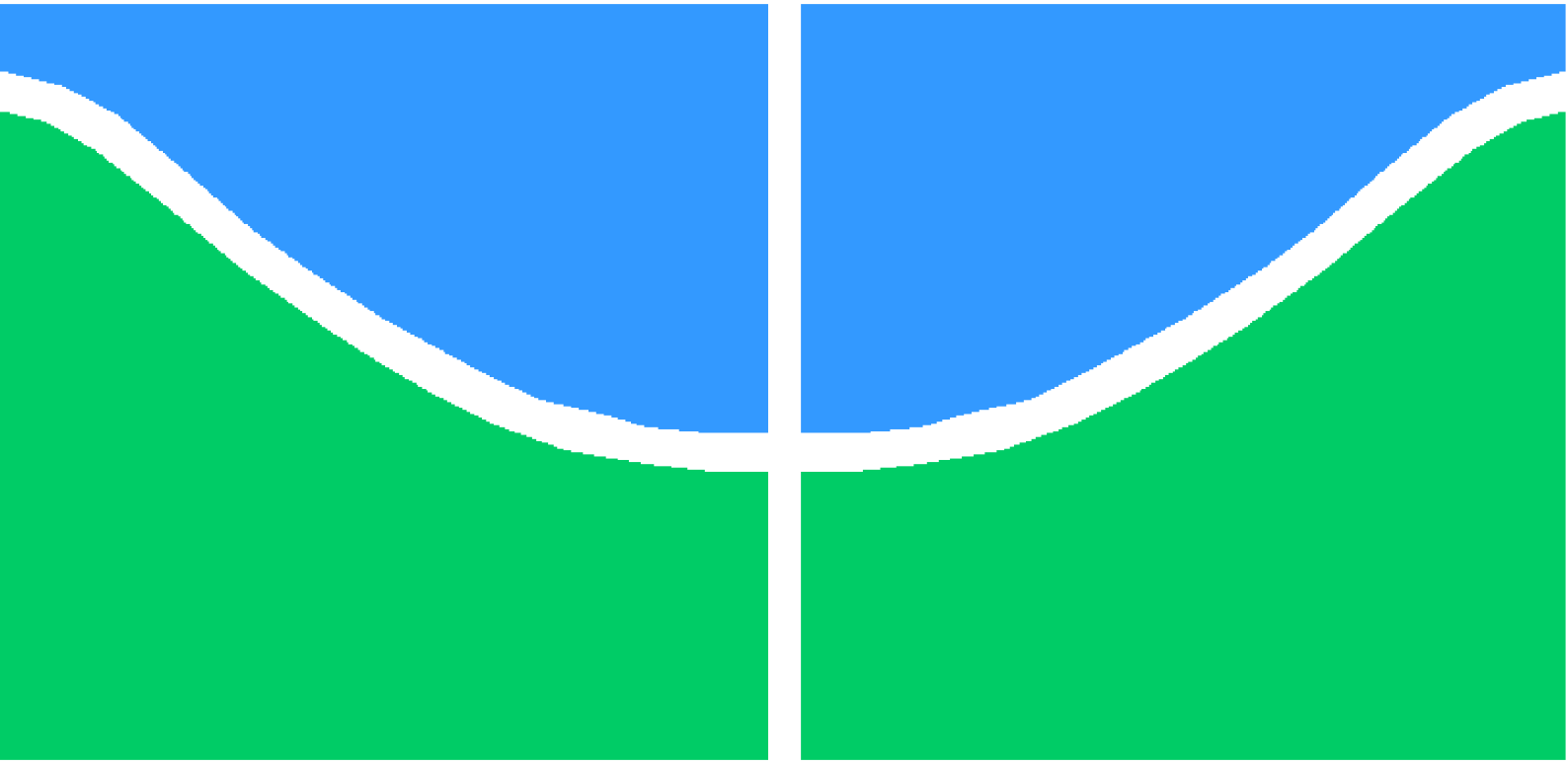

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de Software

Planejamento Clássico: Análise para otimização do tempo de geração de fórmulas do SATPLAN

Autor: Marcelo Martins de Oliveira

Orientador: Prof. Dr. Bruno César Ribas

Brasília, DF

2023

Marcelo Martins de Oliveira

Planejamento Clássico: Análise para otimização do tempo de geração de fórmulas do SATPLAN

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Dr. Bruno César Ribas

Brasília, DF

2023

Marcelo Martins de Oliveira
Planejamento Clássico: Análise para otimização do tempo de geração de fórmulas do SATPLAN/ Marcelo Martins de Oliveira. – Brasília, DF, 2023-
50 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Bruno César Ribas

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2023.

1. Planejamento. 2. Satisfatibilidade. I. Prof. Dr. Bruno César Ribas. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Planejamento Clássico: Análise para otimização do tempo de geração de fórmulas do SATPLAN

CDU 02:141:005.6

Marcelo Martins de Oliveira

Planejamento Clássico: Análise para otimização do tempo de geração de fórmulas do SATPLAN

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 24 de Julho de 2023:

Prof. Dr. Bruno César Ribas
Orientador

Prof. Dr. John Lenon Cardoso Gardenghi
Convidado 1

Prof. Dr. Tiago Alves da Fonseca
Convidado 2

Brasília, DF
2023

Agradecimentos

Agradeço primeiramente a Deus por conseguir concluir esse processo tão difícil. Agradeço a toda minha família, que sempre me apoiou nessa longa jornada, em especial aos meus pais, Marcelo e Maria Aparecida, que me apoiaram e deram toda a condição para que eu pudesse estudar longe de casa.

Aos meus irmãos, Camila e Igor, que sempre pensavam em ideias mirabolantes de aplicar os meus conhecimentos em alguma coisa da área deles. E que apesar de suas maneiras diferentes de apoiar sempre estavam presentes quando eu precisava.

A minha esposa, Jéssica, que nunca me deixou desistir e sempre me dava forças para terminar tudo que me era imposto.

Aos meus amigos, Victor, Ygor, João Lucas, Gabriel e Andrew que sempre passaram madrugadas em claro resolvendo ou me ajudando de alguma forma.

A todos os professores que tive o prazer de conhecer na UNB. E em especial ao meu orientador, Bruno César, que teve toda a paciência para que esse trabalho fosse concluído.

Obrigado a todos por impactarem de alguma maneira na minha vida.

Resumo

Planejamento é o ato de se preparar para algo quando se possui um objetivo a ser alcançado. O planejamento de negócios é usado por qualquer empresa que precise manter uma abordagem estruturada para o crescimento ao longo do tempo. O planejamento automático é um dos principais campos da inteligência artificial. Ele desempenha um papel crucial na concepção de abordagens que permitem que programas automatizados resolvam problemas específicos de forma automática.. Essa área da inteligência artificial se concentra em criar métodos para que os agentes descubram problemas sem a necessidade de serem programados. Problemas que são auto planejados requerem modelos de estado determinísticos com soluções definidas. As soluções são formuladas pelos agentes e, ao serem executadas, o problema é resolvido. Neste trabalho, propomos um estudo sobre como o *software* SATPLAN funciona, quais são as entradas e saídas que são necessárias para a resolução de problemas utilizando-o. Com isso pretende-se encontrar pontos de melhorias na performace do *software*, melhorando assim o tempo de resolução dos seus problemas.

Palavras-chaves: Planejamento. Planejamento Clássico. Satisfatibilidade. SATPLAN. SAT solver.

Abstract

Planning is the act of preparing for something when you have a goal to achieve. Business planning is used by any company that needs to maintain a structured approach to growth over time. Automatic planning is one of the main fields of artificial intelligence. It plays a crucial role in designing approaches that enable automated programs to solve specific problems automatically.. This area of artificial intelligence focuses on creating methods for agents to discover problems without having to be programmed. Problems that are self-planning require deterministic state models with defined solutions. The solutions are formulated by the agents and, when executed, the problem is solved. In this work, we propose a study on how the SATPLAN software works, what are the inputs and outputs that are necessary for solving problems using it. The aim is to find points of improvement in the performance of the software, thus improving the resolution time of your problems.

Key-words: Planning. Classic Planning. Satisfaction. SATPLAN. sat solver.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

BB	BLACKBOX
CNF	Forma Normal Conjuntiva (*Conjuctive Normal Form*)
DPLL	*Davis-Putnam-Logemann-Loveland*
GCC	*GNU Compiler Collection*
IA	Inteligência Artificial
Mutex	Proposições Mutuamente exclusivas
PDDL	*Planning Domain Definition Language*
SAT	Satisfatibilidade
STRIPS	*STanford Research Institute Problem Solver*

Sumário

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO E JUSTIFICATIVA

Planejamento é o ato de se preparar para algo quando se possui um objetivo a ser alcançado. Uma sequência de ações é criada quando temos o ato de planejar, o que permite alcançar objetivos a partir de fatos iniciais. O objetivo e os fatos iniciais são dois estados distintos (SCHREINER, 2012).

Os fatos representam os estados, e as situações descrevem o conjunto de proposições, afirmações que podem ser verdadeiras ou falsas. Quando há aplicação de uma ação há uma mudança de estado. Para obter um plano é necessário um conjunto de ações que alcancem um estado objetivo a partir do estado inicial (SCHREINER, 2012).

1.2 OBJETIVOS

1.2.1 OBJETIVO GERAL

O objetivo geral deste trabalho foi identificar as causas que afetam a geração das fórmulas do programa SATPLAN e investigar estratégias para reduzir o tempo total de execução.

1.2.2 OBJETIVOS ESPECÍFICOS

- Executar problemas de competição;
- Utilizar ferramentas de profiling;
- Modificar o código de impressão das fórmulas do SATPLAN

1.2.3 ORGANIZAÇÃO DO TRABALHO

O trabalho está disposto da seguinte forma:

- Capítulo 1 contextualiza e mostra o objetivo deste trabalho;
- Capítulo 2 apresenta informações para entendermos como o SATPLAN funciona;
- Capítulo 3 apresenta as análises feitas e as modificações aplicadas no SATPLAN;
- Capítulo 4 traz as conclusões deste trabalho.

2 REVISÃO TEÓRICA

2.1 LÓGICA PROPOSICIONAL E SATISFATIBILIDADE

Lógica proposicional é uma representação formal de sentenças declarativas podendo ser verdadeiras ou falsas, podendo ser uma ou outra, não podendo ser ambas. Um exemplo disso é “O carro é preto” (RIBAS, 2015).

De acordo com Bedegral e Acióly (2007), os conectivos lógicos ou juntores são cinco:

- $\neg$ (negação)
- $\wedge$ (e)
- $\vee$ (ou)
- $\rightarrow$ (se)
- $\leftrightarrow$ (se somente se)

2.1.1 FORMA NORMAL CONJUNTIVA

A forma normal conjuntiva (CNF) é uma classe de fórmulas proposicionais, que dispõe de operadores lógicos de conjunção ($\wedge$), disjunção ($\vee$) e negação ($\neg$).

Conjunções de cláusulas $\wedge_i c_i$ sendo que cada cláusula c_i uma disjunção de literais $\vee_J l_j$ cada literal l_j uma variável booleana v ou uma negação v. CNF é uma forma muito simples, levando facilidade na implementação de algoritmos em um arquivo comum (BIERE; HEULE; MAAREN, 2009).

