UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

Yuri Castro Neo de Carvalho

SAT e SMT solvers: fundamentos e aplicação ao problema de alocação de frequências

São Carlos
2019

YURI CASTRO NEO DE CARVALHO

SAT E SMT SOLVERS: FUNDAMENTOS E APLICAÇÃO AO PROBLEMA DE ALOCAÇÃO DE FREQUÊNCIAS

Trabalho de Conclusão de Curso apresentado à Escola de Engenharia de São Carlos, da Universidade de São Paulo

Curso de Engenharia Elétrica com ênfase em Eletrônica

Orientadores:
Prof. Dr. Rogério Andrade Flauzino (EESC)
Prof. Dr. Orlando de Andrade Figueiredo (UNESP/Rio Claro)

São Carlos
2019

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

Carvalho, Yuri Castro Neo de
C331s SAT e SMT solvers: fundamentos e aplicação ao problema de alocação de frequências / Yuri Castro Neo de Carvalho; orientador Orlando de Andrade Figueiredo; coorientador Rogério Andrade Flauzino. São Carlos, 2019.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2019.

1. SAT. 2. SMT. 3. Lógica. 4. Problema de Alocação de Frequência. 5. Z3. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

FOLHA DE APROVAÇÃO

Nome: Yuri Castro Neo de Carvalho

Título: "SAT e SMT solvers: fundamentos e aplicação ao problema de alocação de frequências"

Trabalho de Conclusão de Curso defendido e aprovado em 19 / 11 / 2019,

com NOTA 7,5 (sete , cinco), pela Comissão Julgadora:

Professor Assistente Doutor Orlando de Andrade Figueiredo - Orientador - UNESP/Campus Rio Claro

Prof. Associado Rogério Andrade Flauzino - SEL/EESC/USP

Prof. Dr. Maximiliam Luppe - SEL/EESC/USP

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Rogério Andrade Flauzino

“Contrariwise,' continued Tweedledee, 'if it was so, it might be; and if it were so, it would be; but as it isn't, it ain't. That's logic.”
Lewis Carroll, Through the Looking Glass

SUMÁRIO

Lista de Figuras

Lista de Abreviaturas e Siglas

BCP	*Boolean Constraint Propagation*
CDCL	*Conflict-Driven Clause Learning*
CNF	*Conjunctive Normal Form*
DNF	*Disjunctive Normal Form*
DPLL	Procedimento de Davis-Putnam-Logeman-Loveland
DPLL(T)	Procedimento de Davis-Putnam-Logeman-Loveland Procedure para a teoria T
DPP	Procedimento de Davis-Putnam
FBF	Fórmula bem formada
LIA	*Linear Integer Arithmetic*
NNF	*Negation Normal Form*
SAT	*Satisfiability / Satisfied*
SMT	*Satisfiability Modulo Theories*
UNSAT	*Unsatisfied*

Lista de Símbolos

$\top$	Verdadeiro
$\bot$	Falso
$\neg$	Negação
$\wedge$	Conjunção
$\vee$	Disjunção
$\bigwedge$	Conjunção n-ária
$\bigvee$	Disjunção n-ária
$\rightarrow$	Implicação
$\leftrightarrow$	Equivalência / Bi-implicação
$\vDash$	Satisfaz
$\nvDash$	Não satisfaz / Falsifica
$\mapsto$	Atribuição
$\forall$	Quantificador Universal
$\exists$	Quantificador Existencial

Lista de Tabelas

Resumo

Carvalho, Y. C. N. **SAT e SMT solvers: fundamentos e aplicação ao problema de alocação de frequência.** 2019. 70p. Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos.

O problema da satisfatibilidade booleana (Boolean Satisfiability, ou simplesmente SAT) é decidir se existe alguma interpretação para uma dada fórmula booleana de maneira que avaliação da fórmula seja verdadeiro. Desde seus primeiros algoritmos na década de 1960, os SAT solvers evoluíram muito, não só incorporando conceitos como aprendizado, mas também permitindo representações mais compactas e expressivas com o uso de solvers específicos para uma dada teoria, conhecido como Satisfiability Modulo Theories (SMT). Antes restritos a apenas algumas centenas de variáveis, SAT e SMT solvers alcançaram capacidade de lidar com problemas industriais, lidando com milhões de variáveis e restrições. Possuem aplicações práticas em diversas áreas como Electronic Design Automation (EDA), Verificação, Inteligência Artificial e Pesquisa Operacional. O presente trabalho fornece o embasamento teórico de lógica e a evolução dos principais algoritmos até atingir o estado-da-arte. No contexto de uma aplicação militar, existe o problema de se alocar frequências para pares de links de rádios (RLFAP), de maneira que as interferências sejam evitadas. Neste trabalho, o problema de RLFAP foi abordado utilizando-se com sucesso o Z3 SMT Solver da Microsoft e um conhecido conjunto de dados construído a partir de redes reais e fornecido pelo CELAR (Centro de Eletrônica do Exército Francês).

Palavras-chave: SAT, SMT, Lógica, Problema de Alocação de Frequências, Z3.

Abstract

Carvalho, Y. C. N. **SAT and SMT solvers: fundamentals and application to the problem of frequency assignment.** 2019. 70p. Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos

The problem of Boolean Satisfiability (or simply SAT) is deciding if there exists an interpretation for a given boolean formula such that all clauses are satisfied. Since its first algorithms in the 1960s, SAT solvers have evolved a lot, incorporating not only concepts such as learning, but also allowing for more compact and expressive representations with the use of specific solvers for a given theory, known as Satisfiability Modulo Theories (SMT). What once could only handle hundreds of variables, SAT and SMT solvers have reached industrial capability, now being able to handle millions of variables and constraints. Their practical applications range from Electronic Design Automation (EDA), Verification, Artificial Intelligence and Operations Research. This work provides a background in logic and the evolution of the main algorithms until reaching the state-of-the-art. In the context of a military application, the problem of assigning frequencies to pairs of radio links (RLFAP), such that interference is avoided, is successfully explored by using Microsoft's Z3 SMT Solver with a well-known real-world dataset provided by CELAR (French Army Electronics Center).

Keywords: SAT, SMT, Logic, Frequency Assignment Problem, Z3.

1. Introdução

1.1. Motivação

O problema da satisfatibilidade booleana, ou simplesmente SAT, consiste em verificar se uma dada fórmula contendo conectivos lógicos e variáveis binárias possui alguma combinação de valores (*True/False*, 0/1) para os quais todas componentes da fórmula (cláusulas) são satisfeitas. Isto é, ao substituir os valores dessa interpretação, a fórmula avalia para *True*/1. Tais fórmulas podem ser utilizadas para descrever propriedades de sistemas e, utilizando algum *solver*, verificar se tal sistema se comporta de acordo com uma especificação.

Contudo, há domínios de problemas nos quais a expressividade de uma codificação binária não é intuitiva, podendo levar a erros e perda de eficiência. Para contornar tal problema, os SAT *solvers* foram expandidos para interagir com procedimentos que decidem a satisfatibilidade de uma fórmula em relação a outras teorias como Aritmética de Inteiros. Chamados de SMT *(Satisfiability Modulo Theories)*, permitem a descrição de sistemas e restrições utilizando notações mais expressivas e intuitivas.

No campo da engenharia, SAT *solvers* apresentam inúmeras aplicações em problemas de Automação de Projeto Eletrônico, sendo capaz de resolvê-los de maneira eficiente, como lista [1]:

- *Equivalence Checking*: provar que dois circuitos produzem a mesma saída para qualquer entrada
- *Model Checking*: verifica se um modelo satisfaz as especificações, utilizado na *Intel* desde os primeiros modelos Pentium, após uma falha na unidade de ponto flutuante que resultou em um prejuízo de 475 milhões de dólares [2, 3]
- *Delay Fault Testing:* busca por padrões em falhas de circuitos causadas por atrasos nas portas lógicas
- *Logic Synthesis:* síntese de circuitos lógicos que satisfazem uma dada especificação
- *FPGA routing:* busca de rotas em circuitos de uma FPGA
- *Redundancy Identification:* busca por partes redundantes em um circuito

Outro campo no qual SAT *solvers* atuam buscando inconsistências é em *software*:

- Verificação de drivers e controladores [4]
- Verificação de controle de acesso a recursos na plataforma de computação em nuvem *Amazon Web Services* [5]

SAT e SMT *solvers* também são utilizados em problemas combinatórios, nos quais se tem um conjunto de variáveis e restrições, buscando um modelo que faça satisfaça todas as restrições, ou pelo menos minimize a quantidade das violadas. A otimização de material utilizado em gráfica de grande porte [6] e o problema de agendamento de ligas esportivas profissionais européias [7] são alguns exemplos.

Apesar das várias aplicações, é preciso notar que SAT é um problema NP-Completo [8], para os quais as soluções que conhecemos atualmente possuem, no pior dos casos, tempo computacional exponencial em relação ao número de variáveis. Todavia, de acordo com [9], deve-se enfatizar que o pior dos casos é exponencial, de maneira que há casos em domínios de problemas específicos nos quais isso não ocorre, e são nesses casos que os SAT e SMT solvers são explorados.

A solidificação da base teórica e prática levou, a partir dos anos 2000, a um crescimento no número de SAT e SMT *solvers* no meio acadêmico, o que levou a criação da *SAT Competitions* (2002) [9] e *SMTCOMP* (2005) [10], competições nas quais são utilizados conjuntos de dados industriais reais.

Problemas da classe NP-Completo podem ser reescritos em termos de SAT [11], permitindo casos como o da coloração de grafos, no qual deve-se atribuir cores para os vértices a partir de um conjunto finito, de maneira que vértices vizinhos não possuam a mesma cor. Uma variação desse caso é o problema de se alocar frequências (ou canais) a células de uma rede de telecomunicação, de maneira que interferências sejam evitadas ou mitigadas.

1.2. Objetivos

Este trabalho tem como objetivos:

- realizar um levantamento sobre a evolução dos principais métodos e algoritmos utilizados por SAT e SMT *solvers* na literatura, fornecendo a fundamentação teórica de Lógica Proposicional e de Primeira Ordem, necessária para modelar sistemas;
- como aplicação, é utilizado o conhecido e premiado *solver* Z3 da Microsoft em um problema de alocação de frequências.

1.3. Organização do trabalho

O trabalho é composto por 5 capítulos. Neste foram apresentadas as motivações para o estudo de SAT e SMT solvers a partir de suas vastas aplicações. O capítulo 2 introduz os fundamentos teóricos da Lógica Proposicional e de Primeira Ordem, bem como os principais procedimentos de decisão e a definição do problema de alocação de frequências. O capítulo 3 trata do conjunto de dados (materiais) e métodos (codificação e *solver* utilizado). O capítulo 4 apresenta os resultados apresentados e o 5 as conclusões.

2. Fundamentos Teóricos

2.1. Lógica Proposicional e de Primeira Ordem

2.1.1. Introdução

Em sua tese de mestrado, intitulada *A Symbolic Analysis of Relay and Switching Circuits*, Claude Shannon [12] demonstrou como era possível utilizar a álgebra de Boole para analisar circuitos comutativos. Anos mais tarde ele foi além e forneceu uma definição para o conceito de informação, de maneira semelhante àquela que define entropia em física, tornando possível o estudo de sua transmissão por meios de comunicação, os quais podem apresentar interferências.

Apesar de neste trabalho abordarmos um problema de comunicação, mais especificamente tentando evitar ou mitigar as interferências, faz-se necessário compreender a importância da análise simbólica utilizada por Shannon.

George Boole publicou em 1847 um panfleto intitulado *The Mathematical Analysis of Logic* [13], no qual ele afirma que o uso da Álgebra Simbólica não depende dos símbolos utilizados, e sim das regras que definem a combinação dos mesmos. Portanto, é possível utilizar uma notação algébrica para descrever as propriedades elementares de conjuntos, e a partir disso, formular problemas sobre números, geometria , dinâmica, óptica, entre outros, o que Boole chamou de "Cálculo da Lógica".

Dessa maneira, este capítulo se dedica a fornecer os fundamentos das lógicas Proposicional e de Primeira Ordem, as quais serviram de base para modelar um problema de engenharia.

