforme demonstrado no Capítulo 3, a ferramenta foi funcionalmente validada. O objetivo desta seção é, portanto, avaliar quantitativamente a eficiência de cada heurística conforme implementadas neste trabalho, para resolver problemas SAT de diferentes complexidades.

4.1 Metodologia Experimental

Para realizar a análise comparativa, foi selecionado um conjunto de instâncias no formato DIMACS, provenientes do repositório SATLIB (Hoos; Stützle, 2000). Este é um acervo de benchmarks clássico e confiável, amplamente adotado pela comunidade de pesquisa em SAT para a avaliação de solucionadores. Uma vantagem crucial deste repositório é que as instâncias já são categorizadas como satisfatíveis (SAT) ou insatisfatíveis (UNSAT), possuindo um resultado conhecido. Foram escolhidos problemas de ambas as categorias, com 100, 125, 150 e 175 variáveis, a fim de avaliar o comportamento das heurísticas em diferentes cenários: a busca por uma solução existente (nas instâncias SAT) e a prova completa de sua ausência (nas instâncias UNSAT).

Cada instância foi executada duas vezes na ferramenta: uma utilizando a heurística MOM's e outra utilizando a VSIDS. O tempo de execução e o número de decisões tomadas foram os principais indicadores de desempenho monitorados. Todos os testes foram conduzidos no mesmo ambiente computacional (*Google Colab*) para garantir a consistência e a justa comparabilidade dos resultados.

A definição do conjunto de testes seguiu um critério de relevância. Em uma fase exploratória, instâncias com um número reduzido de variáveis foram testadas, mas seus tempos de execução se mostraram insignificantes (como mostrado na Figura 3 e na Figura 6, o que inviabilizaria uma análise comparativa de desempenho. Com base nisso, foram selecionadas as instâncias mais complexas (com 100, 125, 150 e 175 variáveis) para o estudo. Para este conjunto final, realizou-se uma média de 10 execuções por teste, nas quais se confirmou que os resultados eram altamente estáveis: o número de decisões mostrou-se idêntico em todas as execuções — um comportamento esperado para algoritmos determinísticos — e a variação no tempo foi desprezível. Sendo assim, os resultados apresentados neste capítulo correspondem a uma única execução representativa de cada teste.

4.2 Apresentação dos Resultados

Os resultados da execução comparativa são apresentados visualmente nas figuras a seguir. A cada passo da execução do algoritmo DPLL, foram extraídos o número de decisões realizadas até o momento, assim como o tempo total acumulado da execução. Os dados são

extraídos até que a instância seja resolvida. Os gráficos ilustram, para cada heurísitca, a relação entre o número de decisões realizadas e o tempo acumulado total até o final da execução.

Na maioria das instâncias testadas, a heurística MOM's concluiu a busca de forma mais eficiente. Contudo, uma exceção notável e de grande importância para a análise ocorre na instância satisfatível de 150 variáveis (Figura 9), onde a heurística VSIDS demonstrou um desempenho superior, encontrando a solução em um tempo menor e com um número menor de decisões.

Por fim, a Figura 15 apresenta um panorama geral dos tempos de execução obtidos em todos os testes e a Figura 16 representa o número de decisões.

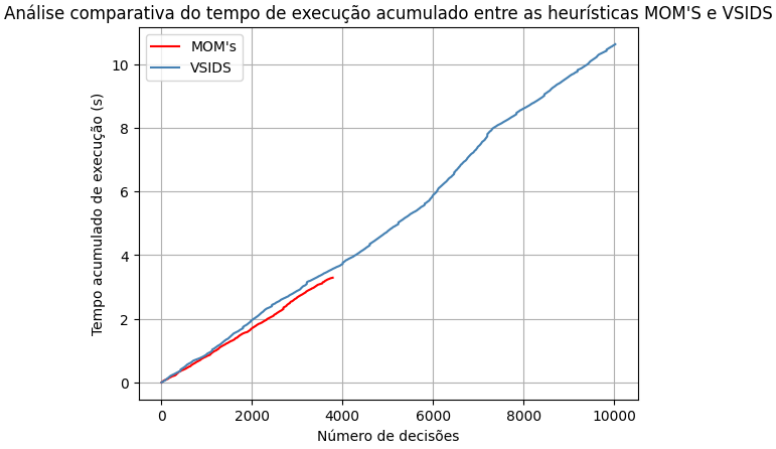

Figura 7 – Instância SAT com 100 variáveis.[1]
Fonte: autoria própria (2025).