A Equação 2.1, retirada de Thiffault, Bacchus e Walsh (2004), é um exemplo de fórmula em CNF:

$$(\overline{a} \vee \overline{b} \vee c) \wedge (\overline{a} \vee \overline{b} \vee d \vee e). \tag{2.1}$$

Uma fórmula estará em CNF quando ela for composta por uma sequência de conjunções representadas por $\wedge$, possuindo uma ou mais cláusulas. Cada cláusula deve ser uma disjunção, $\vee$, de um ou mais literais. Um símbolo ou a negação dele são considerados um literal, que representa na lógica um valor que pode assumir Verdadeiro ou Falso, sendo que esse valor é uma instrução booleana. Todas as conjunções e disjunções de

literais, respectivamente, podem ser vistas como conjunções não-literais e conjunções de cláusula única (MONTANO; RIBAS, 2017).

Na Listagem 2.1 ,temos um exemplo básico de como seria a escrita de um arquivo em CNF.

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

Listagem 2.1 – Exemplo de um arquivo em CNF

Na Listagem 2.1 do arquivo a primeira linha com p caracteriza a linha do problema, e possui o seguinte formato p FORMATO VARIÁVEIS CLÁUSULAS, no exemplo temos que `p cnf 3 2`, cnf é o formato, `3 2` a quantidade de variáveis e quantidade de cláusulas, respectivamente. As variáveis podem ser enumeradas de `1 a n` (CHALLENGE, 1993).

As variáveis que tem como predecessor o símbolo de subtração (-), são variáveis que estão negadas. Os espaços em branco entre as variáveis são disjunções ($\vee$). Para se mostrar que uma cláusula chegou ao fim utiliza-se 0. Ao final de uma linha temos uma conjunção ($\wedge$) unindo as linhas de cláusulas. Na Equação 2.2 temos o exemplo de fórmula CNF para a Listagem 2.1 (CHALLENGE, 1993).

$$(a \vee \neg(c)) \wedge (b \vee c \vee \neg(a)). \tag{2.2}$$

2.1.2 SATISFATIBILIDADE

O Problema de Satisfação Booleano (SAT) é o problema que identifica se uma dada fórmula proposicional possa ser satisfeita por uma determinada tarefa (VIZEL; WEISSENBACHER; MALIK, 2015)

O *Davis-Putnam-Logemann-Loveland*(DPLL), o algoritmo criado por Martin Davis e Hilary Putnam na década de 60 (DAVIS; PUTNAM, 1960), foi refinado por Donald W, George Logemann, Martins Davis (MARTIN, 1962).

O algoritmo DPLL escolhe uma variável da fórmula e define um valor verdade. Simplificando a fórmula, ela entrará em um processo recursivo que fará a verificação até que a fórmula simplificada seja satisfatível. Se ela não for satisfatível, o valor verdade da variável será trocado e a fórmula anterior desfeita, repetindo esse processo (RIBAS, 2011).

Utilizando a Listagem 2.1, podemos utilizar o resolvedor LINGELING para saber se fórmula é satisfatível ou não satisfatível:

```
s SATISFIABLE
```

```
v −1 −2 −3 0
```

Listagem 2.2 – Execução do resolvedor *LINGELING*

A fórmula será satisfeita se todas as cláusulas forem atendidas, a cláusula só será atendida se pelo menos uma de suas literais forem atendidas. O literal positivo será atendido se a variável correspondente receber o valor True (Verdadeiro) e um literal negativo será atendido se for atribuído o valor False (Falso) (CHALLENGE, 1993).

Na saída da Listagem 2.2, temos a letra **s** iniciando a primeira linha, ela representa uma linha de solução, no caso temos **s SATISFIABLE**, que diz que todas as cláusulas foram satisfeitas. A linha iniciada com *v* representa uma linha de variáveis. As variáveis podem ser verdadeiras ou falsas. No exemplo temos **v -1 -2 -3**, que mostra que para o exemplo ser satisfatório e suas cláusulas serem atendidas, todas as suas variáveis tem que ser negadas (CHALLENGE, 1993).

2.2 PLANEJAMENTO CLÁSSICO

A inteligência artificial (IA) é a capacidade de um software ou sistema de computador executar tarefas de forma inteligente, simulando ou emulando o comportamento humano. Uma parte crítica da IA é o seu planejamento, devido à elaboração de um plano de ação para chegar em determinados objetivos (RUSSELL, 2013).

O processo de encontrar sequências de ações que possam atingir metas predefinidas é chamado de Planejamento. O problema é descrito pela descrição de um estado inicial, um conjunto de ações e o objetivo. No planejamento clássico, existe apenas um estado inicial e as ações são determinísticas (BIERE; HEULE; MAAREN, 2009).

Para alcançar esses determinados objetivos são utilizados algoritmos acompanhados principalmente de heurísticas que são derivadas automaticamente da estrutura de representação.

2.2.1 GRAFO DE PLANOS

O grafo de planos aperfeiçoa a representação de informações, o que auxilia a obtenção de outros planos parcialmente ordenados (BLUM; FURST, 1997).

- Planos parcialmente ordenados possuem ações paralelas entre si;
- Planos ordenados não possuem ações paralelas entre si.

O grafo de planos possui dois tipos de nós, de proposições e de ações, que estão ordenados em camadas. Incluindo a camada zero, em que se encontra todos os tipos

de literais positivos do estado inicial, todas as camadas ímpares i+1 são consideradas camadas de proposições. Já nas camadas pares i+2 estão as camadas de ações, suas instâncias e suas precondições devem ser satisfeitas pelas proposições do nível anterior i-1 (SILVA, 2000).

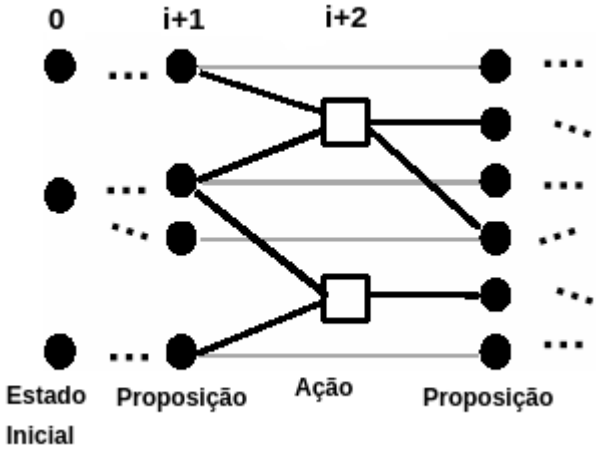

Figura 1 – Expansão grafo de plano, baseado em Ghallab, Nau e Traverso (2004)

No grafo de planos, também é possível notar que há ações mutuamente exclusivas, Mutex, são aquelas que não podem ser executadas em paralelo. Para ações serem consideradas mutuamente exclusivas os efeitos de uma devem negar a precondição ou o efeito da outra. A representação do mutex é feita através de arestas que ligam as ações a um estado subsequente. Duas proposições são consideradas mutex, no grafo de planos, se todas as ações que possuem como efeito estas proposições também forem mutex. As proposições mutuamente exclusivas são um meio de identificação de mutex no grafo de planos (SCHREINER, 2012).

2.2.2 STRIPS

STanford Research Institute Problem Solver (STRIPS) é uma classificação formal para descrever ações (RUSSELL, 2013). Em STRIPS, as ações são especificadas tendo-se duas condições.

A primeira são as precondições da ação, que são as condições que devem ser satisfeitas para que a ação seja executada. A segunda são os efeitos de uma ação, são um conjunto de literais positivos e negativos que a descrevem. Os literais negativos se referem as proposições que devem ser removidas para atingir o estado atual, já os literais positivos descrevem aquelas proposições que deverão ser adicionadas ao novo estado. As proposições que não estão incluídas nas lista de adição (literais positivos) ou na lista de remoções (literais negativos) se mantém inalteradas (FIKES; NILSSON, 1971).

2.2.3 SATPLAN

Kautz, Selman et al. (1992), em 1992, conduziram pesquisas relacionadas a planejamento em IA quando apresentaram o SATPLAN, um planejador que traduzia o conhecimento do GRAPHPLAN ao problema da satisfatibilidade. Tal transformação na forma de representar os problemas se mostrou mais eficiente que o GRAPHPLAN na maioria dos problemas em que os dois sistemas foram comparados. O SATPLAN recebe como entrada um conjunto de esquemas de axiomas e usa algoritmos gerais para encontrar um plano solução (KAUTZ; MCALLESTER; SELMAN, 1996).

Segundo Schreiner (2012), um problema de planejamento no SATPLAN pode ser traduzido para uma instância SAT, desde que haja uma proposição e uma ação que pertencem a um determinado instante de tempo, pois o plano é uma sequência cronológica de ações, uma vez que os problemas de satisfatibilidade são problemas estáticos de valoração.

2.2.3.1 Conversão para instâncias SAT

Ao tentarmos resolver problemas de planificação com satisfabilidade, devemos levar em consideração conjunto de axiomas que possuam propriedade de valoração que permita o conjunto ser satisfatível a um certo plano. As proposições correspondem ao estado inicial e ao objetivo, e outras descrevem as ações (SILVA, 2000).

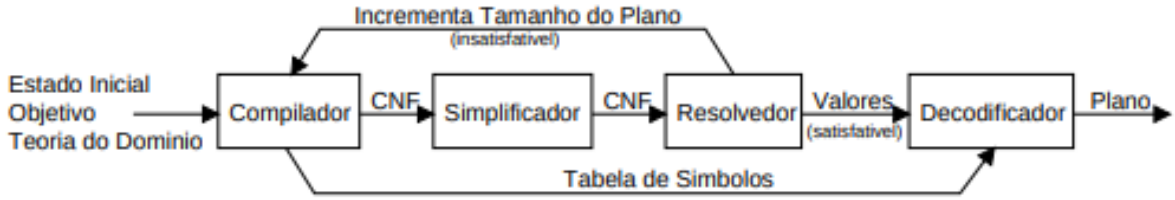

Figura 2 – Planificador baseado em SAT.

FONTE: (SILVA, 2000)

A figura 2 retrata como funcionaria a construção de um planificador que realizaria essa conversão para um problema SAT. O compilador receberá como entrada uma fórmula lógica proposicional na forma CNF, indicando assim um plano de tamanho proposto, se for satisfeita. Após isso, a fórmula em CNF passa-se para um simplificador que tenta deixá-la na forma mais reduzida possível. Atingindo essa menor fórmula, o resolvedor investiga se essa fórmula será satisfeita, atribuindo valores aos literais. Se esses literais satisfazerem a fórmula, um decodificador, mostrará um plano solução, utilizando a tabela de símbolos do compilador, se ela não for satisfeita, o resolvedor a devolve para o compilador para reiniciar o processo (SILVA, 2000).

Segundo Montano e Ribas (2017), um problema de planejamento em STRIPS pode ser definido por uma tríplice P = *I,G,A*. Temos que I é um conjunto de fatos verídicos no estado inicial, G um conjunto de fatos verídicos no estado inicial do objetivo e A são todos as ações que são permitidas. Toda ação a $\in A$ possui precondições (pre (a)), adição de efeitos (add (a)) e exclusão de efeitos (del (a)).

Em SAT, preposições são marcadas com tempo para representar ações e fatos em diversas situações. Sendo a $\in A$ uma ação, então *a(t)* será uma decisão apenas se a ação a for tomada ou não em um instante t. Podemos usar a mesma noção para os fatos, *f(t)* (MONTANO; RIBAS, 2017).