2.1.2. Sintaxe da Lógica Proposicional

A sintaxe de uma linguagem lógica consiste em um conjunto de símbolos e regras usadas para combiná-los e formar fórmulas (ou "sentenças") [14]. Os seguintes elementos compõem o alfabeto da Lógica Proposicional :

- Símbolos de pontuação : (e)
- Valores-verdade : $\top$ (verdadeiro) e $\bot$ (falso)
- Variáveis proposicionais : $x_1, x_2, \ldots, x_n$
- Átomos : $\top, \bot, x$ (variável)
- Literais : a , $\neg a$ (um átomo ou sua negação)
- Conectivos :
  $\neg$(*negação*), $\wedge$ (*conjunção*), $\vee$ (*disjunção*), $\rightarrow$ (*implicação*, *se ... então*), $\leftrightarrow$ (*se somente se ...*)
- Fórmulas: a , $\neg f_1$, $f_1 \wedge f_2$, $f_1 \vee f_2$, $f_1 \rightarrow f_2$, $f_1 \leftrightarrow f_2$ (um átomo ou conectivos aplicados à fórmulas)

Note que nem toda combinação de símbolos da linguagem é válida, assim damos o nome de fórmula bem formada (fbf) àquelas que seguem as seguintes regras de formação indutiva:

1. Toda proposição isolada p é uma fbf
2. Se p é uma fbf, então $\neg p$ também é uma fbf
3. Se p e q são fbfs então $(p \vee q)$, $(p \wedge q)$, $(p \rightarrow q)$ *e* $(p \leftrightarrow q)$ são fbfs

Podemos classificar os conectivos em relação a quantos argumentos o mesmo utiliza. O conectivo de negação $(\neg)$ necessita de um único termo (unário), enquanto os outros necessitam de dois (binário).

Algumas fórmulas podem ser decompostas, então caso G ocorra sintaticamente em F dizemos que G é subfórmula de F. Considera-se que toda fórmula é subfórmula de si mesma.

Ademais, é possível definir uma ordem para as subfórmulas : F_1 precede F_2 caso F_1 seja sub fórmula de F_2.

2.1.3. Semântica da Lógica Proposicional

A semântica de uma lógica é aquilo que fornece seu significado [14], que em Lógica Proposicional é dado pelos valores constantes $\top$ e $\bot$. Assim, para se determinar o significado de uma fórmula é preciso avaliá-la dentro de um contexto, que recebe o nome de interpretação da fórmula, e corresponde a um conjunto de valores a serem atribuídos às variáveis das fórmulas [14], únicos para variável proposicional. Caso uma interpretação I de uma fórmula F resulte em verdadeiro, dizemos que a mesma satisfaz a fórmula.

De maneira compacta, podemos reformular utilizando os seguintes símbolos :

$$I \vDash F \quad (a\ interpreta\text{ção}\ I\ satisfaz\ a\ f\text{órmula}\ f)$$
$$I \nvDash F \quad (a\ interpreta\text{ção}\ I\ falsifica\ a\ f\text{órmula}\ f)$$

Supondo uma fórmula $F : p \wedge q \rightarrow p \vee \neg q$, uma possível interpretação é:

$$I : \{ p \mapsto \bot,\ q \mapsto \bot \}$$

Por outro lado, se uma fórmula F é satisfazível por todas as possíveis interpretações I, diz-se que a fórmula é válida. Podemos então omitir I e escrever apenas $\vDash F$. Note que são características duais : uma fórmula F só é válida se sua negação $\neg F$ não for satisfazível.

Uma tabela-verdade oferece a maneira mais simples de verificar se uma interpretação satisfaz ou mesmo valida uma fórmula, e para construí-la utilizamos como ponto de partida as tabelas-verdade dos conectivos.

Tabela 1. Tabela-verdade dos conectivos proposicionais

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$
$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\top$	$\top$
$\top$	$\bot$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$
$\bot$	$\top$	$\top$	$\bot$	$\top$	$\top$	$\bot$
$\top$	$\top$	$\bot$	$\top$	$\top$	$\top$	$\top$

Fonte: elaborada pelo autor

Note também que, se uma fórmula F contém n proposições, então existem 2^n possíveis interpretações. Como o número de linhas cresce exponencialmente com o de variáveis, a construção de tabelas-verdade torna-se um método ineficiente.

Para verificar se uma interpretação satisfaz (ou não) uma fórmula, aplica-se um conjunto de regras para manipular os símbolos apresentados anteriormente. No caso da Lógica Proposicional, o cálculo utiliza as seguintes regras:

1. $I \vDash \top$
2. $I \nvDash \bot$
3. $I \vDash x$ *se e somente* $I[x] = \top$ (*I atribui* $\top$ à x)
4. $I \vDash \neg F_1$ *se e somente se* $I \nvDash F_1$
5. $I \vDash F_1 \wedge F_2$ *se e somente se* $I \vDash F_1$ *e* $I \vDash F_2$
6. $I \vDash F_1 \vee F_2$ *se e somente se* $I \vDash F_1$ *ou* $I \vDash F_2$

7. $I \vDash F_1 \rightarrow F_2$ *se e somente se* $I \nvDash F_1$ *ou* $I \vDash F_2$
8. $I \vDash F_1 \leftrightarrow F_2$ *se e somente se* $I \vDash F_1$ *e* $I \vDash F_2$, *ou* $I \nvDash F_1$ *e* $I \nvDash F_2$

Suponha a seguinte fórmula $F : p \wedge q \rightarrow p \vee \neg q$ e a interpretação $I : \{ p \mapsto \top, q \mapsto \bot \}$. Substituindo os valores fornecidos em I pode-se facilmente montar uma tabela verdade para avaliar *F*, e verificar que a mesma é satisfeita por *I*:

Tabela 2. Tabela-verdade parcial da fórmula $F : p \wedge q \rightarrow p \vee \neg q$

p	q	$\neg q$	$p \wedge q$	$p \vee \neg q$	F
$\top$	$\bot$	$\top$	$\bot$	$\top$	$\top$

Fonte: elaborada pelo autor

Contudo, para determinar se *F* é válida é necessário completar a tabela com todos as possíveis combinações de p e q :

Tabela 3. Tabela-verdade completa da fórmula $F : p \wedge q \rightarrow p \vee \neg q$

p	q	$\neg q$	$p \wedge q$	$p \vee \neg q$	F
$\bot$	$\bot$	$\top$	$\bot$	$\top$	$\top$
$\bot$	$\top$	$\bot$	$\bot$	$\bot$	$\top$
$\top$	$\bot$	$\top$	$\bot$	$\top$	$\top$
$\top$	$\top$	$\bot$	$\top$	$\top$	$\top$

Fonte: elaborada pelo autor

Pode-se assegurar, portanto, que *F* é válida.

Uma alternativa para avaliar a validade de uma fórmula consiste em tomar uma abordagem sintática, provando-se por dedução ou refutação, a partir de regras de inferência. Regras de inferência

relacionam as premissas (ou antecedentes) com as deduções (ou consequentes), separando-as com uma linha vertical, com as premissas acima da linha e as conclusões abaixo [14].

Assim, partindo da semântica dos conectivos, podemos estabelecer as seguintes regras (note que dependendo da premissa, é necessário fazer considerar cada caso separadamente):

- $\frac{I \vDash \neg F}{I \nvDash F}$ e $\frac{I \nvDash \neg F}{I \vDash F}$
- $\frac{I \vDash F \wedge G}{I \vDash F \quad I \vDash G}$ e $\frac{I \nvDash F \wedge G}{I \nvDash F \mid I \nvDash G}$
- $\frac{I \vDash F \vee G}{I \vDash F \mid I \vDash G}$ e $\frac{I \nvDash F \wedge G}{I \nvDash F \quad I \nvDash G}$
- $\frac{I \vDash F \rightarrow G}{I \nvDash F \mid I \vDash G}$ e $\frac{I \nvDash F \rightarrow G}{I \vDash F \quad I \nvDash G}$
- $\frac{I \vDash F \leftrightarrow G}{I \vDash F \wedge G \mid I \nvDash F \vee G}$ e $\frac{I \nvDash F \leftrightarrow G}{I \vDash F \wedge \neg G \mid I \vDash \neg F \wedge G}$
- $\frac{I \vDash F \quad I \nvDash F}{\bot}$ (contradição)

Considere a seguinte fórmula $F : p \wedge q \rightarrow p \vee \neg q$. Para estabelecer a validade de *F* primeiro assumimos que ela é inválida $I \nvDash F$, ou seja, existe uma interpretação *I* que satisfaz a fórmula. Em seguida, utilizando as regras de inferência para construir argumentos [14] :

1. $I \nvDash p \wedge q \rightarrow p \vee \neg q$ premissa
2. $I \vDash p \wedge q$ utilizando 1 e a semântica de $\rightarrow$
3. $I \nvDash p \wedge \neg q$ utilizando 1 e a semântica de $\rightarrow$
4. $I \vDash p$ utilizando 2 e a semântica de $\wedge$
5. $I \nvDash p$ utilizando 3 e a semântica de $\vee$

Observando 4 e 5, pode-se concluir que temos uma contradição, e a prova pode ser encerrada, comprovando que $\neg F$ é, de fato, válida.

2.1.4. Sintaxe da Lógica de Primeira Ordem

A Lógica de Primeira Ordem estende a Proposicional com predicados, funções e quantificadores [14]. Com isso, ganha-se expressividade para fórmulas cujas variáveis podem ser números inteiros, reais, *arrays*, vetores de *bits*, entre outros.

O alfabeto da Lógica de Primeira Ordem é formado por:

- Símbolos de pontuação : (e)
- Variáveis : x, y, z
- Constantes : a, b, c

- Predicados : p, q, r
- Variáveis proposicionais (predicados de aridade 0) : P, Q, R
- Funções: f, g, h
- Conectivos proposicionais: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$
- Quantificadores : $\forall, \exists$
- Átomos : $\bot, \top,$ p (*predicados*)
- Literais : átomos ou sua negação
- Fórmulas : literais, aplicação de conectivos à fórmulas, aplicação de quantificadores à fórmulas

As funções adicionam expressividade ao permitir o uso de domínios que não sejam apenas os valores-verdade $\bot$ e $\top$. Podem variar na quantidade de argumentos que utilizam (aridade), sendo que as de aridade 0 podem ser vistas como constantes. Dessa maneira, pode-se trabalhar com domínios de interesse como naturais, inteiros e reais.

Predicados, assim como as funções, variam em relação à aridade, mas quando avaliados mapeiam valores para o domínio booleano.

As fórmulas em Lógica de Primeira Ordem também seguem uma regra indutiva na definição de fórmulas bem formadas e expandem aquelas da Lógica Proposicional:

1. Todo literal l é uma fbf
2. A aplicação dos conectivos às fbfs é uma fbf
3. A aplicação dos quantificadores a uma fbf é uma fbf

Os quantificadores universal ($\forall$) e existencial ($\exists$) podem ser vistos como operadores que aplicam um predicado em todo um domínio. O quantificador universal somente retorna $\top$ caso todos os elementos do domínio validem o predicado, já o quantificador existencial retorna $\top$ caso pelo menos um elemento do domínio valide o predicado.

2.1.5. Semântica da Lógica de Primeira Ordem

Com a adição de funções, pode-se trabalhar com diversos domínios além do booleano, assim é necessário incluir o domínio D_I da interpretação. Trata-se de um conjunto não-vazio de valores como inteiros, dias da semana, etc. Podendo ser finitos, como um baralho de 52 cartas, ou incontavelmente finito como os números reais [14].

Assim, uma interpretação $I : (D_I, \alpha_I)$ é um par formado pelo domínio D_I e atribuições (α_I), mapeando constantes, funções e predicados para elementos, funções e predicados do domínio D_I.

Considerando a fórmula $F : x + y > z \rightarrow y > z - x$, podemos ver que ela é composta de símbolos $+, -, >$, os quais são escolhidos para dar uma intuição sobre o significado proposto para a fórmula. Note que a mesma pode ser reescrita como $F' : p(f(x,y),z) \rightarrow p(y,g(z,x))$.