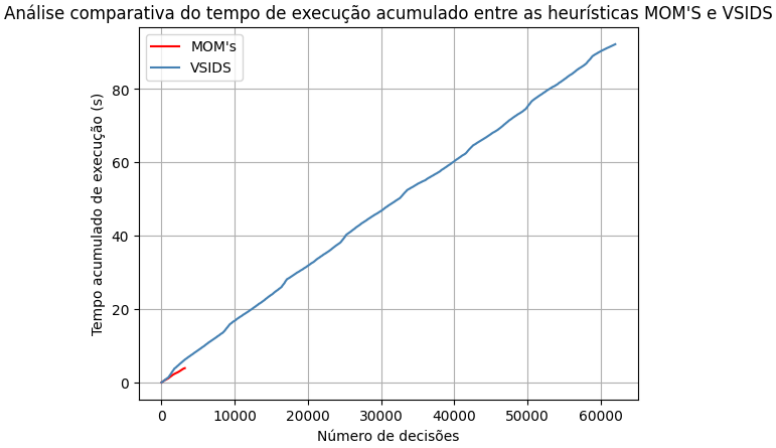

Figura 8 – Instância SAT com 125 variáveis.[2]
Fonte: autoria própria (2025).

[1] Instância `sat_100` disponível em: GitHub
[2] Instância `sat_125` disponível em: GitHub

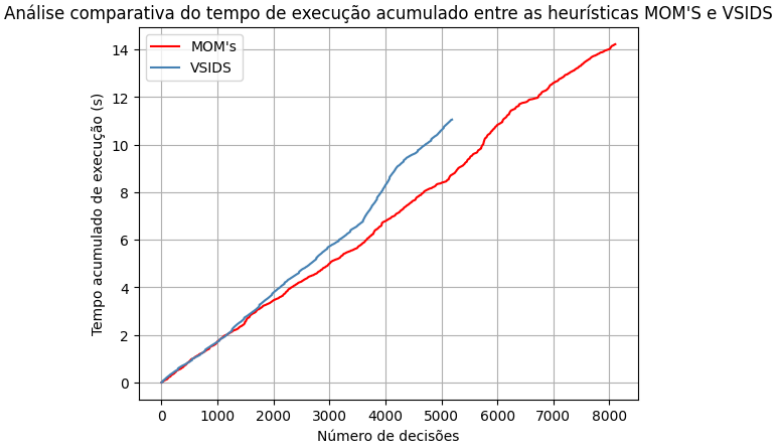

Figura 9 – Instância SAT com 150 variáveis.[3]
Fonte: autoria própria (2025).

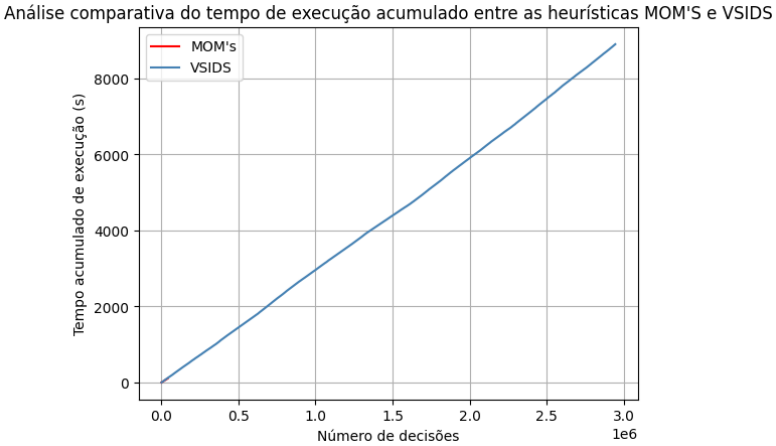

Figura 10 – Instância SAT com 175 variáveis.[4]
Fonte: autoria própria (2025).