Segundo Sideris e Dimopoulos (2010), para converter um grafo de planos em lógica proposicional, são utilizados subconjuntos das seguintes cláusulas:

1 Cláusulas unitárias para estado inicial e para objetivo;

2 a(t) $\rightarrow$ f(t), para qualquer ação a e fato $f \in$ *pre(a)*;

3 f(t) $\rightarrow$ a_1(t-1) V ... V a_m(t-1), para quaisquer fatos $f \in$ *add*(a_i) e quaisquer ações a_i , m $\geq$ i $\geq$1;

4.1 $\neg a_1(t) \vee a_2(t)$, qualquer par de ações a_1, a_2, assim o conjunto *del*(a_1) $\cap$ *pre*(a_2) não estará vazio;

4.2 $\neg a_1(t) \vee \neg a_2(t)$, qualquer par de ações a_1, a_2, assim o conjunto *del*(a_1) $\cap$ *add*(a_2) não estará vazio;

4.3 $\neg a_1(t) \vee \neg a_2(t)$, caso haja um par de fatos $f_1 \in$ *pre*(a_1), $f_2 \in$ *pre*(a_2), assim f_1, f_2 serão mutuamente exclusivos no momento t

5 $\neg f_1 \vee \neg f_2$, para qualquer par de fatos f_1, f_2 serão mutex no mesmo tempo t

O SATPLAN06 suporta as seguintes codificações:

- SATPLAN06: Cláusulas 1, 2, 5

2.2.3.2 BLACKBOX

O Blackbox, ou *BB*, é um planejador que converte STRIPS em problemas de satisfatibilidade booleana e em seguida os resolve. Ele é uma junção de abordagens baseadas em SAT e em grafos, que resultou em uma instância menor, comparada ao grafo de planos (KAUTZ; SELMAN, 1999).

A conversão de um grafo de planos em lógica proposicional necessita da utilização de alguns subconjuntos das cláusulas, como visto na Subseção 2.2.3.1, que podem ser aplicadas ao BB.

O BB aceita as seguintes codificações para problemas de planejamento:

1. BB-7: Clauses 1, 2, 5
2. BB-31: Clauses 1, 2, 3, 4, 5
3. BB-32: Clauses 1, 2, 3, 4, 5

2.2.3.3 ENTENDENDO O SATPLAN

O SATPLAN, versão 2006, é um *software* escrito na linguagem C, que possibilita que um problema escrito em PDDL através de definições com STRIPS seja resolvido por um SAT *solver* com um planejador, ou seja, o SATPLAN é um intermediador entre eles.

Na Listagem 2.3 é possível notar que há duas funções que não estão entre colchetes, `-problem x` e `-domain x`, sendo essas são as funções necessárias para a execução do SATPLAN. Em ambas, devemos passar o caminho onde os arquivos estão, no caso da primeira passamos o caminho do problema, já na segunda passamos o caminho do domínio:

```
-problem /blocks-strips-typed/instances/instance-1.pddl

-domain blocks-strips-typed/domain.pddl
```

Algumas das funções que foram importantes para o desenvolvimento deste trabalho são:

- `-cnf x` gera um arquivo com o nome x desejado, que tem como saída as fórmulas em CNF de cada camada de decisão que o SATPLAN passou até atingir a camada que satisfaz o problema atribuído a ele;

- `-level x` a utilização desse parâmetro permite que o SATPLAN execute apenas naquela camada x especificada, podendo assim analisar o que acontece em cada camada separadamente;

- `-solver x` utilizando esse parâmetro é possível utilizar o resolvedor de sua escolha, porém antes de utilizar este parâmetro é importante utilizar `-listsolvers`, com esse comando é possível ver os resolvedores que estão disponíveis para se utilizar com o SATPLAN.

Abaixo temos um visão geral dos parametros que podem ser utilizados para execução do SATPLAN:

```
-problem X (:string - problem filename)
-domain X (:string - domain filename)
[-bcheck X (:bool 1 means additional checks)]
[-cnf X (:string - CNF encoding output filename)]
[-cnfonly X (:bool - 1 means don't solve)]
[-iscas X (:bool - 0 normal, 1 means ISCAS)]
[-pbs X (:bool - 0 normal, 1 means PBS)]
[-mhf X (:bool - 0 normal, 1 means MHF)]
[-pure X (:bool - 0 normal, 1 generate pure literals)]
[-prefs X (:integer - 0 off (default), 1 generate one pref
var - first one)]
[-encoding X (:integer - 1 to 4)]
[-globalmemory X (:integer - bytes)]
[-globaltime X (:integer - minutes)]
[-level X (:integer - try this plan level only)]
[-maxlevel X (:integer - try <= this plan level)]
[-options [ X ] (:list - direct input to -solver)]
[-path X (:string - problem and domain directory)]
[-restart X (:integer - #tries to solve at level)]
[-seed X (:integer - for rand())]
[-solution X (:string - solution filename)]
[-solver X (:string - sat solver name)]
[-timeout X (:integer - seconds before new rand())]
[-verbose X (:bool - 0 means OFF, 1 means ON)]
```

Listagem 2.3 – Parâmetros utilizados para execução do SATPLAN

2.3 CONSIDERAÇÕES

Ao decorrer deste capítulo foi possível compreender como transformar ou representar, por meio da lógica, fórmulas em CNF que serão utilizadas por resolvedores. Também conseguimos entender o que é planejamento e qual sua utilidade. Ao utilizar um arquivo com descrição em PDDL conseguimos compreender como resolver um grafo de planos que por meio de resolvedores, que são utilizados pelo SATPLAN, indicam se plano foi satisfeito ou não.

3 RESULTADOS

3.1 ANÁLISE DO SATPLAN

Como foi visto até agora, há diferentes formas de se resolver um problema de planejamento, seja ele através de regressões ou progressões, podendo formalizar a escrita dos problemas utilizando PDDL juntamente com STRIPS. Essas resoluções de problemas são possíveis utilizando o SATPLAN juntamente com os SAT *solvers* e os planejadores.

Dispondo do que já foi visto, desde o que vem à ser Planejamento Clássico até a geração de fórmulas CNF foram levantados questionamentos após a execução de alguns problemas utilizando o *software*.

3.2 ANALISANDO OS PROBLEMAS

Utilizando os problemas encontrados no Potassco (2017) para a realização de testes para entender esses questionamentos, os problemas utilizados são de competições de Planejamento que ocorrem a cada dois anos.

Foram executados 340 testes, a Tabela 3.2 mostra os domínios utilizados e a quantidade de instâncias disponíveis para cada um deles.

Como são problemas conhecidos foi possível encontrar quais as camadas que os problemas se tornam satisfatíveis[1]. Em um primeiro momento foram rodados todos os 340 testes sem nenhum critério. Os testes foram executados em uma máquina com 16 *giga bytes* de memória ram, com um processador *Intel Core* I7 de 3ª geração.

Ao decorrer da execução desses testes, encontramos uma possível proposta de melhoria de tempo na execução do SATPLAN.

Utilizando o domínio do Mundo dos Blocos, apresentado nas Figuras 3 e 4, foram executados testes na última camada de cada instância, que é a camada que torna o problema satisfatível.

Na Figura 3 temos o tempo de execução total do SATPLAN representado pela linha azul e o tempo de geração e impressão das fórmulas CNF representada pela linha vermelha. Contudo ele não executa apenas a geração das fórmulas o que dificulta em alguns casos, como no da instância 35, a identificar a proposta de melhoria de tempo observada. A instância 35 foi executadas diversas vezes, mostrando sempre um alto tempo de execução do SATPLAN.

[1] Camadas em que os problemas mais conhecidos se tornam satisfatíveis <https://l1nk.dev/QC7jd>

Domínios	Quantidade de Instâncias
Depósito Automático	15
Driverlog Automático	15
Elevador Não Tipado	42
Elevador Tipado	41
Freecell Não Tipado	17
Freecell Tipado	42
Garra Rodada 1 ADL	17
Garra Rodada 1 STRIPS	6
Lanche Infantil Ágil	6
Lanche Infantil Ótima	16
Logística Não Tipado	5
Logística Rodada 1	6
Logística Rodada 2	12
Mundo dos Blocos Não Tipado	22
Mundo dos Blocos Tipado	22
Mundo dos Canos Sem Tanque Não Temporal	50
Veículos Robóticos	27
Total	340

Tabela 1 – Domínios utilizados nos testes

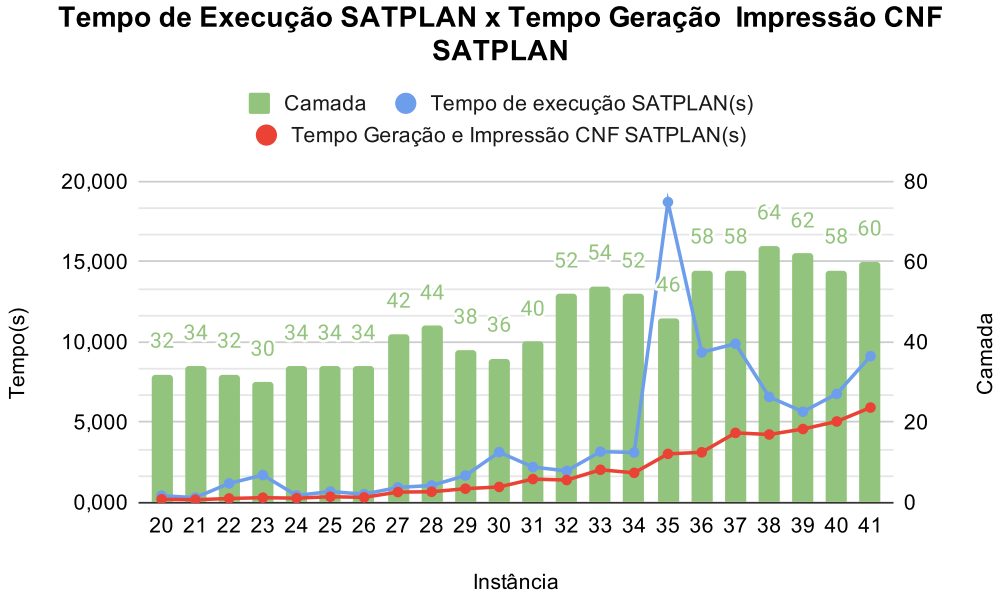

Figura 3 – Tempo de Execução do SATPLAN x Tempo Geração e Impressão CNF SATPLAN(Mundo dos Blocos Tipado)

Observando o fato do SATPLAN fazer diversas operações e chamadas , tentou-se executar de forma isolada o *BB* para analisar o tempo de resolução do problema, pois assim, ele irá resolver o problema e imprimir apenas a fórmula que torna o problema satisfatível.

Na Figura 4 temos o *BB* sendo executado de forma separada do SATPLAN, o que nos permite identificar com uma precisão maior o tempo gasto na resolução dos problemas.