Portanto, se considerarmos:

- $D_I = \mathbb{Z}$
- $\alpha_I = \{ + \mapsto +_{\mathbb{Z}}, - \mapsto -_{\mathbb{Z}}, > \mapsto >_{\mathbb{Z}}, x \mapsto 13_{\mathbb{Z}}, y \mapsto 42_{\mathbb{Z}}, z \mapsto 1_{\mathbb{Z}} \}$

Podemos avaliar se a interpretação *I* satisfaz a fórmula *F* avaliando recursivamente os termos das fórmulas e utilizando as mesmas regras da Lógica Proposicional :

1. $I \vDash x + y > z$ pois $\alpha_I [x + y > z] = 13_{\mathbb{Z}} +_{\mathbb{Z}} 42_{\mathbb{Z}} >_{\mathbb{Z}} 1_{\mathbb{Z}}$
2. $I \vDash y > z - x$ pois $\alpha_I [y > z - x] = 42_{\mathbb{Z}} >_{\mathbb{Z}} 1_{\mathbb{Z}} -_{\mathbb{Z}} 13_{\mathbb{Z}}$
3. $I \vDash F$ a partir de 1, 2 e a semântica de $\rightarrow$

No caso de fórmulas que contêm quantificadores, consideramos variantes da interpretação, denotando por $J : I \lhd \{ x \mapsto v \}$ a variante de *I* na qual $\alpha_J[x] = v$ para algum $v \in D_I$, e utilizamos as seguintes regras:

1. $I \vDash \forall x. F$ se e somente se para todo $v \in D_I$, $I \lhd \{ x \mapsto v \} \vDash F$
2. $I \vDash \exists x. F$ se e somente se existe $v \in D_I$ de maneira que $I \lhd \{ x \mapsto v \} \vDash F$

Apesar da introdução dos quantificadores, iremos restringir a atenção para fragmentos não quantificados de alguma teoria T de primeira ordem.

Uma teoria de primeira ordem é um par composto de uma assinatura Σ, um conjunto de constantes, símbolos de funções e predicados, e o conjunto de axiomas A, um conjunto de fórmulas nas quais só aparecem elementos presentes em Σ, e que fornece o significado de seus símbolos [14].

Dessa maneira, pode-se definir a validade e satisfatibilidade de uma *F* construída a partir de Σ em relação a uma teoria *T* observando se toda interpretação *I* que satisfaz os axiomas A de T também satisfaz *F* (validade), ou se existe pelo menos uma interpretação *I* que satisfaz *F* (satisfatibilidade).

2.1.6. Formas Normais

Existem três formas normais (canônicas) de se representar fórmulas em Lógica Proposicional e Lógica de Primeira Ordem, variando em relação ao modo como os conectivos são utilizados.

A *Negation Normal Form* (NNF) ou Forma Normal da Negação requer que apenas os conectivos $\neg$, $\wedge$, $\vee$ sejam utilizados, e negações só aparecem em literais. Por exemplo $F : \neg(\neg P \vee \neg(P \wedge Q))$ está escrita em NNF.

Pode-se transformar uma fórmula F em uma equivalente F' utilizando as seguintes regras recursivamente, da esquerda para a direita, onde o símbolo $\Leftrightarrow$ representa equivalência [14]:

1. $\neg\neg F \Leftrightarrow F$
2. $\neg\top \Leftrightarrow \bot$
3. $\neg(F_1 \wedge F_2) \Leftrightarrow \neg F_1 \vee \neg F_2$ (De Morgan)
4. $\neg(F_1 \vee F_2) \Leftrightarrow \neg F_1 \wedge \neg F_2$ (De Morgan)
5. $F_1 \rightarrow F_2 \Leftrightarrow \neg F_1 \vee F_2$
6. $F_1 \leftrightarrow F_2 \Leftrightarrow (F_1 \rightarrow F_2) \wedge (F_2 \rightarrow F_1)$

A *Disjunctive Normal Form* (DNF) ou Forma Normal Disjuntiva é formada por disjunções de conjunções de literais, isto é $\bigvee_i \bigwedge_j l_{i,j}$.

Por exemplo, $F : (Q_1 \wedge R_1) \vee (Q_1 \wedge R_2) \vee (Q_2 \wedge R_1) \vee (Q_2 \wedge R_2)$ está escrita em DNF.

A conversão para DNF consiste em primeiro transformá-la em NNF e em seguida aplicar as seguintes regras de equivalência, também da esquerda para a direita:

1. $(F_1 \vee F_2) \wedge F_3 \Leftrightarrow (F_1 \wedge F_3) \vee (F_2 \wedge F_3)$
2. $F_1 \wedge (F_2 \vee F_3) \Leftrightarrow (F_1 \wedge F_2) \vee (F_1 \wedge F_3)$

Dualmente, temos a *Conjunctive Normal Form* (CNF) ou Forma Normal Conjuntiva, que consiste em uma conjunção de disjunções $\bigwedge_i \bigvee_j l_{i,j}$, onde cada bloco de disjunção é chamado de cláusula. Por exemplo $F : (Q_1 \vee Q_2) \wedge (Q_2 \vee Q_3)$, está em CNF e $(Q_1 \vee Q_2)$ e $(Q_2 \vee Q_3)$ são suas cláusulas.

As regras de equivalência para converter uma fórmula *F* em CNF, utilizadas após a conversão para NNF, são :

1. $(F_1 1 \wedge F_2) \vee F_3 \Leftrightarrow (F_1 \vee F_3) \wedge (F_2 \vee F_3)$
2. $F_1 \vee (F_2 \wedge F_3) \Leftrightarrow (F_1 \vee F_2) \wedge (F_1 \vee F_3)$

SAT/SMT solvers utilizam CNFs como formato padrão de entrada. Apesar de ser possível converter uma fórmula arbitrária para DNF, e esse ser um formato no qual decidir a satisfatibilidade pode ser feito em tempo linear [15], a transformação pode aumentar exponencialmente o tamanho da fórmula.

Por outro lado, a conversão em CNF utilizando o método de transformação de Tseiting [16] aumenta o tamanho da fórmula linearmente, através da adição de novas variáveis.

2.2. Procedimentos de Decisão

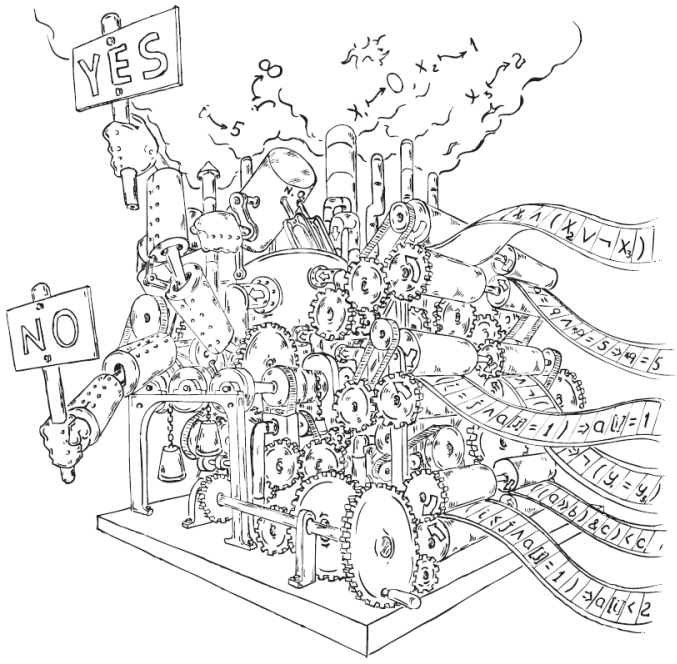
2.2.1. Introdução

O problema de se verificar a satisfatibilidade de uma fórmula proposicional é comumente abreviado para *SAT*. Programas que implementam procedimentos para resolver esse problema são chamados *SAT solvers*. Caso a fórmula utilize uma teoria *T* de primeira ordem, os *solvers* recebem o nome de *SMT (Satisfiability Modulo Theories)*. Esta seção se dedica a explorar maneiras (procedimentos) para solucionar essas duas versões do problema.

Existem algumas propriedades que gostaríamos que tais procedimentos tivessem. A primeira (*soundness)* é que, dada uma fórmula arbitrária válida, o procedimento sempre retorne "válida". A segunda (*completeness*) requer que o procedimento garantidamente termine. Assim, os procedimentos que apresentam as duas propriedades para qualquer fórmula de uma teoria *T* são chamados de procedimentos de decisão. [17] aponta que existe uma confusão enraizada na literatura, já que assume-se que um procedimento não necessariamente termine, enquanto um algoritmo sim, de modo que faria mais sentido chamar de "algoritmo de decisão".

Apesar de sempre buscarmos utilizar procedimentos completos, isso nem sempre é possível. Por outro lado, em vários casos práticos, procedimentos incompletos acabam terminando [17]. A figura abaixo, extraída de [17] ilustra um procedimento de decisão como um engenhoso e complexo processo que, no final das contas, decide se uma fórmula (representada pelas fitas à direita) é decidível ou não.

Fig. 1. Ilustração de um Procedimento de Decisão



Fonte: [17, p. VIII]

2.2.2. Procedimento de Davis-Putnam (DPP)

O primeiro procedimento surge em um manuscrito não publicado para a Agência Nacional de Segurança americana (NSA) [18], no qual Davis e Putnam sugerem os seguintes princípios, assumindo uma fórmula em CNF:

1. Regra da cláusula unitária: caso exista uma cláusula contendo um único literal (l), remova todas as cláusulas contendo l, e todos os literais $\neg l$.
2. Regra do literal puro: caso um literal l apareça em alguma cláusula mas a sua negação $\neg l$ não, remova todas as cláusulas contendo l.
3. Regra da resolução: caso duas cláusulas contenham um mesmo literal v, mas divirjam em relação à sua polaridade, remova o literal e forme uma nova disjunção a partir dos outros literais restantes, isto é , substitua

$$(v \vee l_1 \vee l_2) \wedge (\neg v \vee l_3 \vee l_4)$$

por

$$(l_1 \vee l_2 \vee l_3 \vee l_4)$$

As regras devem ser aplicadas recursivamente até que não haja mais cláusulas, o que significa que a fórmula é satisfatível, ou até derivar a cláusula vazia, no qual a fórmula é insatisfatível.

Esse procedimento, apesar de ter *soundness*, não é completo, e além disso a regra da resolução acaba utilizando muita RAM quando implementada [18], uma forte restrição na época.

2.2.3. Davis-Putnam-Logemann-Loveland (DPLL)

Em 1962, os programadores Logemann e Loveland foram contratados por Davis e Putnam para trabalhar no algoritmo, e forneceram sugestões mais eficientes para o cálculo da satisfatibilidade. O resultado, comumente referenciado por DPLL, se tornou a base para solvers modernos.

A sugestão dos programadores foi restringir o uso da resolução para o caso de cláusulas unitárias, o qual recebeu o nome de *Boolean Constraint Propagation* (BCP). Suponha que em uma fórmula F exista uma cláusula unitária (l) e que também exista uma cláusula C na qual l aparece como $\neg l$, assim, pode-se substituir a cláusula C pelo subconjunto que não contém $\neg l$. Isto é:

$$\frac{l \quad C[\neg l]}{C[\bot]}$$

Suponha que $F : (P) \wedge (\neg P \vee Q) \wedge (R \vee \neg Q \vee S)$. Pode-se aplicar BCP nas primeiras duas cláusulas :

$$\frac{P \quad (\neg P \vee Q)}{Q}$$

Que resulta em $F' : (Q) \wedge (R \vee \neg Q \vee S)$. Podemos continuar aplicando BCP :

$$\frac{Q \quad (R \vee \neg Q \vee S)}{R \vee S}$$

Resultando em $F'' : (R \vee S)$.