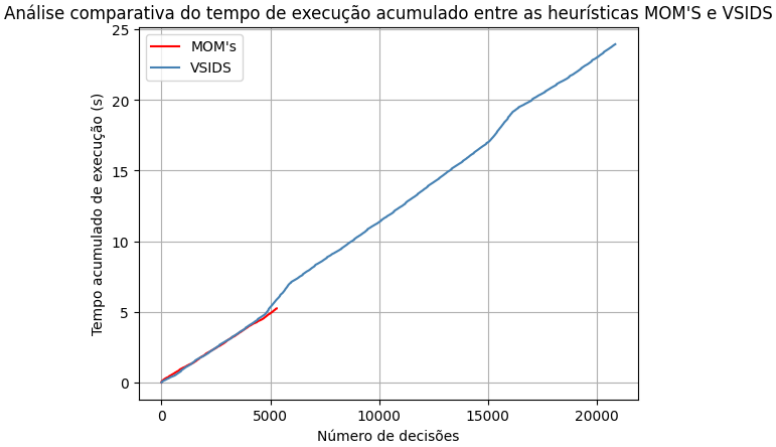

Figura 11 – Instância UNSAT com 100 variáveis.[5]
Fonte: autoria própria (2025).

[3] Instância `sat_150` disponível em: GitHub
[4] Instância `sat_175` disponível em: GitHub
[5] Instância `unsat_100` disponível em: GitHub

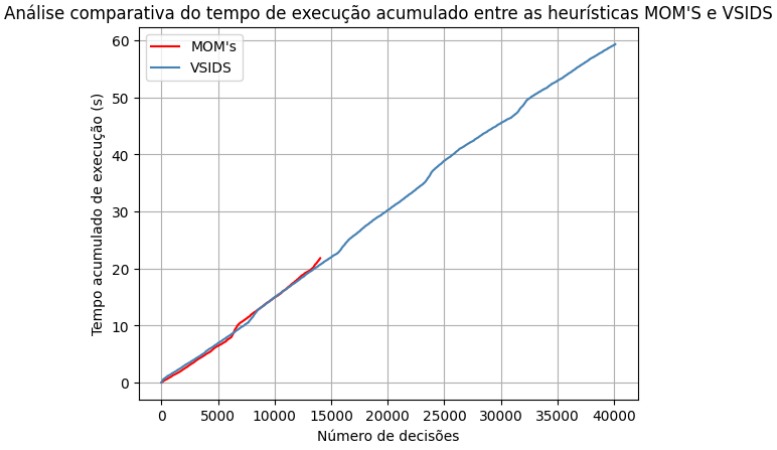

Figura 12 – Instância UNSAT com 125 variáveis.[6]
Fonte: autoria própria (2025).

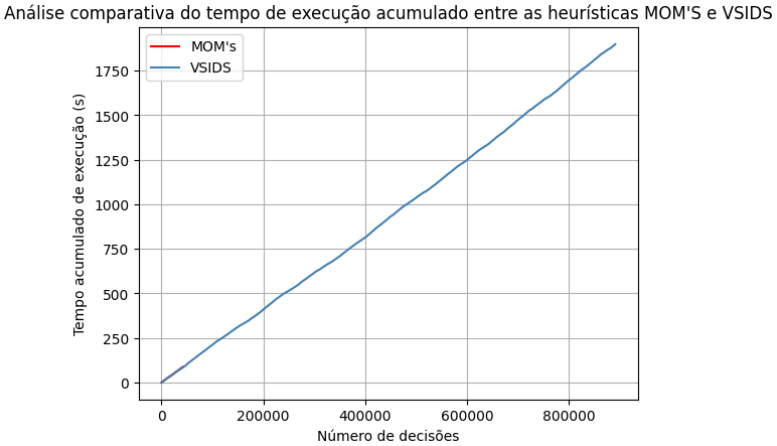

Figura 13 – Instância UNSAT com 150 variáveis.[7]
Fonte: autoria própria (2025).