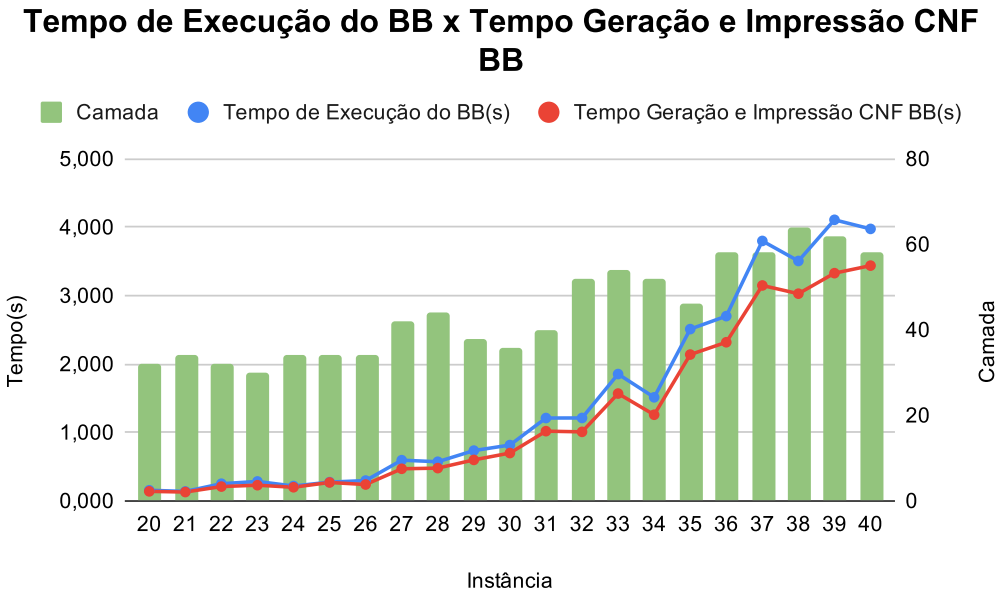

Figura 4 – Tempo de Execução do BB x Tempo Geração e Impressão CNF BB(Mundo dos Blocos Tipado)

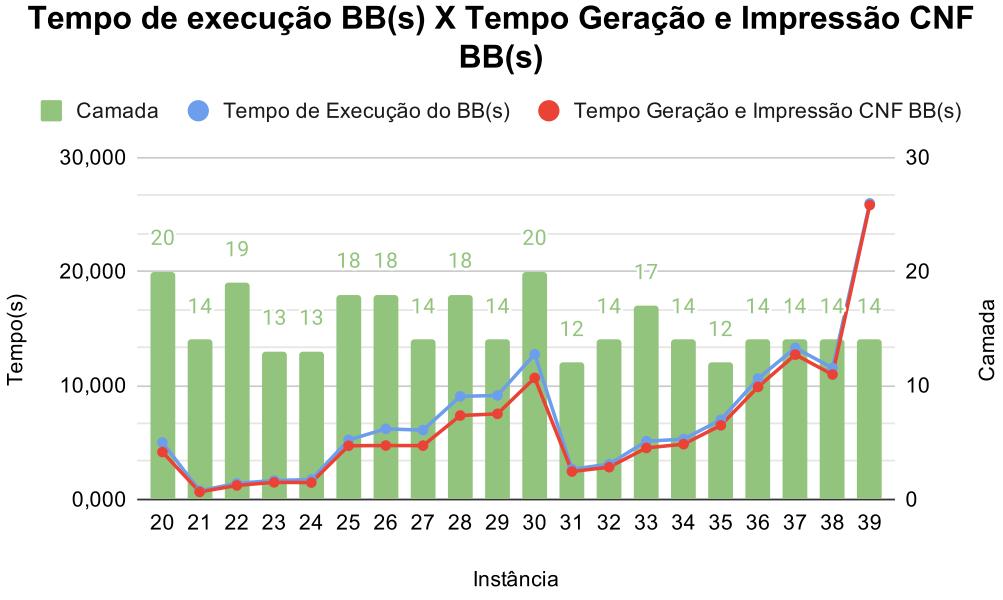

Figura 5 – Tempo de Execução do BB x Tempo Geração e Impressão CNF BB(Mundo dos Canos Não Tipado)

Na Figura 5, temos um exemplo de um domínio do Mundo dos Canos, diferente do apresentado na Figura 4. Ele possui a mesma semelhança no tempo de execução total

do *software* com o tempo de resolução do problema proposto, sendo mínima a diferença entre eles.

Com essas observações foi indagado à possibilidade da redução desse tempo de resolução do problema, fazendo assim com que o SATPLAN efetue essa resolução rapidamente, diminuindo assim o seu tempo.

3.3 ESTUDANDO A HIPÓTESE

No capítulo anterior, foi levantada a hipótese sobre a quantidade de tempo gasta na geração das fórmulas. Para tentarmos entender melhor foram utilizados alguns recursos do *GNU Compiler Collection* (GCC) para a análise dos tempos, além dos próprios tempos já fornecidos pelo SATPLAN.

3.3.1 PROFILE

Realizar um *profile* de um *software*, é verificar qual a quantidade de recurso computacional e a quantidade de tempo é gasta por cada parte do código. Uma das maneiras de realizar *profiles* é através de ferramentas que possam medir o tempo. O SATPLAN já possui o *profile* de algumas funções importantes que ele executa.

A Listagem 3.1 é um exemplo da saída fornecida pelo SATPLAN. Essa saída nos permite analisar o tempo gasto para geração das fórmulas através, do 'CNF output time'. E através do 'real' temos o tempo de execução total do *software*.

```
time spent:

0.00 seconds instantiating 8064 action templates
0.00 seconds reachability analysis, yielding 656 facts and 1696 actions
0.01 seconds collecting 656 relevant facts
0.00 seconds building connectivity graph
0.00 seconds building (std) graph
13.97 seconds CNF output time
13.98 seconds total planner time (solving not included)

real    0m16,264s
user    0m12,651s
sys     0m1,441s
```

Listagem 3.1 – Saída fornecida pelo *BB*

Contudo, apenas os tempos gerados pelo SATPLAN, não são conclusivos para comprovar a hipótese levantada. A utilização dos recursos provindos do GCC foram essenciais para essa comprovação.

3.3.1.1 GProf

O *GNU profiling* (*GProf*) é uma ferramenta que faz parte do GCC, que é utilizado para medir o tempo gasto por funções dentro de códigos na linguagem de programação C. Ao utilizar essa ferramenta, ela irá gerar um arquivo de saída *gmon.out*, nele estará quantidade de tempo gasta por cada função, e também a quantidade de vezes que a função foi chamada.

Na Listagem 3.2 temos a saída do *GProf*, que nos mostra as funções em ordem decrescente com a porcentagem de tempo gasto, o tempo cumulativo, o tempo em segundos, a quantidade de chamadas, o tempo gasto em cada chamada e o tempo total gasto em cada chamada respectivamente.

```
%   cumulative   self              self     total
time   seconds   seconds    calls  s/call   s/call  name
26.97      3.43     3.43                             vfprintf
16.98      5.59     2.16        1     2.16     3.49  print_thin_gp_based_encoding
8.41       6.66     1.07                             _IO_new_file_xsputn
7.70       7.64     0.98                             _int_malloc
6.76       8.50     0.86                             _itoa_word
6.68       9.35     0.85                             write
6.05      10.12     0.77  2764776     0.00     0.00  interfere
```

Listagem 3.2 – Saída fornecida pelo *GProf*

3.3.1.2 Clock()

A função *clock()* faz parte da biblioteca *time.h*. Com ela é possível a tomada de tempos utilizando-a dentro de trechos do código, conseguindo assim verificar a quantidade de tempo gasto pelas funções que desejarmos. O resultado de saída é impresso em milissegundos, tendo assim que fazer a conversão para segundos para termos um dado coerente com o restante, que já estão nessa unidade.

3.3.1.3 Buffer

Buffer é armazenamento temporário usado durante transferências de processos. No nosso contexto ele é utilizado para armazenar uma string e escreve-la no arquivo, quando atingido seu tamanho máximo.

3.4 ANALISANDO OS DADOS

Ao utilizar os dados fornecidos pelo SATPLAN, como o tempo total de execução e o tempo gasto em algumas funções, juntamente com os tempos obtidos pela utilização do *GProf*, que fornece uma lista com o tempo total gasto por cada função e quais funções consomem mais tempo, foi possível realizar um estudo aprofundado sobre o motivo pelo qual o SATPLAN apresenta um tempo elevado de execução.

Quando utilizamos o *GProf*, foi possível identificar as principais funções que possuem um alto consumo de tempo dentro do SATPLAN. As Listagens 3.3 e 3.4 são exemplos de saída do GProf que mostram as principais funções responsáveis pelo consumo de tempo.

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
27.24	2.37	2.37				vfprintf
11.61	3.38	1.01	1	1.01	2.07	print_thin_gp_based_encoding
9.37	4.20	0.82				_IO_new_file_xsputn
8.62	4.95	0.75				_itoa_word
7.82	5.63	0.68				_int_malloc
6.78	6.22	0.59	2764776	0.00	0.00	interfere
5.52	6.70	0.48				write
4.60	7.10	0.40				fwrite

Listagem 3.3 – Saída fornecida pelo *GProf*

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
24.32	0.99	0.99				vfprintf
14.74	1.59	0.60	1	0.60	1.10	print_thin_gp_based_encoding
9.09	1.96	0.37				_itoa_word
7.99	2.29	0.33				_IO_new_file_xsputn
7.37	2.59	0.30				_int_malloc
7.13	2.88	0.29	1642578	0.00	0.00	interfere
4.42	3.06	0.18				__strchrnul_sse2

Listagem 3.4 – Saída fornecida pelo *GProf*

Como podemos ver nas Listagens 3.3 e 3.4, as mesmas funções se repetem, *vprintf* e *print_thin_gp_based_encoding*. Cada instância dentro de um domínio é considerado um teste, e em todos eles essas funções estão em evidência no topo da lista gerada pelo *GProf*.

Conseguindo identificar quais as funções que estavam consumindo mais tempo dentro do software, foi possível começar a utilizar a função *clock()* para confirmar e correlacionar com os dados adquiridos pelo *GProf*. Os tempos fornecidos pela função foram impressos no final do arquivo em que as fórmulas CNF são gravadas.

A Listagem 3.5 mostra como foi utilizada a função *clock()* e onde se encontra a função de impressão das fórmulas CNF. Na Listagem utilizamos duas variáveis *execution_time*, que armazena o tempo da função *print_thin_gp_based_encoding*, e *specific_time* amazena o tempo de escrita da fórmula no arquivo por camada.

```
1  clock_t start, end, specific_end, specific_start;
2  double execution_time, specific_time;
3  start = clock();
4        .
5        .
6        .
```

```
7     specific_start = clock();
8   for (i = 0; i < lnum_clauses; i++) {
9     for (j = 0; j < lclause_size[i]; j++) {
10      fprintf(CNF, "%d ", lclause[i][j]);
11    }
12    fprintf(CNF, "0\n");
13  }
14        .
15        .
16        .
17  end = clock();
18  execution_time = ((double)(end - start))/CLOCKS_PER_SEC;
19  specific_time = ((double)(specific_end - specific_start))/CLOCKS_PER_SEC;
20
21  printf("\nExectuion time: %fs\n", execution_time);
22  printf("\nSpecific time: %fs\n", specific_time);
```

Listagem 3.5 – Utilização da função clock

Nas Figuras 4 e 5, foi utilizado apenas o *BB* para demonstrar que os tempos de execução, o tempo de resolução e o tempo da impressão da fórmula do problema possuem uma diferença mínima entre eles.