Caso a fórmula não seja reduzida para $\bot$ ou $\top$, basta escolher uma variável x e aplicar BCP para as atribuições $x \mapsto \top$ e $x \mapsto \bot$. O algoritmo pode ser expresso em pseudocódigo como [19]:

Entrada: fórmula proposicional em CNF

Saída: “SAT” caso a fórmula for satisfatível, “UNSAT” caso contrário

---

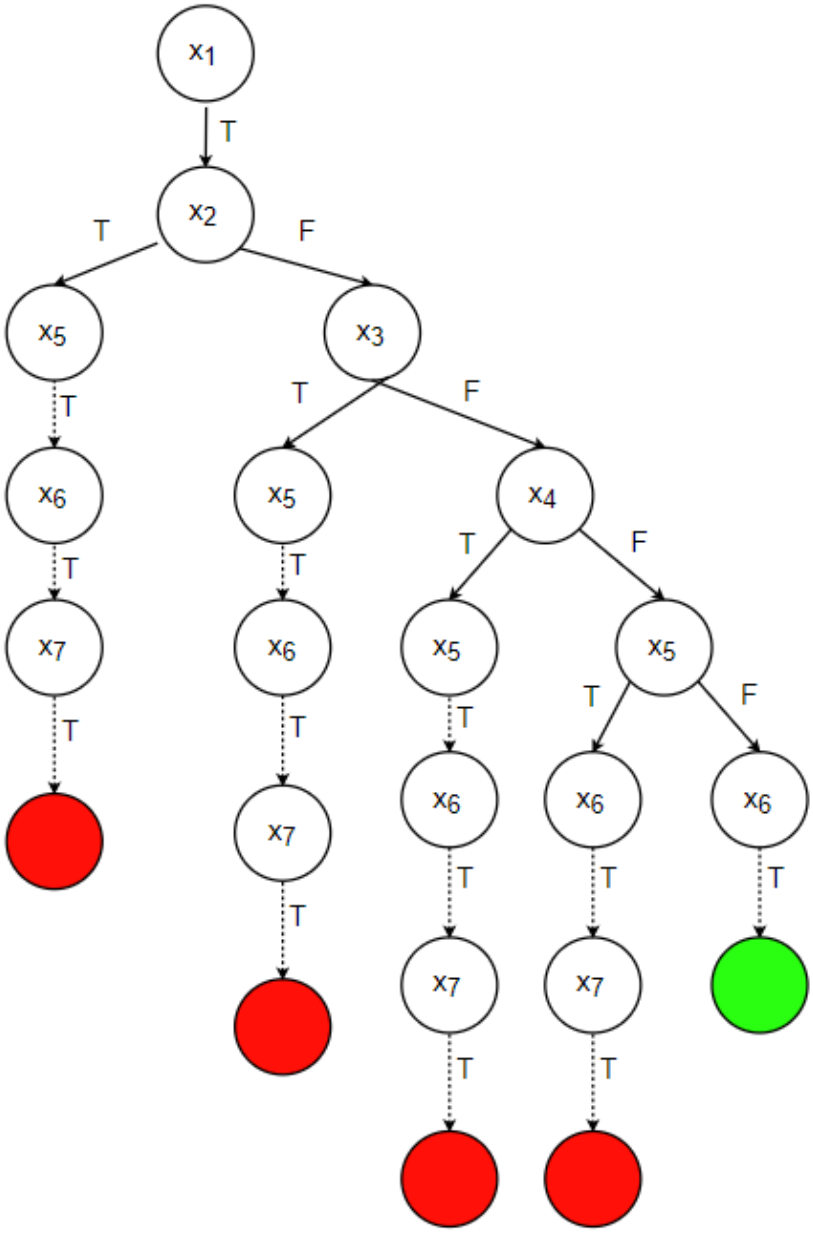
```
DPLL(F):
      F'= BCP(F)                              (1)
      IF F' = ⊤ RETURN "SAT"                  (2)
     IF F' = ⊥ RETURN "UNSAT"                 (3)
     X = CHOOSE(VARS(F'))                     (4)
     RETURN DPLL(F' ∧ X) || DPLL(F' ∧ ¬X)     (5)
```

O DPLL pode ser compreendido como uma busca em uma árvore binária, cujas ramificações são atribuições dos valores $\top$ e $\bot$ para cada variável na fórmula. Caso uma atribuição termine em um modelo que não satisfaz a fórmula, o algoritmo volta um nível de decisão e atribui um outro valor. A figura a seguir ilustra a árvore gerada durante a execução do DPLL para uma fórmula composta pelas seguintes cláusulas :

- $C_1 = (\neg x_2 \vee \neg x_3 \vee \neg x_4 \vee x_5)$
- $C_2 = (\neg x_1 \vee \neg x_5 \vee x_6)$
- $C_3 = (\neg x_5 \vee x_7)$
- $C_4 = (\neg x_1 \vee \neg x_6 \vee \neg x_7)$
- $C_5 = (\neg x_1 \vee \neg x_2 \vee x_5)$
- $C_6 = (\neg x_1 \vee \neg x_3 \vee x_5)$
- $C_7 = (\neg x_1 \vee \neg x_4 \vee x_5)$
- $C_8 = (\neg x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5 \vee \neg x_6)$

As setas cheias representam atribuições definidas pela função *CHOOSE*, enquanto as setas pontilhadas representam deduções realizadas pela função *BCP*. Os nós vermelhos representam ramos da busca cujo modelo representa uma contradição, isto é, não satisfaz a fórmula. Já o nó verde representa um modelo que de fato satisfaz a fórmula.

Fig. 2 - Ilustração da execução do algoritmo DPLL.



Fonte: Adaptado de [21]

2.2.4. Conflict-Driven Clause Learning

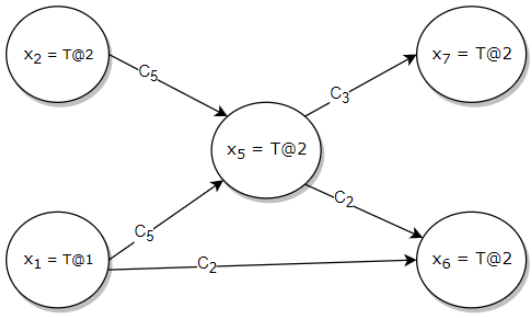
Em alguns casos, o algoritmo DPLL pode acabar perdendo tempo buscando em ramos que inevitavelmente irão falhar. Isso ocorre porque, ao encontrar um conflito, nenhuma informação é extraída e ele só desfaz um nível de decisão.

Pode-se construir um grafo a partir das decisões tomadas em cada nível e as implicações provocadas pelo BCP, e, através dele, identificar novas implicações que ajudarão a guiar o processo de busca, como proposto em [20].

Considere uma fórmula em CNF composta pelas mesmas cláusulas do exemplo anterior [21]. Como não há nenhuma cláusula unitária, não podemos utilizar BCP, assim escolhemos uma variável e atribuímos um valor, por exemplo $x_1 \mapsto \top$, e associamos ao nível de decisão 1 na forma $x_1 = \top@1$. Essa decisão não gera nenhuma cláusula para utilizar BCP, assim continua-se com uma nova atribuição $x_2 = \top@2$.

Observe que a segunda decisão faz com que a cláusula C_5 se torne unitária, implicando que a variável $x_5 = \top@2$. Após isso, a cláusula C_2 também se torna unitária, de modo que $x_6 = \top@2$. O mesmo ocorre em C_3, implicando $x_7 = \top@2$. Até o momento temos o seguinte grafo de implicação:

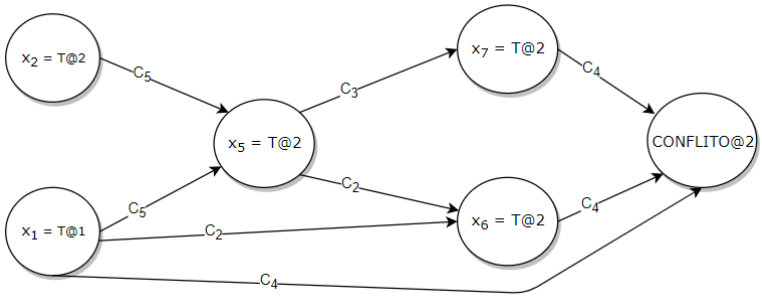
Fig. 3. Grafo de implicação parcial. Adaptado de [21]



Fonte:. Adaptado de [21]

Contudo, atribuindo $x_1 = \top@1$, $x_6 = \top@2$ e $x_7 = \top@2$ implica que a cláusula C_4 não é satisfeita, ou seja, ocorreu um conflito.

Fig. 4. Grafo de implicação com conflito.



Fonte: Adaptado de [21]

Em vez de simplesmente voltar em um nível de decisão, é possível analisar as cláusulas envolvidas no conflito partindo-se da cláusula envolvida no mesmo, C_4 nesse caso. Assim, visitam-se as variáveis implicadas via BCP no nível de decisão atual (2, no caso) mantendo dos antecedentes (cláusulas incidentes) as variáveis atribuídas em níveis de decisão anteriores ao mais recente através da aplicação da regra de resolução (ver seção 2.2.2), repetindo o processo até que a decisão mais recente seja visitada [22].

Dessa maneira, aplicando resolução nas cláusulas C_4 e C_3 obtemos a cláusula intermediária:

$$\frac{(\neg x_1 \vee \neg x_6 \vee \neg x_7) \quad (\neg x_5 \vee x_7)}{(\neg x_1 \vee \neg x_5 \vee \neg x_6)}$$

Em seguida. aplica-se resolução entre a cláusula intermediária e C_2, obtendo:

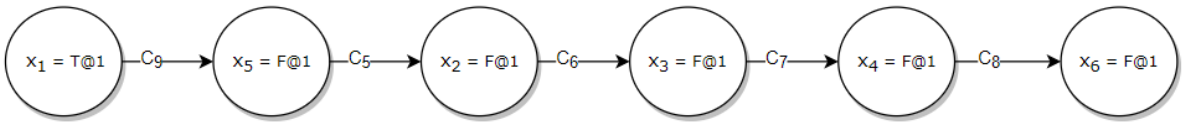
$$\frac{(\neg x_1 \vee \neg x_5 \vee \neg x_6) \quad (\neg x_1 \vee \neg x_5 \vee x_6)}{(\neg x_1 \vee \neg x_5)}$$

Como só resta um literal no nível de decisão 2, e x_1 possui o nível de decisão mais alto, volta-se ao nível de decisão 1, desfazendo as decisões do nível 2, e aplica-se a nova cláusula aprendida:

$$C_9 = (\neg x_1 \vee \neg x_5)$$

Como $x_1 = \top@1$, para satisfazer C_9 segue que $x_5 = \bot$. A partir disso, uma série de propagações são realizadas:

Fig. 5. Grafo de implicação final após propagações.



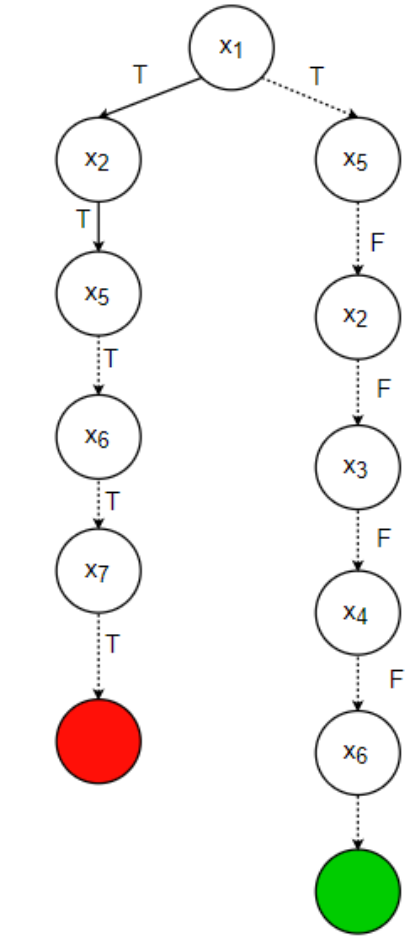
Fonte: Adaptado de [21]

Assim, conclui-se que a fórmula é satisfatível com a interpretação:

$$I : \{ x_1 \mapsto \top,\ x_2 \mapsto \bot,\ x_3 \mapsto \bot,\ x_4 \mapsto \bot,\ x_5 \mapsto \bot,\ x_6 \mapsto \bot \}$$

A figura a seguir ilustra a execução do algoritmo CDCL, note como o mesmo evita percorrer vários ramos devido ao aprendizado de cláusulas.

Fig. 6 - Ilustração da execução do algoritmo CDCL



.

Fonte: Adaptado de [21].