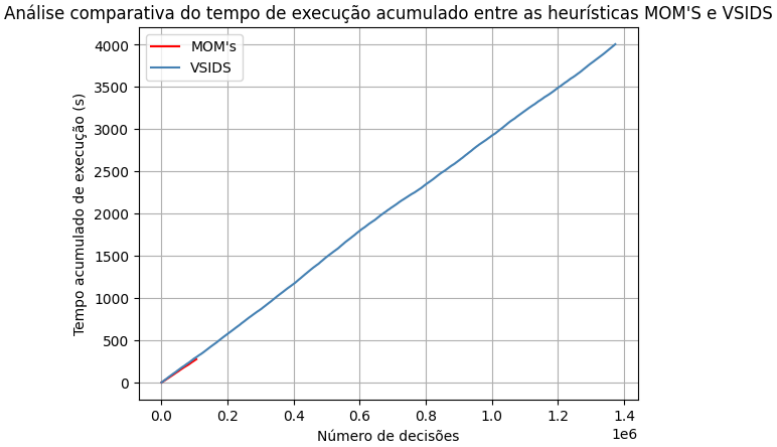

Figura 14 – Instância UNSAT com 175 variáveis.[8]
Fonte: autoria própria (2025).

[6] Instância `unsat_125` disponível em: GitHub
[7] Instância `unsat_150` disponível em: GitHub
[8] Instância `unsat_175` disponível em: GitHub

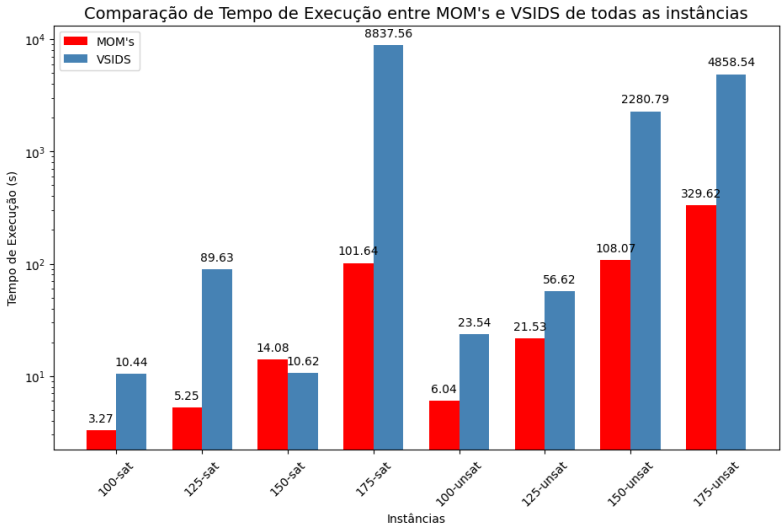

Figura 15 – Tempo comparativo de todas as instâncias entre a heurística MOM's e VSIDS.
Fonte: autoria própria (2025).