Para obter uma visão mais detalhada, podemos observar as Figuras 6 e 7, que foram alguns dos testes feitos utilizando o *BB* e todos os recursos de *profiles* vistos.

Na Figura 6 temos o tempo de execução total do *BB* representado pela linha azul, o tempo de geração e impressão da função CNF representado pela linha vermelha, o tempo de impressão da fórmula pela linha amarela e as instâncias pelas colunas verdes. Analisando as linhas do tempo de resolução da fórmula com o tempo em que ela é impressa, podemos notar que a diferença entre os tempos é mínima.

Essa diferença de tempo ocorre em diferentes domínios, como pode ser visto nas Figuras 7 e 8.

A função *vfprintf*, apresentada na Listagem 3.3, é a função que imprime a fórmula CNF no arquivo. Esse consumo de tempo elevado para imprimir a fórmula ocorre devido ao seu tamanho. A Figura 9 representa o domínio do Mundo dos Blocos Tipado, nela está expressa o tamanho de cada fórmula na camada de resolução do problema, ou seja a última camada. Como pode ser observado a linha vermelha indica o tamanho de cada fórmula. O tamanho de cada fórmula pode variar de acordo com o tamanho do problema proposto ao *BB*, isso pode ser observado na instância 33 e na instância 34, onde o tamanho da fórmula da primeira é maior.

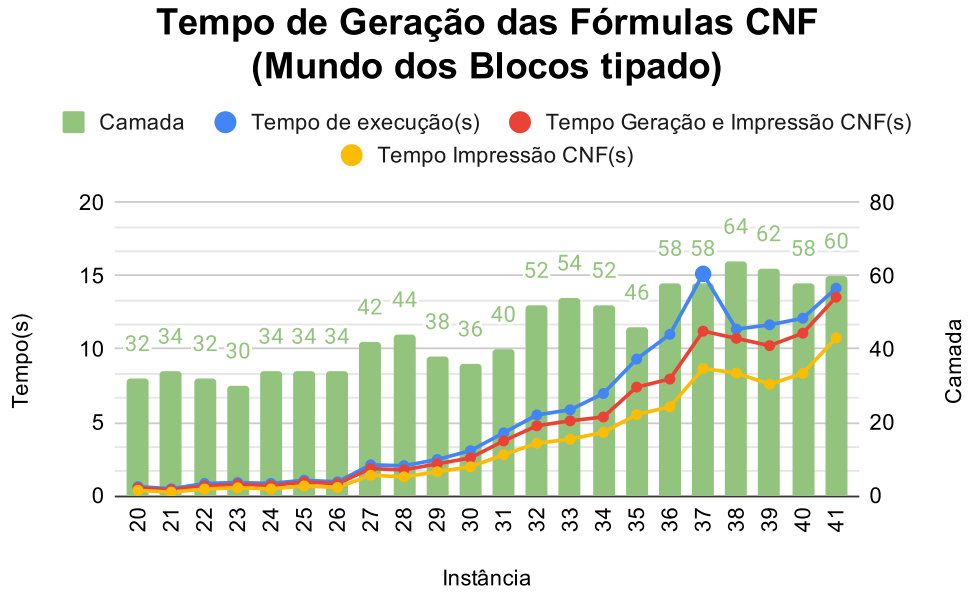

Figura 6 – Tempo de Geração da fórmula CNF em cada instância do domínio do Mundo dos Blocos Tipado

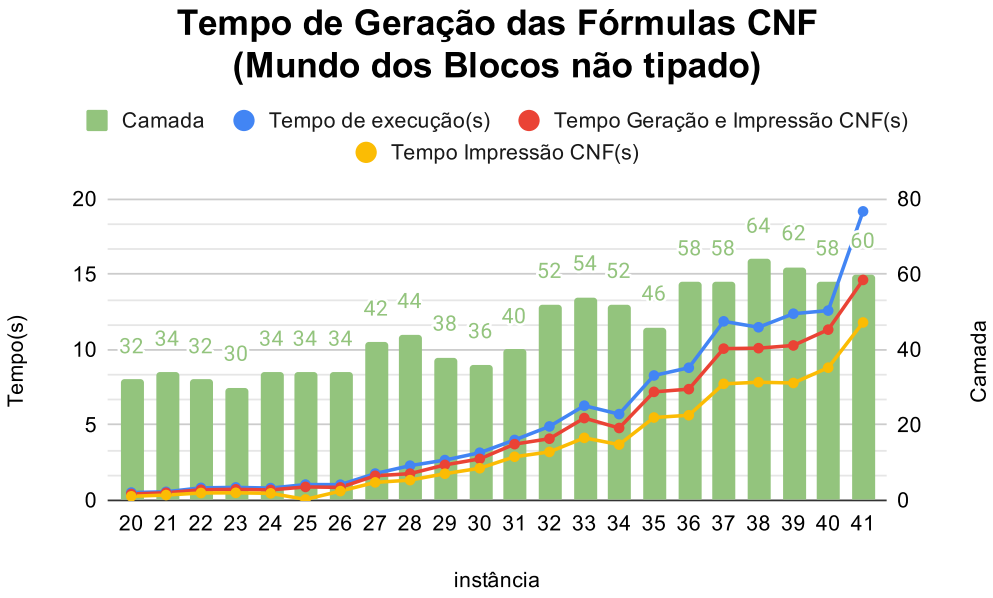

Figura 7 – Tempo de Geração da fórmula CNF em cada instância do domínio do Mundo dos Blocos Não Tipado

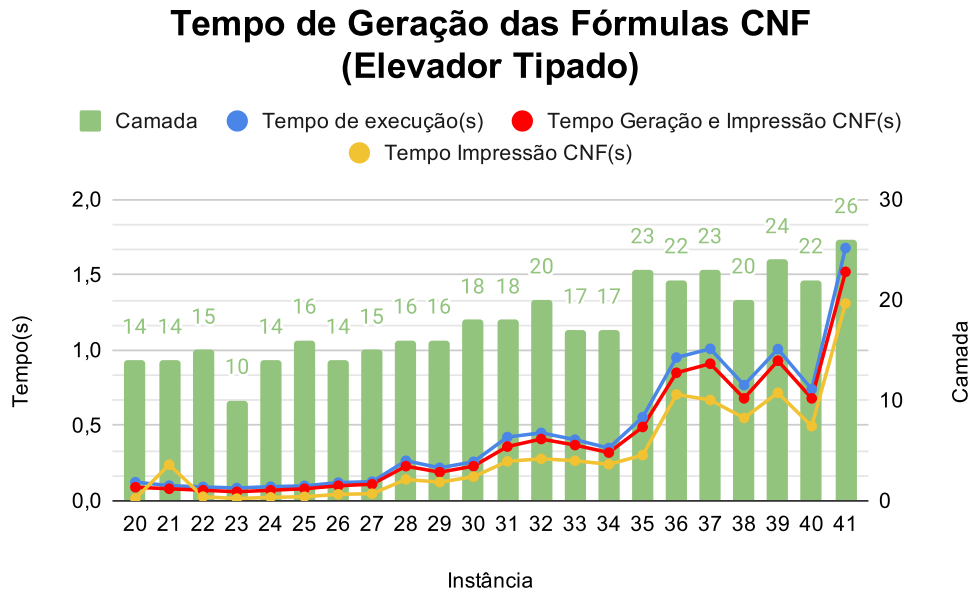

Figura 8 – Tempo de Geração da fórmula CNF em cada instância do domínio do Elevador Tipado

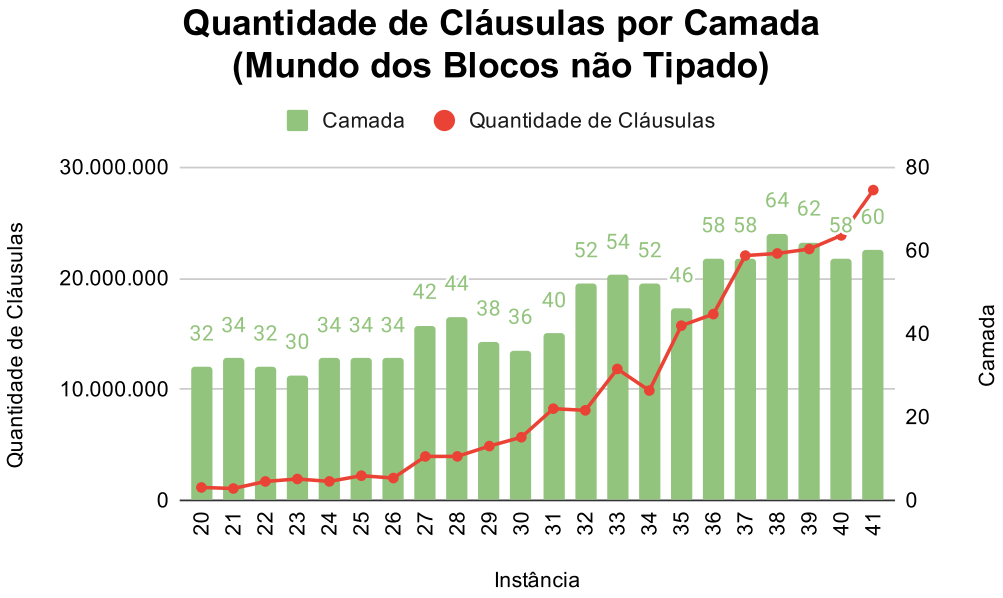

Figura 9 – Quantidade de Cláusulas por instância do domínio do Mundo dos Blocos Tipado

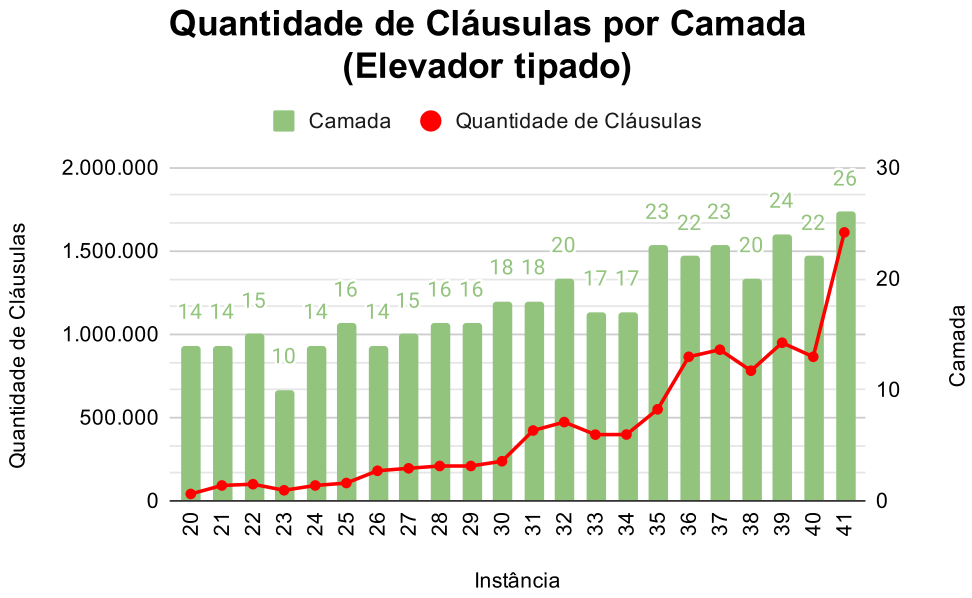

Figura 10 – Quantidade de Cláusulas por instância do domínio do Elevador Tipado

Observando as Figuras 9 e 10 conseguimos perceber que devido ao tamanho das fórmulas há um grande gasto no tempo para gravá-las em um arquivo.