Seja F uma fórmula proposicional em CNF, V um conjunto de tuplas (x_i, v_i) associando variáveis e valores $\top$ ou $\bot$. O algoritmo, em pseudocódigo é [22] :

Entrada: fórmula proposicional F, interpretação V

Saída: *SAT* ou *UNSAT*

```
CDCL(F, V):
   IF BCP(F,V) ="CONFLICT" THEN RETURN "UNSAT"     (1)
   dl ← 0   (nível de decisão)                     (2)

   WHILE(NOT AllVariablesAssigned(F))              (3)
       DO (x,v) = PickBranchingVariable(F,V)       (4)
          dl  dl + 1                               (5)
          v  v ∪ { (x,v) }
          IF BCP(F,V) ="CONFLICT"                  (6)
          THEN β = ConflictAnalysis(F,V)           (7)
               IF β < 0 THEN RETURN "UNSAT"        (8)
               ELSE Backtrack(F,V,β)               (9)
                    dl ← β                         (10)
   RETURN "SAT"                                    (11)
```

Onde *AllVariablesAssigned* simplesmente verifica se já foram atribuídos valores para as variáveis e *PickBranchingVariable* faz o mesmo papel de *Choose* do DPP. *ConflictAnalysis* corresponde ao processo de identificar as causas do conflito, analisando as decisões feitas a partir do grafo de implicação, gerando novas cláusulas e o nível de decisão para o qual deve se retornar, que é realizado por *Backtrack*.

2.2.5. DPLL(T)

Apesar dos diversos avanços nos algoritmos, resolver problemas utilizando SAT é uma tarefa difícil, pois codificações eficientes não são triviais, o formato que os *solvers* recebem como entrada é feito especialmente para eles (o que dificulta a sua interpretação por parte das pessoas), e, além disso, a expressividade é baixa, o que induz a erros [23].

Uma solução, introduzida por [24], é estender o algoritmo DPLL (ou CDCL) para incorporar um procedimento de decisão de uma teoria T de primeira ordem como Aritmética Linear de Inteiros, Reais, Arrays, vetores de bits, etc.

Substituindo-se as cláusulas que contêm funções de outras teorias por variáveis proposicionais, a fórmula é primeiro analisada por um SAT solver, e, em seguida, o *solver* específico de cada teoria é invocado para verificar a consistência da cláusula de acordo com sua teoria.

Assim como no CDCL, a arquitetura DPLL(T) inclui ações como propagação (semelhante à BCP) e análise de conflitos, a partir das quais são geradas novas cláusulas que guiam o processo de busca.

Considere a Teoria da Aritmética Linear de Inteiros (*LIA - Linear Integer Arithmetic*), cuja assinatura é composta por $\Sigma_{LIA} = \{ +, -, <, \leq, >, \geq, \mathbb{Z} \}$ e cada interpretação avalia as funções no sentido usual (isto é, 1+1=2, 1 < 0 = falso, etc). Os seguintes exemplos ilustram a semântica de uma interpretação $I = \{ x \mapsto 7, y \mapsto 1 \}$ [25] :

- $I \vDash x > y + 5$ já que $x^I > y^I + 5$ resulta em $7 > 6$, que resulta em $\top$
- $I \nvDash y - x > 0$ já que $y^I - x^I > 0$ resulta em $-6 > 0$, que resulta em $\bot$

Seja $F : (x + 1 > 0 \vee x + y > 0) \wedge (x < 0 \vee x + y > 4) \wedge \neg(x + y > 0)$, o exemplo a seguir ilustra uma execução do *solver* instanciado com DPLL(*LIA)* :

O primeiro passo é fazer o codificação dos literais em termos de variáveis proposicionais:

$$F' : (A \vee B) \wedge (C \vee D) \wedge (\neg B)$$

Onde $A = x + 1 > 0$, $B = x + y > 0$, $C = x < 0$ e $D = x + y > 4$.

Seguindo o que ocorre no DPLL e CDCL, o próximo passo consiste em aplicar BCP, a qual atua no literal $\neg B$, unitário, implicando que $B = \bot@1$. Consequentemente, a primeira cláusula também se torna unitária, implicando $A = \top@1$. Como não há mais nada que possa ser deduzido via BCP, decide-se $C = \top@2$.

Uma vez decidido o valor proposicional, é preciso verificar a satisfatibilidade das atribuições até o momento em relação à *LIA* com o contexto até o momento, o qual identifica que : $x + 1 > 0$ e $x < 0$ não podem ser verdadeiros ao mesmo tempo. Portanto, adiciona-se a cláusula $(\neg A \vee \neg C)$ à fórmula e se reinicia o processo, já que só houve uma decisão. Assim, temos:

$$F'' : (A \vee B) \wedge (C \vee D) \wedge (\neg B) \wedge (\neg A \vee \neg C)$$

Novamente se aplica BCP, deduzindo que $B = \bot@1$ e $A = \top@1$. Em vez de cair no mesmo erro, agora a nova cláusula se torna unitária, implicando $C = \bot@1$. A cláusula $(C \vee D)$ também se torna unitária, implicando $D = \top@1$. Resta agora verificar a satisfatibilidade da fórmula em relação à *LIA*.

Dessa vez são os literais $\neg(x + y > 0)$ e $x + y > 4$ que não podem ser verdadeiros ao mesmo tempo, o que implica em uma nova cláusula $(\neg B \vee D)$. Contudo, como só houve trabalho feito via BCP e nenhuma decisão foi tomada, não havendo outro nível de decisão para o qual retornar, conclui-se que fórmula é *LIA-UNSAT*.

O procedimento de decisão utilizado para verificar a satisfatibilidade de uma fórmula- *LIA* é uma variação do algoritmo *Simplex*, o qual não necessita de uma função objetivo por não se tratar de um problema de otimização e, através da adição de novas variáveis, transforma o sistema de entrada numa forma normal composta de [17]:

- Igualdades da forma: $a_1 x_1 + \ldots + a_n x_n = 0$
- Limites superiores e inferiores: $l_i \leq x_i \leq u_i$ (opcionais)

Note que qualquer restrição linear do tipo $L \odot R$, onde $\odot \in \{ \leq, \geq, = \}$ pode ser convertida para a forma mencionada.

Assim como no método *Simplex* comum, constrói-se um *tableau* para verificar a satisfatibilidade dos limites impostos e manipular operações de pivotação. As variáveis originais do sistema são chamadas de básicas, enquanto aquelas adicionadas durante a transformação são chamadas de não-básicas.

O algoritmo pode então ser visto como [17]:

Entrada: um sistema linear de restrições S

Saída: *SAT* ou *UNSAT*

---

1. Transformar o sistema para a forma normal

   $Ax = 0$ e $\bigwedge_{i=1...m} l_i \leq s_i \leq l_m$

   onde $s_i \ldots s_m$ são as variáveis não-básicas
2. Construir o *tableau* de A
3. Determinar uma ordem fixa das variáveis
4. Caso nenhuma variável básica viole os limites, retorna *SAT*. Caso contrário escolhe-se a primeira variável básica x_i que apresenta violação
5. Busca-se uma variável não-básica x_j para pivotar com x_i, caso não exista, retorna *UNSAT*
6. Pivota-se x_i e x_j
7. Ir para o passo (4)

2.2.6. Otimizações

Além do aumento na velocidade de busca, fornecido pelo algoritmo CDCL, outras abordagens e sugestões foram feitas no início dos anos 2000 para tornar os *solvers* mais competitivos. As otimizações partem da observação que os *solvers* gastam, na prática, cerca de 90% do tempo efetuando BCP [26].

A partir disso, duas melhorias foram propostas, uma no campo da estrutura de dados utilizada pelos *solvers* e a utilização de heurística no momento de se escolher uma variável à qual nenhum valor ainda foi atribuído.

No campo da estrutura de dados, a técnica conhecida como *Two-Watched Literals* [26] propõe utilizar um contador por cláusula que informa a quantos literais atribuiu-se o valor $\perp$. Assim, se uma cláusula contém *N* literais, visita-se a mesma apenas quando o número de literais $\perp$ cair de *N-2* para *N-1*. Para isso, escolhem-se dois literais (ainda não atribuídos valor $\perp$), garantindo que até que um deles seja atribuído o valor $\perp$, não existem mais do que *N-2* literais $\perp$,

Essa técnica torna o processo de se reatribuir valores à variáveis mais rápido, pois os literais observados não precisam ser modificados quando ocorre uma mudança do nível de decisão atual para um anterior (*backtrack)*.

Já no campo da heurística, a técnica conhecida como *VSIDS (Variable State Independent Decay Sum)* propõe manter um contador para cada literal de cada polaridade, inicializando em 0. Quando uma cláusula é adicionada, o contador de cada um dos literais presente nela é incrementado e, no momento de se decidir o próximo literal, escolhe-se aquele com maior valor, enquanto periodicamente se divide os contadores por uma constante. A idéia é tentar satisfazer os conflitos mais recentes, especialmente porque, em problemas difíceis, as cláusulas conflitantes dominam o problema em número de literais.

Além disso, eventuais deleções de cláusulas (para evitar sobrecarga de memória) e reinicializações (mantendo parte das informações adquiridas previamente) também se mostraram boas estratégias para desenvolver *solvers* mais eficientes.

2.3. Radio Link Frequency Assignment Problem (RLFAP)

Segundo [27], o problema de alocação de frequências (*Frequency Assignment Problem, FAP)* ocorre em diferentes tipos de comunicações sem-fio, as quais têm que lidar com o fato de que o espectro de frequências é um recurso finito. Portanto, faz-se necessário balancear o reúso do espectro sem incorrer em perda de qualidade de comunicação por conta de interferências.

Segundo [28], tais interferências são influenciadas por:

- Potência dos transmissores
- Sensitividade dos receptores
- Ganho das antenas
- Distância
- Condições climáticas
- Frequências utilizadas
    - Restrição co-canal
    - Restrição de canal adjacente
    - Restrição co-local

Consideremos que, com exceção das frequências utilizadas, a maioria dos fatores não possam ser influenciados, focaremos a análise do problema nas restrições às escolhas de frequências utilizadas.

A restrição de co-canal ocorre quando não se pode alocar um mesmo canal para certos pares de células ao mesmo tempo, enquanto na restrição de canal adjacente, canais muito próximos não devem ser alocados simultaneamente. Finalmente, a restrição co-local estabelece que deve haver uma separação mínima entre dois canais alocados em uma mesma célula.

Tais restrições são utilizadas no contexto de uso militar de rádios de comunicação [29], no qual o FAP consiste em um conjunto X de links de rádio, para os quais, a cada $i \in X$, deve-se escolher uma frequência f_i a partir de um conjunto finito D_i disponível para cada transmissor. Assim, para cada par de links (i,j) as restrições podem ser dos tipos:

$$|f_i - f_j| > d_{ij}$$

$$|f_i - f_j| = \delta_{ij}$$

Onde d_{ij} representa uma distância mínima para se evitar interferência co-local, e δ_{ij} é definida por restrições tecnológicas dos transmissores.

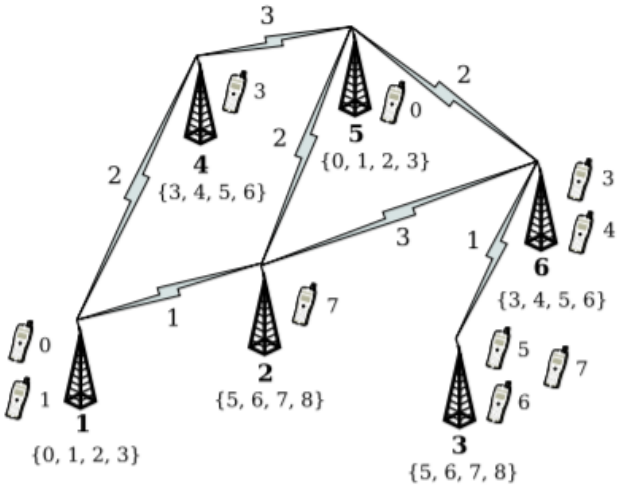
Dadas as restrições, pode-se investigar:

1. A viabilidade de se respeitar todas as restrições
2. Minimizar a máxima frequência utilizada, caso seja viável respeitar todas as restrições

3. Minimizar o número de frequências diferentes usadas, caso seja viável respeitar todas as restrições
4. Maximizar a viabilidade, atribuindo custos às violações de restrições, os quais devem ser minimizados

A figura abaixo ilustra uma rede com 6 antenas [30], cada uma representada com seu identificador e seu conjunto de frequências (embaixo). Os aparelhos celulares representam os links com os canais alocados e os raios entre as antes indica a distância mínima de separação para as frequências usadas por duas antenas.

Fig. 7. Ilustração de uma rede com seus conjuntos de frequência e restrições



Fonte: [30].