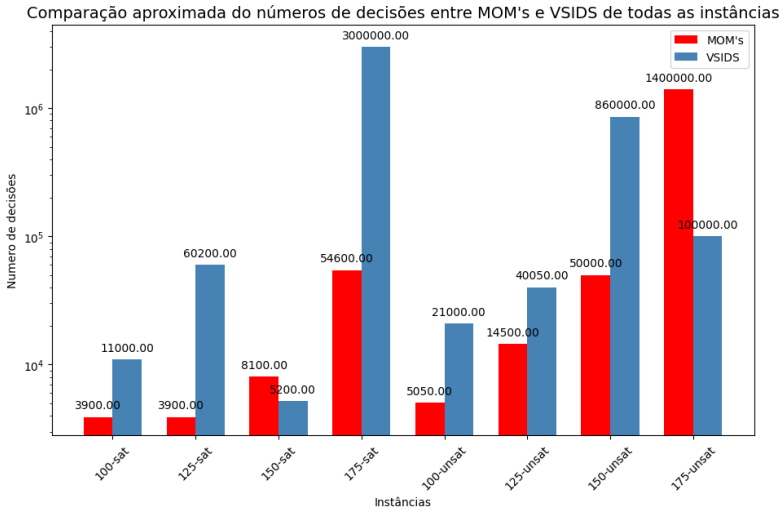

Figura 16 – Comparativo do número aproximado de decisões de todas as instâncias entre a heurística MOM's e VSIDS.
Fonte: autoria própria (2025).

4.3 Discussão dos Resultados

A superioridade da MOM's na maioria dos testes pode ser atribuída à sua implementação computacionalmente mais leve em comparação com a da VSIDS neste trabalho. Conforme descrito na metodologia, a heurística VSIDS realiza uma análise completa da fórmula a cada nova decisão, o que resulta em um custo computacional elevado. Já a heurística MOM's examina apenas um subconjunto reduzido de cláusulas, tornando o processo de escolha da variável significativamente mais rápido em cada etapa.

Nota-se também que o uso da heurística MOM's reduziu drasticamente o número de decisões realizadas até a resolução da fórmula, em comparação com a VSIDS. Esse número de decisões, ao contrário do tempo de execução, reflete diretamente a capacidade da heurística

em podar a árvore de busca, evidenciando que a MOM's pode ser mais adequada à estrutura das fórmulas utilizadas nos testes.

No entanto, a instância apresentada na Figura 9 destaca-se por um comportamento distinto. Nesse caso, a heurística VSIDS obteve desempenho superior em relação à MOM's, apesar de seu maior custo computacional por decisão. Esse resultado indica que a estrutura da fórmula apresentava elevada complexidade combinatória, o que dificultou a eficácia da heurística MOM's, cujo mecanismo de decisão mais restrito a levou a percorrer extensivamente caminhos não satisfatórios.

5 CONCLUSÃO

Este trabalho teve como proposta o desenvolvimento de um solucionador para o problema SAT, fundamentado no algoritmo DPLL, abrangendo desde a verificação e transformação de fórmulas lógicas até a resolução propriamente dita. Além disso, buscou-se comparar o desempenho de duas heurísticas de decisão amplamente estudadas: MOM's, de natureza estática e estrutural, e VSIDS, de caráter dinâmico e adaptativo.

A fase experimental, cujos resultados foram detalhados no Capítulo 4, revelou uma dinâmica de desempenho complexa e instrutiva. Contrariando a expectativa inicial de que a heurística moderna VSIDS seria universalmente superior, a análise demonstrou que a implementação da MOM's, por ser computacionalmente mais leve, obteve melhor desempenho na maioria das instâncias testadas. No entanto, foi observado que em um problema de estrutura de conflitos específica, a VSIDS, mesmo com sua implementação não otimizada, foi capaz de superar a MOM's, evidenciando o potencial de sua estratégia de guiamento de busca.

Diante do exposto, pode-se afirmar que todos os objetivos específicos propostos na subseção 1.1.2 foram alcançados. Foi desenvolvido um sistema completo capaz de verificar se uma fórmula está em CNF, aplicar a transformação de Tseitin, converter para o formato DIMACS, e resolver instâncias por meio de um solucionador baseado no algoritmo DPLL, com suporte à comparação entre as heurísticas de decisão MOM's e VSIDS.

A principal contribuição deste trabalho reside na análise da influência que escolhas de implementação exercem sobre o desempenho de algoritmos de busca para o problema SAT. Mais do que a construção de um solucionador eficiente, o estudo permitiu observar como heurísticas com fundamentos distintos se comportam perante a diferentes estruturas de instância, evidenciando que a eficácia teórica só se traduz em vantagem prática quando acompanhada de uma implementação adequada. Além disso, o projeto proporcionou a aplicação integrada de conceitos fundamentais de lógica, algoritmos e complexidade computacional, contribuindo para uma compreensão sobre a resolução de problemas NP-completos.