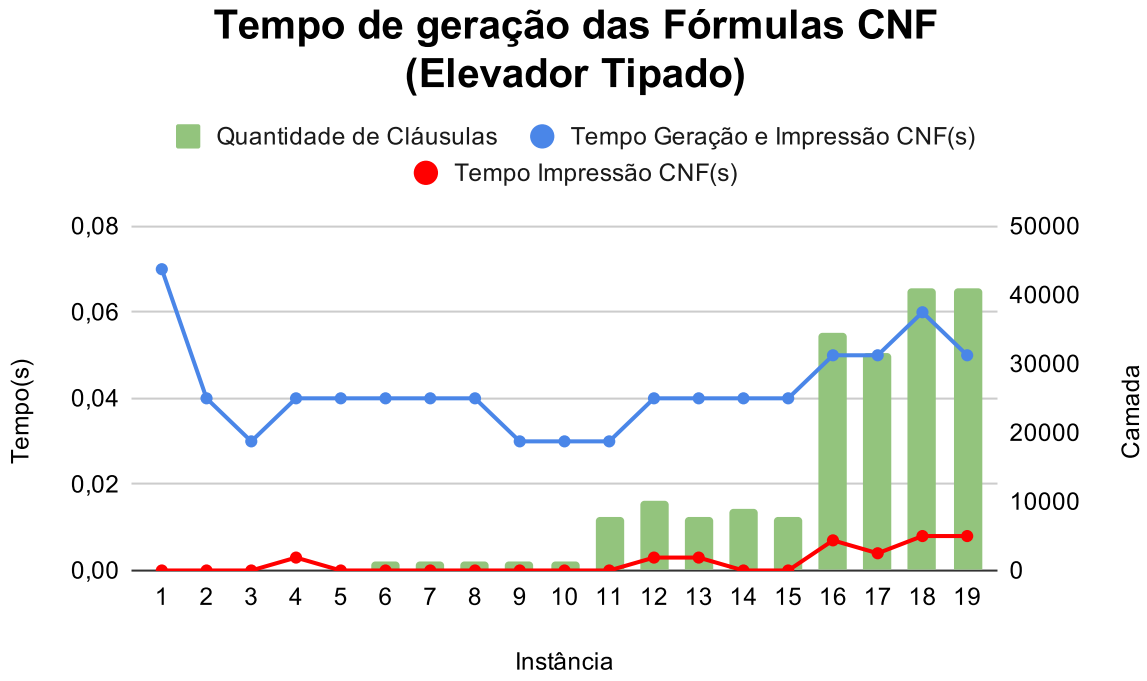

Figura 11 – Tempo de impressão x Quantidade de Cláusulas

O problema enfrentado pelo SATPLAN ocorre devido ao tamanho das fórmulas

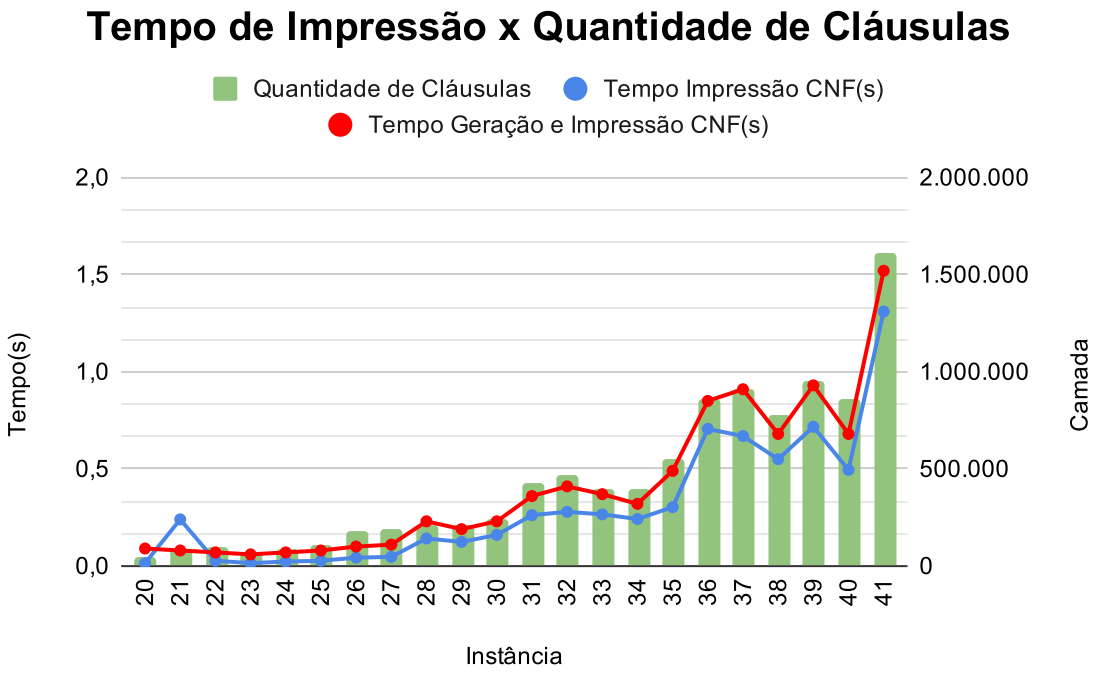

Figura 12 – Tempo de impressão x Quantidade de Cláusulas

CNF. A Figura 11 e 12, demonstram esse problema, a quantidade de cláusulas estão representadas pelas barras verdes, a linha vermelha representa o tempo de execução total do BB e a linha azul o tempo que a fórmula leva para ser impressa. Na primeira Figura temos instâncias com a quantidade de cláusulas pequenas se comparado com a quantidade da segunda Figura. Analisando essas Figuras pode-se perceber que quanto maior o número de cláusulas maior é o tempo para a impressão da fórmula.

Vendo que a maior parte do tempo para resolução das fórmulas está em sua impressão, buscou-se a parte do código em que é realizada a impressão. Na Listagem 3.6 essa parte do código que realiza a escrita no arquivo. Nas linhas 1 e 2 temos a percursão da lista para encontrar a cláusula, na linha 4 temos a impressão das cláusulas, a linha 5 indica o final de cada cláusula. A fórmula é preenchida, e cada cláusula é encerrada com o número '0' como visto na Subseção 2.1.1.

A listagem 3.6 temos o grande problema desse tempo elevado. Como são feitas impressões de strings muito grandes, o código dessa forma acaba formando um gargalo, pois a função é chamada várias vezes para escrever no arquivo. Utilizando dessa forma para escrever no arquivo podemos enfrentar alguns problemas como um elevado consumo de memória, o tempo de execução pode se tornar prolongado, pois quanto maior for o tamanho da string, maior será o tempo necessário para percorrer e imprimir cada um dos elementos.

```
1 for (i = 0; i < lnum_clauses; i++) {
```

```
2     for (j = 0; j < lclause_size[i]; j++) {
3       fprintf(CNF, "%d ", lclause[i][j]);
4         }
5   fprintf(CNF, "0\n");}
```

Listagem 3.6 – Laço de repetição analisado no trabalho

Instância	Camada	Tamanho Arquivo(MB)	Tempo(s)	Cláusulas
27	42	21	0,175	1.390.267
28	44	21	0,174	1.391.503
29	38	25	0,215	1.711.472
30	36	30	0,250	1.993.338
31	40	43	0,363	2.881.834
32	52	43	0,357	2.823.015
33	54	62	0,519	4.107.541
34	52	52	0,432	3.433.188
35	46	82	0,689	5.455.514
36	58	88	0,734	5.804.819
37	58	116	0,960	7.611.734
38	64	117	0,972	7.674.098
39	62	119	0,982	7.795.349
40	58	125	1,038	8.219.223
41	60	146	1,211	9.605.558
42	72	179	1,482	11.757.432
43	78	229	1,891	14.983.978
44	68	189	1,566	12.391.695
45	72	341	2,740	21.910.778
46	66	276	2,252	18.022.388
47	76	399	3,177	25.317.121

Tabela 2 – Tabela demonstrativa do tamanho do arquivo, tempo de execução da última camada e a quantidade das cláusulas.

Na Tabela 2, estão registrados os tempos de impressão das fórmulas, em segundos, juntamente com as seguintes informações, a camada em que o problema se torna satisfatível, o tamanho do arquivo em *megabytes* e a quantidade de cláusulas presentes na camada em questão. Esses levantamentos fornecem uma visão do desempenho da impressão das fórmulas no SATPLAN, permitindo uma análise mais detalhada dos resultados.

3.4.1 Alterações utilizadas

Visto que temos um gargalo, foram testadas algumas alterações no código 3.6 para tentativa de redução desse tempo de impressão das fórmulas. Para esses testes utilizamos uma Máquina Virtual, onde teríamos poucos processos sendo executados simultaneamente aos testes, dando assim uma maior precisão para os testes.

Dado o gargalo identificado, foram realizados testes com algumas alterações no código da Listagem 3.6 na tentativa de reduzir o tempo de impressão das fórmulas. Para esses testes, utilizamos uma Máquina Virtual de 16 *gigabytes* de ram e um processador *Intel Core* I7-9700, na qual havia poucos processos sendo executados simultaneamente durante os testes, proporcionando uma maior precisão nos resultados.

```
const int BUFFER_SIZE = 1024*30;
char buffer[BUFFER_SIZE];
int i, j;
int pos = 0;

for (i = 0; i < lnum_clauses; i++) {
    for (j = 0; j < lclause_size[i]; j++) {
        pos += sprintf(&buffer[pos], "%d ", lclause[i][j]);
        if (pos >= BUFFER_SIZE - 10) {
            fwrite(buffer, pos, sizeof(char), CNF);
            pos = 0;
        }
    }
    pos += sprintf(&buffer[pos], "0\n");
    if (pos >= BUFFER_SIZE - 10) {
        fwrite(buffer, pos, sizeof(char), CNF);
        pos = 0;
    }
}
if (pos > 0) {
    fwrite(buffer, pos, sizeof(char), CNF);
    }
```

Listagem 3.7 – Laço de repetição alterado para utilização de buffer

Tendo a Tabela 2 como refêrencia dos tempos sem alterações, uma das alternativas foi a utilização de *buffers*. A utilização do buffer se deu na tentiva de armazenar toda a fórmula CNF gerada, com isso ela seria escrita no arquivo utilizando menos chamadas possíveis do *fprintf()*.

Alteramos também o *fprintf()* pela função *fwrite()*, essa alteração foi feita, devido ao fato da primeira função precisar interpretar o formato especificado e converter os elementos para uma sequência de caracteres para depois gravá-las no arquivo, já a segunda função, ela grava os *bytes* diretamente da memória para o arquivo, tornando-o mais rápido que a primeira. Vale ressaltar que os tempos expressos na Tabela 3 são apenas da escrita das fórmulas no arquivo.

Utilizando desse método, conseguimos notar uma pequena diferença no tempo de escrita das fórmulas no arquivo, com podemos observar na tabela 3. Foi utilizado o Mundo dos Blocos Tipado para exemplificação.