Esse é o contexto que serve de base para os experimentos apresentados nos capítulos seguintes, deixando claro desde já que o trabalho se ateve especialmente no item 1 acima, a satisfação de todas as restrições.

3. Materiais e Métodos

3.1. Conjunto de Dados

O *CELAR* (Centro de Eletrônica do Exército Francês) construiu um conjunto de dados para o problema RLFAP a partir de dados de redes reais, como parte do projeto *EUCLID CALMA (Combinatorial Algorithms for Military Applications)* [29], e pode ser encontrado em [31].

O conjunto é composto por três instâncias nas quais são fornecidas as variáveis, o conjunto do domínio das frequências, as restrições na forma $|f_i - f_j| > d_{ij}$ ou $|f_i - f_j| = \delta_{ij}$, bem como os critérios a serem otimizados e os custos para cada violação:

- CELAR: 11 instâncias
- GRAPH: ou *Generating Radio link Assignment Problems Heuristically* [32], são 14 instâncias geradas a partir das características presentes na CELAR como os domínios das frequências e proporção de links e restrições
- SUBCELAR : 5 sub-instâncias extraídas a partir da instância CELAR 6, possui tamanho reduzido mas igual dificuldade, sendo ideal para *benchmarking*

Cada instância possui quatro arquivos:

- *var.txt*: associa as variáveis (*links*) a seus domínios;
- *dom.txt*: fornece a definição dos domínios;
- *ctr.txt*: fornece as restrições e os índices de mobilidade, que vão de 0 a 4, onde 0 significa que a restrição deve ser respeitada, enquanto as mobilidades 1-4 apresentam um aumento dos custos;
- *cst.txt*: fornece os custos associados a cada índice de mobilidade.

3.2. Codificação do problema

A codificação utilizada foi baseada em [33], e pode ser quebrada em duas partes:

- domínios, isso é, codificar que $f_i \in D_i$
- restrições $|f_i - f_j| > d_{ij}$ ou $|f_i - f_j| = \delta_{ij}$

Os domínios podem ser vistos como uma união disjunta de quatro subdomínios:

$$\{2 + 14m \mid 1 \leq m \leq 11\} \qquad \{2 + 14m \mid 18 \leq m \leq 28\}$$
$$\{8 + 14m \mid 29 \leq m \leq 39\} \qquad \{8 + 14m \mid 46 \leq m \leq 56\}$$

Assim, foram utilizadas duas variáveis, uma proposicional t_i , para representar se f_i é $2 \; mod \; 14$ ou não, e um número inteiro m_i para representar $f_i \; mod \; 14$, obtendo :

$$t_i \rightarrow (1 \leq m_i \leq 11 \vee 18 \leq m_i \leq 28)$$
$$\neg t_i \rightarrow (29 \leq m_i \leq 39 \vee 46 \leq m_i \leq 56)$$

A codificação das restrições é feita analisando, para cada caso, se t_i e t_j são $2 \; mod \; 14$ ou não. Supondo que t_i seja e que t_j não, temos:

$$| 2 + 14 \, m_i - 8 - 14 \, m_j | > k$$

que após manipular se torna:

$$(t_i \wedge \neg t_j) \rightarrow (m_i - m_j \geq \lfloor (k+6)/14 \rfloor + 1 \vee m_i - m_j \leq \lceil (-k+6)/14) \rceil - 1)$$

Fazendo o mesmo para o caso $|f_i - f_j| = \delta_{ij}$, obtemos todos os outros casos:

$$(\neg t_i \wedge t_j) \rightarrow (m_i - m_j \geq \lfloor (k-6)/14 \rfloor + 1 \vee m_i - m_j \leq \lceil (-k-6)/14) \rceil - 1)$$
$$(t_i \wedge t_j) \rightarrow (m_i - m_j \geq \lfloor k/14 \rfloor + 1 \vee m_i - m_j \leq \lceil -k/14 \rceil - 1)$$
$$(\neg t_i \wedge \neg t_j) \rightarrow (m_i - m_j \geq \lfloor k/14 \rfloor + 1 \vee m_i - m_j \leq \lceil -k/14 \rceil - 1)$$

$$(t_i \wedge \neg t_j) \rightarrow (m_i - m_j = \lfloor (k+6)/14 \rfloor + 1 \vee m_i - m_j = \lceil (-k+6)/14) \rceil - 1)$$

$$(\neg t_i \wedge t_j) \rightarrow (m_i - m_j = \lfloor (k-6)/14 \rfloor + 1 \vee m_i - m_j = \lceil (-k-6)/14 \rceil - 1)$$

$$(t_i \wedge t_j) \rightarrow (m_i - m_j = \lfloor k/14 \rfloor + 1 \vee m_i - m_j = \lceil -k/14 \rceil - 1)$$

$$(t_i \wedge t_j) \rightarrow (m_i - m_j = \lfloor k/14 \rfloor + 1 \vee m_i - m_j = \lceil -k/14 \rceil - 1)$$

3.3. Z3 SMT Solver

O Z3 [34], da *Microsoft Research*, é um *solver* condecorado [35, 36], que se tornou *open-source* em 2015, e suporta as principais teorias, como Aritmética de Inteiros, Reais, Funções Não-Interpretadas, *Arrays*, Vetores de *bits*, tipos de dados algébricos, entre outros, podendo ser obtido em [37].

Internamente na Microsoft, o Z3 é utilizado na verificação de software como *drivers* para o sistema operacional *Windows* e na geração de casos de testes[38]. No campo da biologia, o Z3 é utilizado na análise e síntese de redes regulatórias genéticas [39]. De maneira semelhante, ele também é usado na análise e síntese de compostos químicos [40].

Apesar de escrito em *C++*, o *solver* pode ser utilizado a partir de diversas linguagens como *Python, C, Java, C#* e *OCaml*.

Neste trabalho, utilizou-se a interface em *Python* em execução em um sistema Intel Core i7-7700HQ @ 2.80 GHz e 16 GB de memória *RAM*, com Sistema Operacional *Ubuntu* 18.04.

O Z3 oferece duas classes básicas de *solvers*. Um *Solver* pode ser instanciado com uma lógica (*QF_LIA - Quantifier-Free Linear Arithmetic*) ou não (ficando o Z3 responsável por identificar a melhor opção). A outra classe é a *Optimize*, na qual pode-se incluir pesos para as cláusulas adicionadas, as quais serão automaticamente minimizadas.

Para adicionar cláusulas, basta usar os métodos *assert* e *assert_soft*, o qual recebe um peso (por padrão é 1) e um *id*, o qual pode ser utilizado para agrupar ou separar objetivos. Uma vez adicionadas as cláusulas, utilizam-se os métodos *check* e *model* para, respectivamente, verificar se as fórmula final é satisfatível (ou não), e obter o modelo encontrado

As variáveis são instanciadas a partir de funções como *Bool*, *Int*, *Real, Array*, etc, as quais recebem um nome como argumento.

Os exemplos que seguem foram retirados do tutorial do Z3 [41] e de [42], sendo o último uma extensa e diversa coleção de problemas discutidos em detalhe.

Suponha o seguinte conjunto de equações lineares:

1. $3x + 2y - z = 1$
2. $2x - 2y + 4z = -2$
3. $-x + \frac{1}{2}y - z = 0$

Pode-se facilmente codificar o problema usando Z3:

```
from z3 import *
x = Real('x')
y = Real('y')
z = Real('z')
s = Solver ()
```

```
s.add (3*x + 2*y - z == 1)
s.add (2*x - 2*y + 4*z == -2)
s.add(-x + 0.5*y - z == 0)
print(s.check())
print(s.model())
```

O qual, após executado, retorna:

```
sat
[z = -2, y = -2, x = 1]
```

Para o próximo exemplo, é bom recordar que, para verificar se uma fórmula F é válida, basta provar sua negação $\neg F$ é inválida. Assim, podemos usar o Z3 para provar o teorema de De Morgan.

```
from z3 import *

s = Solver()
a,b = Bools('a b')
de_morgan = And(a,b) == Not(Or(Not(a),Not(b)))

s.add(Not(de_morgan))
print(s.check())
```

Que retorna:

```
unsat
```

Suponha agora que queremos otimizar a soma de dois números inteiros, x e y, com as restrições de que $x < 2$ e $y - x < 1$:

```
from z3 import *

s = Optimize()
x,y = Ints('x y')

s.add(x < 2)
s.add(y - x < 1)

s.maximize(x+y)

print(s.check())

print(s.model())
print(s.model().evaluate(x + y))
```

Na última linha, utilizamos o próprio modelo gerado pelo *solver* para avaliar a função custo, que nos retorna:

```
sat
[y = 1, x = 1]
2
```

Retornando ao RLFAP, suponha que estejamos iterando sobre uma lista de restrições (presentes nos arquivos *ctr.txt*), na qual cada restrição possui os links f_i (*first_var*) e f_j (*second_var)*, o operador (*operator*) utilizado (> ou =), bem como a distância (*deviation*) e os custos (*costs*). Assumindo que as restrições com operador (=) não possuem mobilidade, pode-se codificar o problema da seguinte maneira:

```
from z3 import *
from math import ceil, floor

s = Optimize()

for ctr in Ctrs:
    # declarando as variáveis
    ti = Bool('t_%d' % ctr.first_var)
    tj = Bool('t_%d' % ctr.second_var)
    mi = Int('m_%d'  % ctr.first_var)
    mj = Int('m_%d'  % ctr.second_var)
    k  = ctr.deviation



    # codificando o domínio de Fi
    # ti  → (1  ≤ mi ≤ 11 ∨ 18 ≤ mi ≤ 28)
    # ¬ti → (29 ≤ mi ≤ 39 ∨ 46 ≤ mi ≤ 56)
    s.add(Implies(
         ti,
         Or( And(1  <= mi, mi <= 11), And(18 <= mi, mi <= 28)))
    )


    s.add(Implies(
         Not(ti), Or( And(29 <= mi, mi <= 39), And(46 <= mi, mi <= 56)))
    )

    # codificando o domínio de Fj
    s.add(Implies(
          tj,
```

```
          Or(And(1  <= mj, mj <= 11), And(18 <=m j,mj <= 28)))
    )

    s.add(Implies(
          Not(tj),
          Or(And(29 <= mj, mj <= 39), And(46 <= mj, mj <= 56)))
    )
```

Note que utilizamos o método *add* para adicionar cláusulas que devem ser respeitadas (*hard constraints*). Já para adicionar as restrições e seus custos:

```
if ctr.operator == '>':

    weight = costs[ctr.weight]

    # (ti ∧ ¬tj)
    s.add_soft(
       Implies(And(ti, Not(tj)), Or(mi - mj >= math.floor((k+6)/14) +
1,
                                    mi - mj <= math.ceil((-k+6)/14) -1)),
              weight
    )
   # (¬ti ∧ tj)
   s.add_soft(
      Implies(And(Not(ti), tj), Or(mi - mj >= math.floor((k-6)/14) + 1,
                                   mi - mj <= math.ceil((-k-6)/14) -
1)),
           weight
   )

  # (ti ∧ tj)
  s.add_soft(
       Implies(And(ti,tj) ,    Or( mi - mj >= math.floor(k/14) + 1,
                                   mi - mj <= math.ceil(-k/14) -1)),
       weight
  )


   # (¬ti ∧ ¬tj)
   s.add_soft(
    Implies(And(Not(ti), Not(tj)), Or(mi - mj >= math.floor(k/14) + 1,
                                    mi - mj <= math.ceil(-k/14) -1)),
    weight)
```

4. Resultados

O Z3 foi utilizado com todas as instâncias do grupo GRAPH, e seus resultados são expostos na tabela a seguir, os dados sobres as instâncias são encontrados em [29].