5.1 Limitações e Trabalhos Futuros

A principal limitação deste trabalho, identificada através da análise de resultados e que explica diretamente o desempenho superior da MOM's na maioria dos cenários, é a implementação da heurística VSIDS, cuja estratégia de atualização e seleção de variáveis foi realizada por meio de varreduras completas na fórmula, resultando em um alto custo computacional. Tal fator comprometeu uma avaliação mais equilibrada entre os potenciais teóricos das heurísticas analisadas.

Dessa forma, os seguintes trabalhos futuros são propostos:

- **Otimização da Heurística VSIDS:** Reestruturar a implementação da VSIDS, incorporando estruturas de dados mais adequadas, como filas de prioridade (*heaps*), além de

adotar estratégias de atualização mais eficientes e localizadas, baseadas na análise de cláusulas de conflito.

- **Uso de Estruturas de Dados Avançadas:** Integrar técnicas como os *literais vigiados* (*watched literals*) à implementação do DPLL, visando otimizar a propagação unitária e reduzir o custo computacional durante a verificação das cláusulas.
- **Implementação de CDCL com Aprendizado de Cláusulas:** Evoluir a arquitetura do solucionador de DPLL para CDCL, incorporando mecanismos de aprendizado de cláusulas a partir de conflitos (*clause learning*) e estratégias de retrocesso não-cronológico.

Essas melhorias permitirão uma investigação mais aprofundada sobre o comportamento das heurísticas em diferentes cenários e contribuirão para o desenvolvimento de um solucionador mais eficiente.

REFERÊNCIAS

ANBULAGAN. Extending unit propagation look-ahead of dpll procedure. *In*: ZHANG, C.; GUESGEN, H. W.; YEAP, W.-K. (Ed.). PRICAI 2004: TRENDS IN ARTIFICIAL INTELLIGENCE. 2004, Berlin, Heidelberg. **Anais [...]** Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. p. 173–182. ISBN 978-3-540-28633-2.

AUDEMARD, G.; SIMON, L. Predicting learnt clauses quality in modern sat solvers. *In*: PROCEEDINGS OF THE 21ST INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE (IJCAI). 2009. **Anais [...]** [*S.l.*: *s.n.*], 2009. p. 399–404.

BIERE, A.; HEULE, M.; MAAREN, H. van. **Handbook of satisfiability**. [*S.l.*]: IOS press, 2009. v. 185.

BOCHENSKI, J. M. **A History of Formal Logic**. University of Notre Dame Press, 1961. Notre Dame, Ind. ISBN 9788582600542. Disponível em: https://archive.org/details/historyofformall00boch/page/n3/mode/1up.

BOOLE, G. **An Investigation of the Laws of Thought: On Which Are Founded the Mathematical Theories of Logic and Probabilities**. [*S.l.*]: Walton and Maberly, 1854. v. 2.

COCKBURN, S. *et al.* Some problems are np-harder than others. **DIMACS Educational Module Series**, DIMACS Center Rutgers University, 96 Frelinghuysen Road, Piscataway, NJ 08854-8018 v. 06, n. 1„ 2006. Disponível em: http://archive.dimacs.rutgers.edu/Publications/Modules/Module06-1/dimacs06-1.pdf.

COOK, S. A. The complexity of theorem proving procedures. **Proceedings of the Third Annual ACM Symposium on Theory of Computing (STOC)**, „ p. 151–158, 1971.

DAVIS, M.; LOGEMANN, G.; LOVELAND, D. A machine program for theorem-proving. **Commun. ACM**, Association for Computing Machinery v. 5, n. 7, p. 394–397, 1962.