Instância	**Sem Alterações**	**1024*20**	**1024*30**	**1024*40**
27	0,175	0,174	0,143	0,143
28	0,174	0,179	0,143	0,143
29	0,215	0,219	0,17	0,178
30	0,250	0,251	0,205	0,204
31	0,363	0,363	0,297	0,297
32	0,357	0,357	0,294	0,295
33	0,519	0,518	0,425	0,426
34	0,432	0,433	0,355	0,354
35	0,689	0,687	0,564	0,564
36	0,734	0,738	0,684	0,685
37	0,960	0,962	0,790	0,791
38	0,972	0,980	0,795	0,794
39	0,982	0,983	0,809	0,808
40	1,038	1,042	0,851	0,851
41	1,211	1,208	0,996	0,995
42	1,482	1,477	1,217	1,219
43	1,891	1,885	1,553	1,554
44	1,566	1,560	1,284	1,284
45	2,740	2,740	2,278	2,327
46	2,252	2,255	1,873	1,870
47	3,177	3,312	2,646	2,647

Tabela 3 – Tabela comparativa de tempo, em segundos, utilizando Mundo dos Blocos Tipado

Na Tabela 3, temos as instâncias utilizadas, os tempos expressos em segundos nos seguintes cenários: código sem alterações, *buffer(1024*20)*, *buffer(1024*30)* e *buffer(1024*40)*. Os *buffers* possuem pesos (20/30/40) devido ao fato de serem expressos em *bytes*. Tentamos utilizar tamanhos maiores para o *buffer*, como 1024*50,80 ou 150, porém não conseguimos identificar nenhuma diferença significativa entre esses pesos em comparação com o peso de 1024*30, com esse peso foi possível observar um ganho 6,81 a 18,29% , tornando-o a melhor opção para a comparação dos tempos.

Outro ponto observado foi nas instâncias 45, 46 e 47, onde foi necessário utilizar o parâmetro *-globalmemory X"*, em que X representa o tamanho expresso em bytes para a quantidade de memória que o *BB* pode utilizar na resolução daquele problema. Isso ocorreu devido ao tamanho máximo já atribuído ao *BB* para a quantidade de memória alocada durante a resolução de um problema específico. Caso essa configuração não seja utilizada, o *BB* apresenta um erro de segmentação, Figura 13, nas *layers* 54,59 e 52 respectivamente. Para evitar esse erro, utilizamos um tamanho de *22*10^9 conforme mostrado na Listagem 3.8, permitindo que o *BB* concluísse a resolução do problema.

```
./satplan -domain .../domain.pddl -problem .../instance-46.pddl
```

```
–globalmemory 22000000000
```

Listagem 3.8 – Utilização parâmetro *'-globalmemory X'*

```
exclusion constraints layer 25...
exclusion constraints layer 26...
exclusion constraints layer 27...
exclusion constraints layer 28...
exclusion constraints layer 29...
exclusion constraints layer 30...
exclusion constraints layer 31...
exclusion constraints layer 32...
exclusion constraints layer 33...
exclusion constraints layer 34...
exclusion constraints layer 35...
exclusion constraints layer 36...
exclusion constraints layer 37...
exclusion constraints layer 38...
exclusion constraints layer 39...
exclusion constraints layer 40...
exclusion constraints layer 41...
exclusion constraints layer 42...
exclusion constraints layer 43...
exclusion constraints layer 44...
exclusion constraints layer 45...
exclusion constraints layer 46...
exclusion constraints layer 47...
exclusion constraints layer 48...
exclusion constraints layer 49...
exclusion constraints layer 50...
exclusion constraints layer 51...
exclusion constraints layer 52...
exclusion constraints layer 53...
exclusion constraints layer 54...
exclusion constraints layer 55...
exclusion constraints layer 56...
exclusion constraints layer 57...
exclusion constraints layer 58...
exclusion constraints layer 59...
Segmentation fault (core dumped)

--- 35584 --- 139 ---

Memory limit exceeded: bb system call, OR Time limit exceeded: bb system call, OR bb internal error
```

Figura 13 – Erro de segmentação apresentado pelo *BB*

3.5 CONSIDERAÇÕES

Foram executados diversos testes para confirmar a possibilidade de melhoria de tempo do SATPLAN. No capítulo, conseguimos identificar com o auxílio das figuras, que o tempo de execução do BB e o tempo para geração e impressão das fórmulas CNF são praticamente idênticos o que mostra que é possível melhorar esse tempo, contudo é necessário ter dados mais precisos para entender como aplicar essa melhoria.

Com os testes foi possível identificar o problema de desempenho dentro do SATPLAN. Esse problema de desempenho ocorre devido ao tamanho das fórmulas CNF e ao fato de a função de impressão dessas fórmulas ser chamada várias vezes para gravá-las em arquivo, resultando em uma sobrecarga de memória.

Para mitigar esse problema, foi testada a utilização de *buffers* como uma alternativa. Essa abordagem consistiu em consolidar as operações de escrita em uma única

chamada, reduzindo o *overhead* de chamada de função e minimizando a quantidade de operações de I/O necessárias. Como resultado, foram observadas melhorias nos tempos de escrita das fórmulas, o que contribuiu para otimizar o desempenho geral do SATPLAN.

4 CONCLUSÃO

Durante o desenvolvimento deste trabalho, foi possível compreender os conceitos e a aplicação do Planejamento na resolução de problemas em instâncias SAT. Com base nesse conhecimento, realizou-se um estudo com o objetivo de melhorar o tempo de resolução dos problemas no SATPLAN. Para identificar essas melhorias, utilizou-se o *GProf* em conjunto com a função *clock()*, permitindo a identificação da função que consumia a maior parte do tempo no SATPLAN.

Ao analisar a lista de funções gerada pelo *profiling*, identificou-se que a função *vprintf* era responsável por uma parcela significativa do tempo de execução. Compreendeu-se, então, as operações de gravação em arquivos realizadas pelo software. O próprio *profiling* do SATPLAN também foi uma ferramenta valiosa, fornecendo informações sobre a quantidade de cláusulas presentes na fórmula CNF em cada camada do problema.

Com o conhecimento adquirido e a identificação do gargalo, utilizaram-se *buffers* para reduzir a quantidade de chamadas à função de gravação da fórmula em arquivo. Essa abordagem resultou em uma melhoria no tempo de gravação, como observado na análise dos resultados.

Dessa forma, o problema de tempo relacionado à função de gravação no arquivo foi solucionado por meio da utilização de *buffers*, demonstrando uma otimização no processo de escrita das fórmulas no SATPLAN.

Referências

BEDEGRAL, B.; ACIÓLY, B. M. Introduçaoa lógica clássica para a ciência da computaçao. *Versao preliminar, Natal*, 2007. Citado na página 14.

BIERE, A.; HEULE, M.; MAAREN, H. van. *Handbook of satisfiability*. [S.l.]: IOS press, 2009. v. 185. Citado 2 vezes nas páginas 14 e 16.

BLUM, A. L.; FURST, M. L. Fast planning through planning graph analysis. *Artificial intelligence*, Elsevier, v. 90, n. 1-2, p. 281–300, 1997. Citado na página 16.

CHALLENGE, D. Satisfiability: Suggested format. *DIMACS Challenge. DIMACS*, 1993. Citado 2 vezes nas páginas 15 e 16.

DAVIS, M.; PUTNAM, H. A computing procedure for quantification theory. *Journal of the ACM (JACM)*, ACM, v. 7, n. 3, p. 201–215, 1960. Citado na página 15.

FIKES, R. E.; NILSSON, N. J. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, Elsevier, v. 2, n. 3-4, p. 189–208, 1971. Citado na página 17.

GHALLAB, M.; NAU, D.; TRAVERSO, P. *Automated Planning: theory and practice*. [S.l.]: Elsevier, 2004. Citado 2 vezes nas páginas 7 e 17.

KAUTZ, H.; MCALLESTER, D.; SELMAN, B. Encoding plans in propositional logic. 1996. Citado na página 18.

KAUTZ, H.; SELMAN, B. Unifying sat-based and graph-based planning. In: *IJCAI*. [S.l.: s.n.], 1999. v. 99, p. 318–325. Citado na página 19.

KAUTZ, H. A.; SELMAN, B. et al. Planning as satisfiability. In: CITESEER. *ECAI*. [S.l.], 1992. v. 92, p. 359–363. Citado na página 18.

MARTIN, D. A machine program for theorem-proving. *Communications of the ACM*, v. 5, n. 7, p. 397, 1962. Citado na página 15.

MONTANO, R. A.; RIBAS, B. C. Planning as mixed-horn formulas satisfiability. 2017. Citado 2 vezes nas páginas 15 e 19.

POTASSCO. *PDDL Benchmark Instances*. 2017. Disponível em: <https://github.com/potassco/pddl-instances>. Acesso em: 9 nov. 2019. Citado na página 22.

RIBAS, B. C. Satisfatibilidade não-clausal restrita às variáveis de entrada. 2011. Citado na página 15.

RIBAS, B. C. Um método de pré-processamento de fórmulas sat e pseudo-boolean baseado em técnicas de programação linear inteira mista. 2015. Citado na página 14.

RUSSELL, S. J. R. *Stuart J.(Stuart Jonathan), 1962-Inteligência artificial/Stuart Russell, Peter Norvig; tradução Regina Célia Simille*. [S.l.]: Rio de Janeiro: Elsevier, 2013. Citado 2 vezes nas páginas 16 e 17.

SCHREINER, M. A. Planejamento por satisfatibilidade clausal e não-clausal baseado na rede de planos. 2012. Citado 3 vezes nas páginas 13, 17 e 18.

SIDERIS, A.; DIMOPOULOS, Y. Constraint propagation in propositional planning. In: *Proceedings of the International Conference on Automated Planning and Scheduling*. [S.l.: s.n.], 2010. v. 20, p. 153–160. Citado na página 19.

SILVA, F. Algoritmo para planificação baseada em strips. 2000. Citado 2 vezes nas páginas 17 e 18.

THIFFAULT, C.; BACCHUS, F.; WALSH, T. Solving non-clausal formulas with dpll search. In: SPRINGER. *International Conference on Principles and Practice of Constraint Programming*. [S.l.], 2004. p. 663–678. Citado na página 14.

VIZEL, Y.; WEISSENBACHER, G.; MALIK, S. Boolean satisfiability solvers and their applications in model checking. *Proceedings of the IEEE*, IEEE, v. 103, n. 11, p. 2021–2035, 2015. Citado na página 15.

Apêndices

As Figuras abaixo representam os testes feitos, que evidenciam o problema do tempo de geração das fórmulas dentro do SATPLAN exposto no Capítulo 3.1.