Tabela 4 - Resultados obtidos pela verificação das instâncias *GRAPH*

Instância	No. de variáveis	No. de restrições	Realizável?	Resposta Z3	Tempo (s)
GRAPH-1	200	1134	Sim	SAT	0,157
GRAPH-2	400	2245	Sim	SAT	0,924
GRAPH-3	200	1134	Sim	SAT	1,274
GRAPH-4	400	2244	Sim	SAT	87,94
GRAPH-5	200	1134	Não	UNSAT	0,056
GRAPH-6	400	2170	Não	UNSAT	0,127
GRAPH-7	400	2170	Não	UNSAT	0,118
GRAPH-8	680	3757	Sim	SAT	10,601
GRAPH-9	916	5246	Sim	SAT	99,081
GRAPH-10	680	3907	Sim	SAT	65,958
GRAPH-11	680	3757	Não	UNSAT	0,304
GRAPH-12	680	4017	Não	UNSAT	0,404
GRAPH-13	916	5273	Não	UNSAT	0,631
GRAPH-14	916	4638	Sim	SAT	6,848

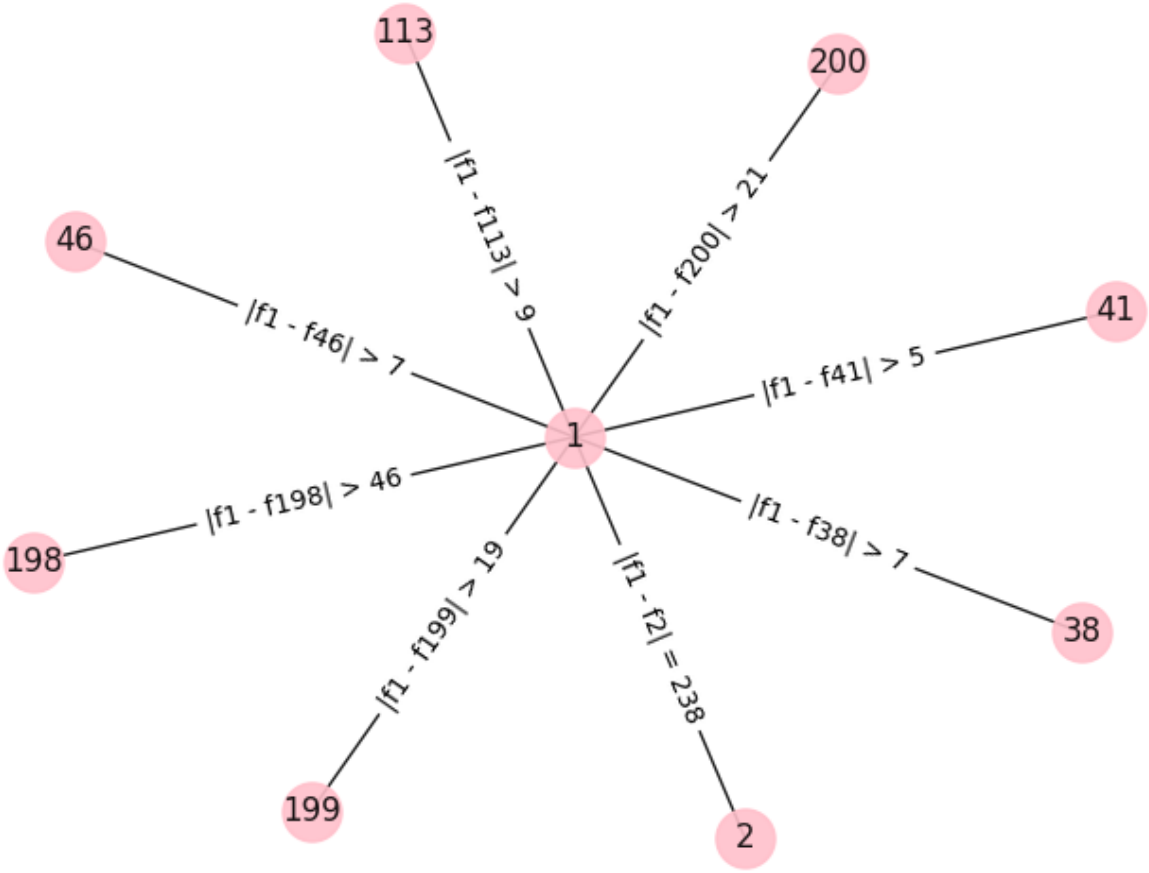
Fonte: Elaborada pelo autor, adaptado de [29]

Pode-se notar que em muitos casos, o Z3 consegue definir a satisfatibilidade de maneira extremamente rápida, no resto ainda apresenta um bom desempenho, dada a dificuldade intrínseca do problema em questão.

A figura a seguir apresenta uma parte das restrições presentes na instância *GRAPH-1*, levando-se em consideração apenas a variável de número 1, e ilustra suas conexões com outras

variáveis. É importante esclarecer que trata-se apenas de uma ilustração, não apresentando nenhuma informação relativa a disposição topológica da rede em questão.

Fig. 8 - Ilustração das restrições relativas à variável 1 da instância *GRAPH-1*



Fonte: Elaborada pelo autor, adaptada de [29].

A execução pelo Z3 resultou em $t_1 = \perp$ e $m_1 = 49$, o que significa que a frequência alocada foi $f_1 = 694\ Hz$. A tabela a seguir indica os valores encontrados para as variáveis ilustradas na figura anterior, mostrando que de fato as restrições foram respeitadas.

Tabela 5 - Restrições e valores encontrados pelo Z3

Variável	t_j	m_j	Tipo da restrição	d_{ij}	f_j	$\lvert f_1 - f_j \rvert$
2	⊥	32	=	238	456	238
38	⊤	22	>	7	310	384
41	⊥	35	>	5	498	196
46	⊥	29	>	7	414	280
113	⊥	36	>	9	512	182
198	⊤	19	>	46	268	426
199	⊥	38	>	19	540	54
200	⊥	56	>	21	792	98

Fonte: Elaborada pelo autor.

Como o aumento do número de variáveis aumenta exponencialmente o espaço de soluções a ser percorrido, é de se esperar que o tempo aumente muito. Contudo, na prática, conjuntos de dados reais envolvidos em casos industriais costumam apresentar uma estrutura que é explorada pelo maquinário dos *solvers*.

As figuras a seguir, provenientes da competição *SAT 2013* [43], ilustram essa diferença estrutural presente em conjuntos de dados de casos industriais, em contraste com conjuntos de dados gerados aleatoriamente.

Fig. 9 - Estrutura de um conjunto de dados industrial [43].

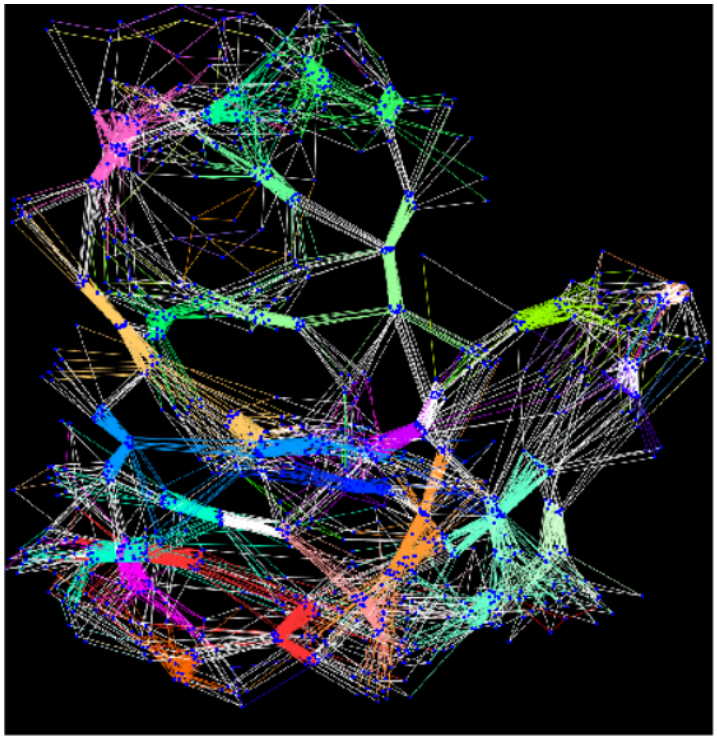
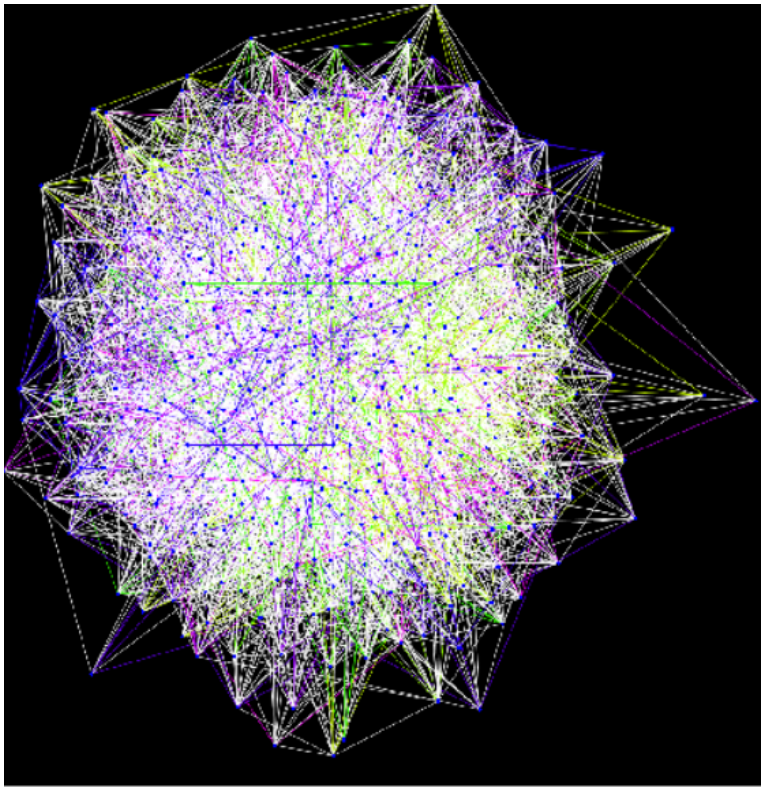


Fig. 10 - Estrutura de um conjunto de dados gerado aleatoriamente [43].



5. Conclusões

O presente trabalho teve como objetivo explorar os conceitos de SAT e SMT, fornecendo o fundamento teórico sobre lógica necessário para entender como modelar problemas e passando pela evolução dos principais algoritmos utilizados. Uma aplicação básica foi utilizada no contexto de engenharia discutida e realizada, na qual o Z3 SMT Solver mostrou um bom desempenho na solução da maioria das instâncias.

Apesar da vasta literatura existente sobre o tema, a principal dificuldade encontrada está na codificação de problemas em termos de Lógica Proposicional e de Primeira Ordem, a qual geralmente requer conhecimento íntimo do sistema abordado e necessitam de perícia por parte do usuário para gerar fórmulas que sejam eficientes. O mesmo pode ser dito sobre as implementações de SAT e SMT *solvers*, as quais requerem um profundo conhecimento nos métodos e abordagens mais eficientes para possibilitar uma interação satisfatória a nível de uso industrial. Assim, a utilização de tais técnicas é mais comumente encontrada em grandes empresas, com porte suficiente para manter grupos especializados na área, como *Amazon*, *Intel* e *Microsoft*, ou em empresas de pequeno porte com membros da academia como *Barcelogic* [7].

Deve-se enfatizar, contudo, que nem todo problema é viável de ser abordado pelos meios expostos neste trabalho. Não apenas por conta da complexidade computacional (NP-Completo) mas também porque necessita que haja um procedimento de decisão (completo ou não) eficiente para os domínios de problemas de interesse.

Contudo, trata-se de uma área madura e promissora, a qual tem ganhado cada vez mais espaço na indústria e continua com um alto fluxo de publicações. A existência de competições certamente colaboram na divulgação e padronização de formatos e métricas para comparar diferentes *solvers* a partir de problemas reais.

Como trabalhos futuros, uma das linhas que podem ser exploradas é lidar com os objetivos de minimizar a frequência máxima, minimizar o número de frequências diferentes e minimizar os custos associados com as restrições que não foram respeitadas, como se propõe em [29]. Trata-se da variante chamada MAX-SAT e MAX-SMT [44], na qual deve-se satisfazer o maior número de cláusulas possíveis.

6. Referências

[1] ROIG, I. A. **Solving Hard Industrial Combinatorial Problems with SAT**. 2013. 160 p. Ph.D. Thesis - Technical University of Catalonia, Barcelona, España, 2013. Disponível em: <https://pdfs.semanticscholar.org/a13b/6a182357eca93ffbeff8abccb2208a76859e.pdf>. Acesso em: 3 nov. 2019.