DAVIS, M.; PUTNAM, H. A computing procedure for quantification theory. **Journal of the ACM**, Association for Computing Machinery v. 7, n. 3, p. 201–215, 1960.

DEQUEN, G.; DUBOIS, O. An efficient approach to solving random k-sat problems. **Journal of Automated Reasoning**, Springer v. 37, n. 4, p. 261–276, 2006.

DU, Y.-H. K.; WANG, C. A branching heuristic based on variable state change. *In*: PROCEEDINGS OF THE 19TH INTERNATIONAL CONFERENCE ON FORMAL METHODS IN COMPUTER-AIDED DESIGN (FMCAD). 2017. **Anais [...]** [*S.l.*]: IEEE, 2017. p. 168–175.

EÉN, N.; MISHCHENKO, A.; SÖRENSSON, N. Applying logic synthesis for speeding up sat. *In*: MARQUES-SILVA, J.; SAKALLAH, K. A. (Ed.). THEORY AND APPLICATIONS OF SATISFIABILITY TESTING – SAT 2007. 2007, Berlin, Heidelberg. **Anais [...]** Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. p. 272–286.

EÉN, N.; SÖRENSSON, N. An extensible sat-solver. *In*: GIUNCHIGLIA, E.; TACCHELLA, A. (Ed.). THEORY AND APPLICATIONS OF SATISFIABILITY TESTING. 2004. **Anais [...]** [*S.l.*]: Springer Berlin Heidelberg, 2004. p. 502–518.

EÉN, N.; SÖRENSSON, N. Temporal induction by incremental sat solving. **Electronic Notes in Theoretical Computer Science**, v. 89, n. 4, p. 543–560, 2003.

FREDRIKSON, M.; PLATZER, A. Lecture notes on sat solvers & dpll, „,. 2023.

FREEMAN, J. W. **Improvements to propositional satisfiability search algorithms**. [*S.l.*]: University of Pennsylvania, 1995.

GONG, W.; ZHOU, X. A survey of sat solver. *In*: AIP PUBLISHING. AIP CONFERENCE PROCEEDINGS. 1836 n. 1., 2017. **Anais [...]** [*S.l.*], 2017.

HOOS, H. H.; STÜTZLE, T. **SATLIB: The Satisfiability Library - Benchmark Problems**. [*S.l.*]: , 2000. Disponível em: https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html.

JOHANNSEN, J. The complexity of pure literal elimination. **Journal of Automated Reasoning**, Springer v. 35„ p. 89–95, 2005.

KLEMENT, K. C. Propositional logic. *In*: **Internet Encyclopedia of Philosophy**. [*S.l.*]: University of Tennessee at Martin 2004.

KLIMEK, R. Visualization of logical formulas. *In*: 2018 FEDERATED CONFERENCE ON COMPUTER SCIENCE AND INFORMATION SYSTEMS (FEDCSIS). 2018. **Anais [...]** [*S.l.*]: IEEE, 2018. p. 419–424.

KWON, O.-H.; PARK, J.; RYU, S.-T. A note on the elimination of pure literals in propositional logic. **Journal of Logic and Computation**, Oxford University Press v. 18, n. 3, p. 345–357, 2008.

Le Berre, D. Exploiting the real power of unit propagation lookahead. **Electronic Notes in Discrete Mathematics**, v. 9„ p. 59–80, 2001. ISSN 1571-0653. LICS 2001 Workshop on Theory and Applications of Satisfiability Testing (SAT 2001). Disponível em: https://www.sciencedirect.com/science/article/pii/S1571065304003142.

LENSSEN, N.; LONSING, F.; BIERE, A. Augmenting vsids with domain-specific heuristics for sat-based planning. *In*: PROCEEDINGS OF THE 26TH INTERNATIONAL CONFERENCE ON THEORY AND APPLICATIONS OF SATISFIABILITY TESTING (SAT). 13958 de **Lecture Notes in Computer Science**., 2023. **Anais [...]** [*S.l.*]: Springer, 2023. p. 250–267.

LI, C. M.; ANBULAGAN. Heuristics based on unit propagation for satisfiability problems. *In*: PROCEEDINGS OF THE FIFTEENTH INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, IJCAI '97, NAGOYA, JAPAN, AUGUST 23-29, 1997. 1997. **Anais [...]** [*S.l.*: *s.n.*], 1997. p. 366–371.

LIANG, J. H. *et al.* Understanding vsids branching heuristics in conflict-driven clause-learning sat solvers. *In*: SPRINGER. PROCEEDINGS OF THE 18TH INTERNATIONAL CONFERENCE ON THEORY AND APPLICATIONS OF SATISFIABILITY TESTING (SAT). 2016. **Anais [...]** [*S.l.*], 2016. p. 225–241.

LYNCE, I. Propositional satisfiability: Techniques, algorithms and applications. **AI Communications**, SAGE Publications Sage UK: London, England„ p. 35, 2004.

MARQUES-SILVA, J.; LYNCE, I.; MALIK, S. Conflict-driven clause learning sat solvers. **Handbook of Satisfiability**, IOS Press„ p. 131–153, 2009.

MEERT, W.; VLASSELAER, J.; BROECK, G. Van den. A relaxed tseitin transformation for weighted model counting. *In*: PROCEEDINGS OF THE SIXTH INTERNATIONAL WORKSHOP ON STATISTICAL RELATIONAL AI (STARAI). 2016. **Anais [...]** [*S.l.*: *s.n.*], 2016. p. 1–7.

MORAIS, L. L. de. **SAT Solver baseado no algoritmo DPLL com heurísticas de decisão**. [*S.l.*]: , 2025. https://github.com/lara-morais/TCC. Acesso em: 18 jun. 2025.

MOSKEWICZ, M. W. *et al.* Chaff: Engineering an efficient sat solver. *In*: PROCEEDINGS OF THE 38TH DESIGN AUTOMATION CONFERENCE (DAC). 2001. **Anais [...]** [*S.l.*]: ACM, 2001. p. 530–535.

NIEUWENHUIS, R.; OLIVERAS, A.; TINELLI, C. Abstract DPLL and abstract DPLL modulo theories. *In*: BAADER, F.; VORONKOV, A. (Ed.). LOGIC FOR PROGRAMMING, ARTIFICIAL INTELLIGENCE, AND REASONING, 11TH INTERNATIONAL CONFERENCE, LPAR 2004, MONTEVIDEO, URUGUAY, MARCH 14-18, 2005, PROCEEDINGS. 2005. **Anais [...]** [*S.l.*]: Springer Berlin Heidelberg, 2005. p. 36–50. ISBN 978-3-540-22836-3.

PAIVA, N. d. N. **Análise e Conversão de Algoritmos Criptográficos para Forma Normal Conjuntiva**. fev. 2017. Dissertação (Dissertação de Mestrado) — Laboratório Nacional de Computação Científica – Programa de Pós-Graduação em Modelagem Computacional Petrópolis, RJ, Brasil fev. 2017.

RIBAS, B. C. **Um Método de Pré-Processamento de Fórmulas SAT e Pseudo-Boolean Baseado em Técnicas de Programação Linear Inteira Mista**. set. 2015. 161 p. Tese (Doutorado) — Universidade Federal do Paraná Curitiba set. 2015.

SHACHAM, O.; ZARPAS, E. K. Tuning the vsids decision heuristic for bounded model checking. *In*: PROCEEDINGS OF THE 5TH INTERNATIONAL WORKSHOP ON MICROPROCESSOR TEST AND VERIFICATION (MTV). 2003. **Anais [...]** [*S.l.*: *s.n.*], 2003. p. 75–80.

TSEITIN, G. S. On the complexity of derivation in propositional calculus. **Automation of reasoning: 2: Classical papers on computational logic 1967–1970**, Springer„ p. 466–483, 1983.

ZHANG, H.; STICKELY, M. E. An e fficient algorithm for unit propagation. **Proc. of AI-MATH**, v. 96„, 1996.