[2] KAIVOLA, R. et al. Replacing Testing with Formal Verification in Intel ® CoreTM i7 Processor Execution Engine Validation. In: Bouajjani A., Maler O. (eds) **Computer Aided Verification. CAV 2009**. Lecture Notes in Computer Science, vol 5643. Springer, Berlin, Heidelberg, 2009. Disponível em: <https://is.muni.cz/el/1433/jaro2010/IA159/um/intel.pdf>. Acesso: em 3 mai. 2019.

[3] NICELY, T. *Pentium FDIV flaw FAQ*. <http://www.trnicely.net/pentbug/pentbug.html>. Acesso em: 3 nov. 2019.

[4] BJORNER, N. SMT Solvers for Testing, Program Analysis and Verification at Microsoft. **11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing**, Timisoara, 2009, p.15. Disponível em: <http://barbie.uta.edu/~xlren/Z3/SMTsolversforTesting,Program%20AnalysisandVerificationatMicrosoft.pdf>. Acesso em: 3 nov. 2019.

[5] BACKES, J. et al. Semantic-based Automated Reasoning for AWS Access Policies using SMT. **2018 Formal Methods in Computer Aided Design (FMCAD)**, Austin, Texas, 2018, p.1-9. Disponível em: <https://ieeexplore.ieee.org/document/8602994>. Acesso em: 3 nov. 2019.

[6] EIT DIGITAL. *Automated Reasoning: satisfiability*. Acesso em: 3 nov. 2019. <https://www.coursera.org/learn/automated-reasoning-sat/>.

[7] BARCELOGIC. *Barcelogic - Efficiency, simplified*. <https://barcelogic.com/en/>. Acesso em: 3 nov. 2019.

[8] COOK, S. A. The complexity of theorem-proving procedures. **Proceedings of the third annual ACM symposium on Theory of computing**, 1971. p.151-158. Disponível em: <https://www.cs.toronto.edu/~sacook/homepage/1971.pdf>. Acesso em: 3 nov. 2019.

[9] SAT COMPETITIONS. *The International SAT Competitions Web Page*. <http://www.satcompetition.org>. Acesso em: 3 nov. 2019.

[10] SMT-COMP. *SMT-COMP 2019*. <https://smt-comp.github.io/>. Acesso em: 3 nov. 2019.

[11] MALIK, S.; ZHANG, L. Boolean Satisfiability From Theoretical Hardness to Practical Success. **Communications of the ACM**, v. 52, n. 8, p. 76-82, August., 2009. Disponível em: <http://www.cs.toronto.edu/~chechik/courses18/csc410/p76-malik.pdf>. Acesso em: 3 mar. 2019.

[12] SHANNON, C. E. **A symbolic analysis of relay and switching circuits**. 1937. 72 p. M.S. Thesis - Dept. of Elect. Eng, MIT, Cambridge, MA, USA, 1937. Disponível em: <https://dspace.mit.edu/handle/1721.1/11173>. Acesso em: 3 nov. 2019.

[13] BOOLE, G. **The Mathematical Analysis of Logic, Being an Essay towards a Calculus of Deductive Reasoning,** Londres, Barclay, & Macmillan, 1847. Disponível em: <http://www.gutenberg.org/ebooks/36884>. Acesso em: 3 nov. 2019.

[14] BRADLEY, A. R.; MANNA, Z. **The Calculus of Computation: Decision Procedures with Applications to Verification**, 1st ed. Heidelberg: Springer, 2007.

[15] GOMES, C. P.; KAUTZ, H.; SABHARWAL, A.; SELMAN, B. Satisfiability Solvers. In: **Handbook of Knowledge Representation**, 1st ed., F. van Harmelen, V. Lifschitz and B. Porter, Eds. Elsevier Science, 2008, p. 89-122.

[16] PRESTWITCH, S.CNF Encodings. In: BIERE, A.; HEULE, M.; VAN MAREEN, H.; WALSH, T. **Handbook of satisfiability**. 1st. ed. Amsterdam. IOS Press, 2009, p. 75-93.

[17] KROENING, D.; STRICHMAN, O. **Decision Procedures: An Algorithmic Point of View.** 1st ed. Springer-Verlag, 2008.

[18] FRANCO, J; MARTIN, J. A History of Satisfiability. In: BIERE, A.; HEULE, M.; VAN MAREEN, H.; WALSH, T. **Handbook of satisfiability**. 1st. ed. Amsterdam. IOS Press, 2009, p. 3-55.

[19] TORLAK, E. *A Primer on Boolean Satisfiability*. Acesso em: 5 out. 2019. <https://homes.cs.washington.edu/~emina/blog/2017-06-23-a-primer-on-sat.html>

[20] MARQUES-SILVA, J. P.; SAKALLAH, K. A. GRASP: A new search algorithm for satisfiability. **International Conference on Computer Aided Design**, p. 220-227, Nov. 1996. Disponível em: <https://www.cs.cmu.edu/~emc/15-820A/reading/grasp_iccad96.pdf>. Acesso em: 6 mai. 2019.

[21] GRITMAN, A.; HA, A.; QUACH, T.; WENGER, D. *Conflict Driven Clause Learning*. <https://cse442-17f.github.io/Conflict-Driven-Clause-Learning/>. Acesso em: 17 set. 2019.

[22] MARQUES-SILVA, J.; LYNCE, I.; MALIK, S. Conflict-Driven Clause Learning SAT Solvers. n: BIERE, A.; HEULE, M.; VAN MAREEN, H.; WALSH, T. **Handbook of satisfiability**. 1st. ed. Amsterdam. IOS Press, 2009, p. 131-150.

[23] BERRE, D. L. **Introduction to SAT - History, Algorithms, Practical considerations**. 203 slides. Disponível em: <http://satsmt2014.forsyte.at/files/2014/07/SAT-introduction.pdf>. Acesso em: 29 out. 2019.

[24] NIEUWENHUIS, R.; OLIVERAS, A.; TINELLI, C. Abstract DPLL and Abstract DPLL Modulo Theories. In: BAADER F.; VORONKOV, A. (Eds) **Logic for Programming, Artificial Intelligence, and Reasoning. LPAR 2005**. Lecture Notes in Computer Science, vol 3452. Springer,

Berlin, Heidelberg. Disponível em <http://web.stanford.edu/class/cs357/NOT04.pdf>. Acesso em: 4 nov. 2009.

[25] REYNOLDS, A. **SMT Solvers for Verification and Synthesis**. 275 slides. Disponível em: <https://homepage.cs.uiowa.edu/~ajreynol/VTSA2017/>. Acesso em: 10 jul. 2019.

[26] MOSKEWICZ, N. M. et al. Chaff: engineering an efficient SAT solver. **Proceedings of the 38th Design Automation Conference**, 2001. p.530-535. Disponível em: <https://www.princeton.edu/~chaff/publication/DAC2001v56.pdf>. Acesso em: 2 set. 2019

[27] KOSTER, A. M. C. **Frequency assignment : models and algorithms**. Universiteit Maastricht, 1999. Disponível em: < https://cris.maastrichtuniversity.nl/portal/files/1339014/guid-1dc820d2-1556-4f4e-8024-80a9dea10058-ASSET1.0.pdf>. Acesso em: 2 set. 2019.

[28] AUDHYA, G.K. et al. A survey on the channel assignment problem in wireless networks. **Wireless Communication and Mobile Computing**, vol 11, n.5, p.583-609, Mai. 2011. Disponível em: <https://onlinelibrary.wiley.com/doi/epdf/10.1002/wcm.898>. Acesso em: 12 set. 2019.

[29] CABON, B.; GIVRY, S.; LOBJOIS, L.; SCHIEX, T.; WARNERS, J. P. Radio link frequency assignment, **Constraints**, v. 4, n. 1, p. 79-89, 1999. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.14.8545&rep=rep1&type=pdf>. Acesso em: 5 jun. 2019.

[30] DIAS, B. R. C.; RODRIGUES, R. de F.; MACULAN FILHO, N. Alocação de canais em redes celulares sem fio: algoritmos e modelos teóricos em grafos e escalonamento. **Anais do XLIV Simpósio Brasileiro de Pesquisa Operacional**, 2012. Disponível em: <http://www.din.uem.br/sbpo/sbpo2012/pdf/arq0007.pdf>. Acesso em: 5 jun. 2019.

[31] SCHIEX, T. *The CELAR Radio Link Frequency Assignment Problems*. <http://www7.inra.fr/mia/T/schiex/Doc/rlfap.shtml>. Acesso em: 12 jun. 2019.

[32] BENTHEM, H. P. V. **GRAPH - Generating Radio link frequency Assignment Problems Heuristically**, Delft University of Technology, 1995. Disponível em: <https://www.researchgate.net/publication/2757325_GRAPH_-_Generating_Radio_link_frequency_Assignment_Problems_Heuristically>. Acesso em: 11 jun. 2019.

[33] NIEUWENHUIS, R.; OLIVERAS, A. On SAT Modulo Theories and Optimization. In: BIERE, A.; GOMES, C.P. (Eds) **Theory and Applications of Satisfiability Testing - SAT 2006**. SAT 2006. Lecture Notes in Computer Science, vol 4121. Springer, Berlin, Heidelberg. Disponível em: <https://www.cs.upc.edu/~oliveras/espai/papers/sat06.pdf>

[34] DE MOURA, L.; BJORNER, N. Z3: An Efficient SMT Solver. **Tools and Algorithms for the Construction and Analysis of Systems (TACAS).** Lecture Notes in Computer Science, vol 4963. Springer, Berlin, Heidelberg, 2008. Disponível em: <https://www.researchgate.net/publication/225142568_Z3_an_efficient_SMT_solver>. Acesso em: 19 out. 2019.

[35] MICROSOFT. *Z3 wins 2015 ACM SIGPLAN Award*. <https://www.microsoft.com/en-us/research/blog/z3-wins-2015-acm-sigplan-award/>. Acesso em: 3 nov. 2019.

[36] DE MOURA, L.; BJORNER, N. *The inner magic behind the Z3 theorem prover.* <https://www.microsoft.com/en-us/research/blog/the-inner-magic-behind-the-z3-theorem-prover/> . Acesso em: 3 nov. 2019.

[37] DE MOURA, L.; BJORNER, N. *The Z3 Theorem Prover*. <https://github.com/Z3Prover/z3>. Acesso em: 3 nov. 2019.

[38] DE MOURA, L.; BJORNER, N. **Applications of SMT solvers to Program Verification**. <https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/nbjorner-smt-application-chapter.pdf>. Acesso em: 12 out. 2019.

[39] PAOLETTI, N. et al. Analyzing and Synthesizing Genomic Logic Functions. In: BIERE,A.; BLOEM, R. (Eds). ***Computer Aided Verification*** (Lecture Notes in Computer Science),.Cham: Springer, 2014, p. 343-357. Disponível em: <https://www.microsoft.com/en-us/research/wp-content/uploads/2014/07/pyhwk14.pdf>. Acesso em: 1 nov. 2019.

[40] FAGERBER, R.; FLAMM, C.; MERKLE, D.; PETERS, P. Exploring Chemistry using SMT. In: MILANO, M. (Ed). **Principles and Practice of Constraint Programming** (Lecture Notes in Computer Science), M. Milano, Eds. Hidelberg: Springer, 2012, p. 900-915. Disponível em: <https://www.tbi.univie.ac.at/newpapers/pdfs/TBI-p-2012-8.pdf>. Acesso em: 3 nov. 2019.

[41] MICROSOFT. *Getting Started with Z3: A Guide*. <https://rise4fun.com/Z3/tutorial/guide>. Acesso em: 11 set. 2019.

[42] YURICHEV, D. *SAT/SMT by Example*. <https://yurichev.com/SAT_SMT.html>. Acesso em: 11 jul. 2019.

[43] NEWSHAM, Z.; LINDSAY, W.; GANESH, V.; LIANG, J.H.; FISCHMEISTER, S.; CZARNECKI, K. SATGraf: Visualizing the Evolution of SAT Formula Structure in Solvers. In: HEULE, M.; WEAVER, S. (Eds). **Theory and Applications of Satisfiability Testing -- SAT 2015**. SAT 2015. Lecture Notes in Computer Science, vol 9340. Springer, Cham.

[44] MARQUES-SILVA, J. **MaxSAT and Related Optimization Problems.** 145 slides. Disponível em: <https://es-static.fbk.eu/events/satsmtschool12/slides/1x04_SS12.pdf>. Acesso em: 3 nov. 2019.