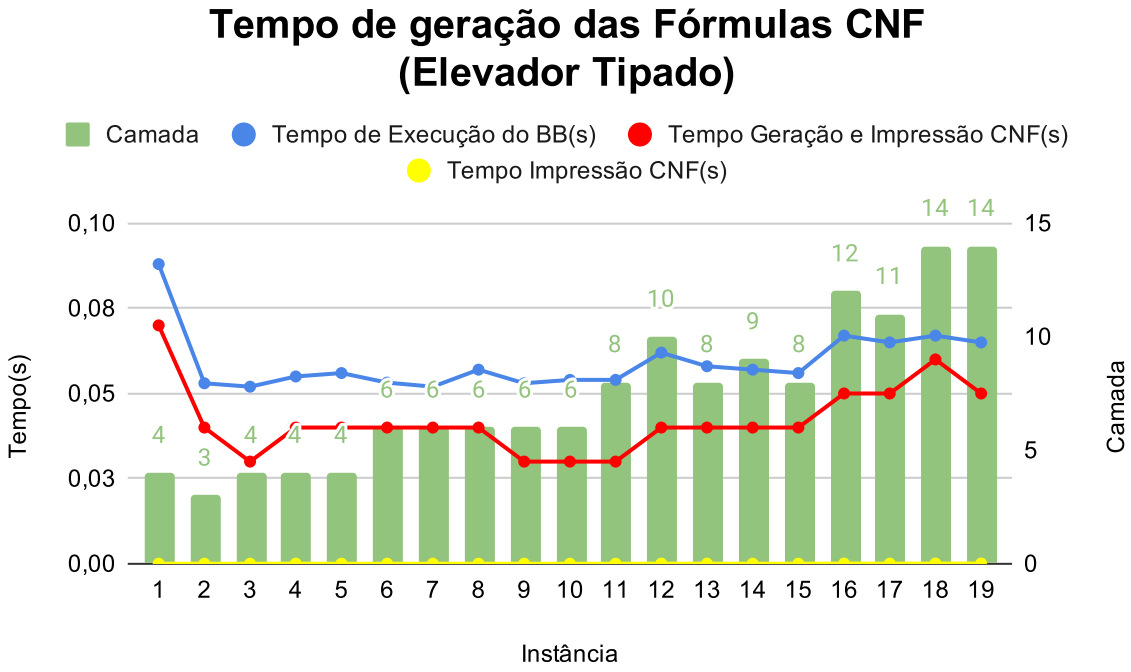

Figura 14 – Tempo de impressão da fórmula CNF em cada instâcia do domínio do Elevador Tipado

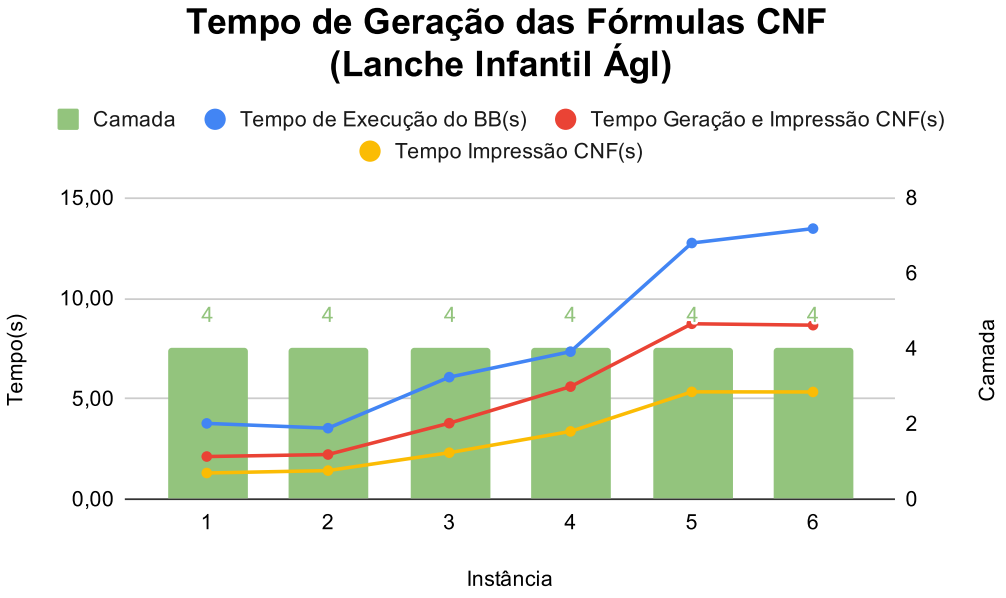

Figura 15 – Tempo de impressão da fórmula CNF em cada instâcia do domínio do Lanche Infantil Ágil

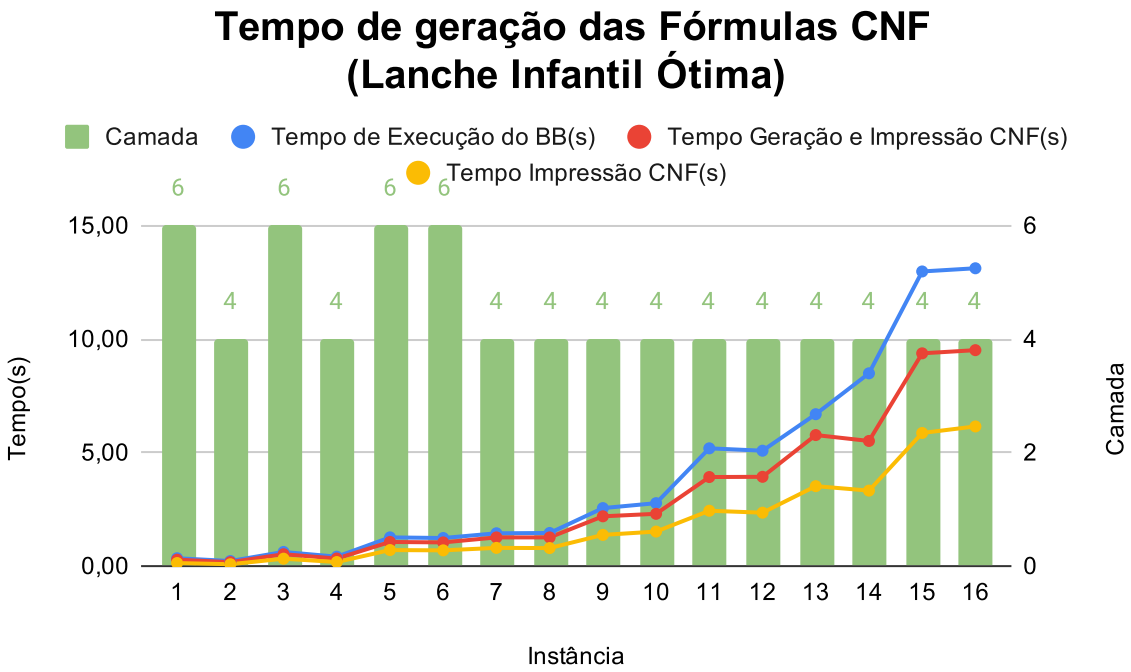

Figura 16 – Tempo de impressão da fórmula CNF em cada instâcia do domínio do Lanche Infantil Ótima

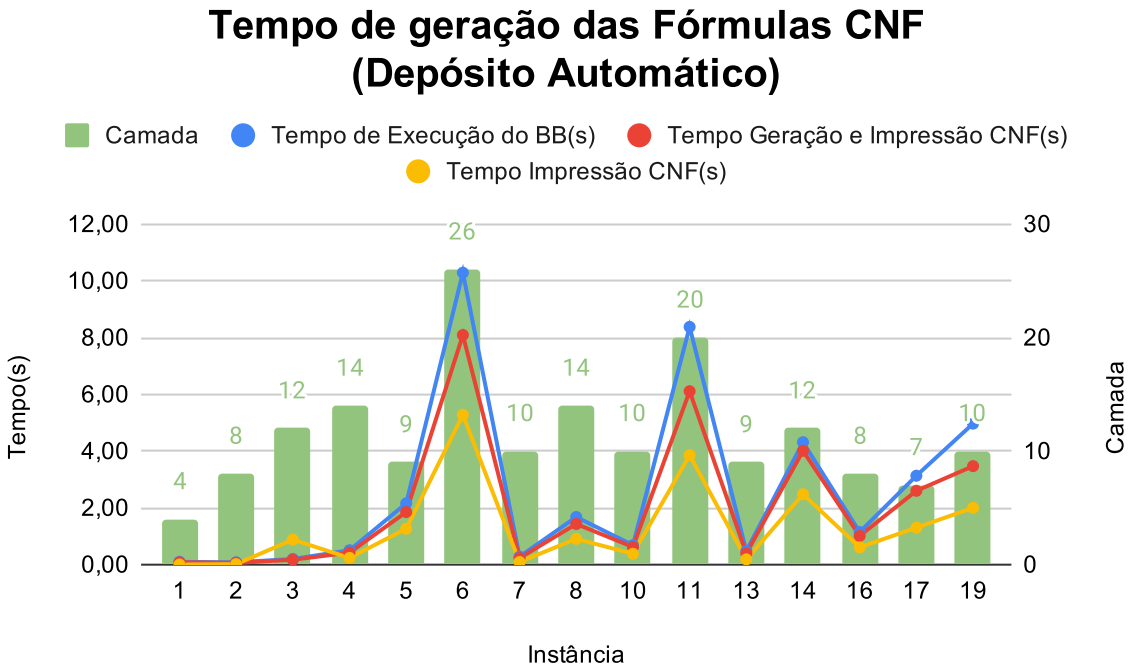

Figura 17 – Tempo de impressão da fórmula CNF em cada instâcia do domínio do Depósito Automático

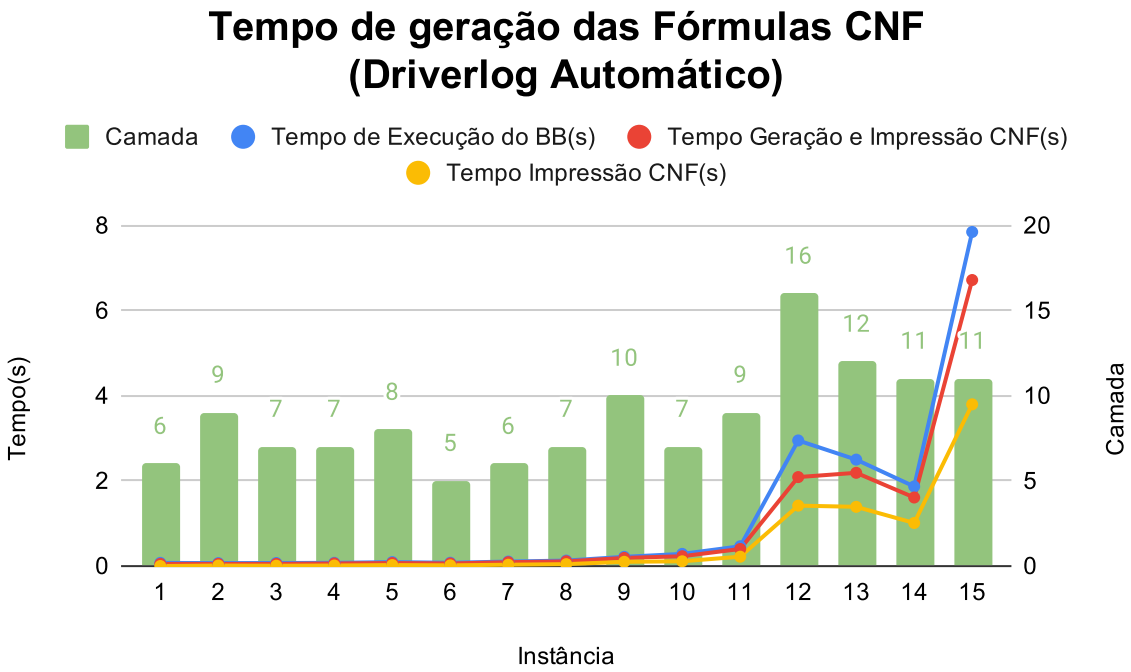

Figura 18 – Tempo de impressão da fórmula CNF em cada instâcia do domínio do Driverlog Automático

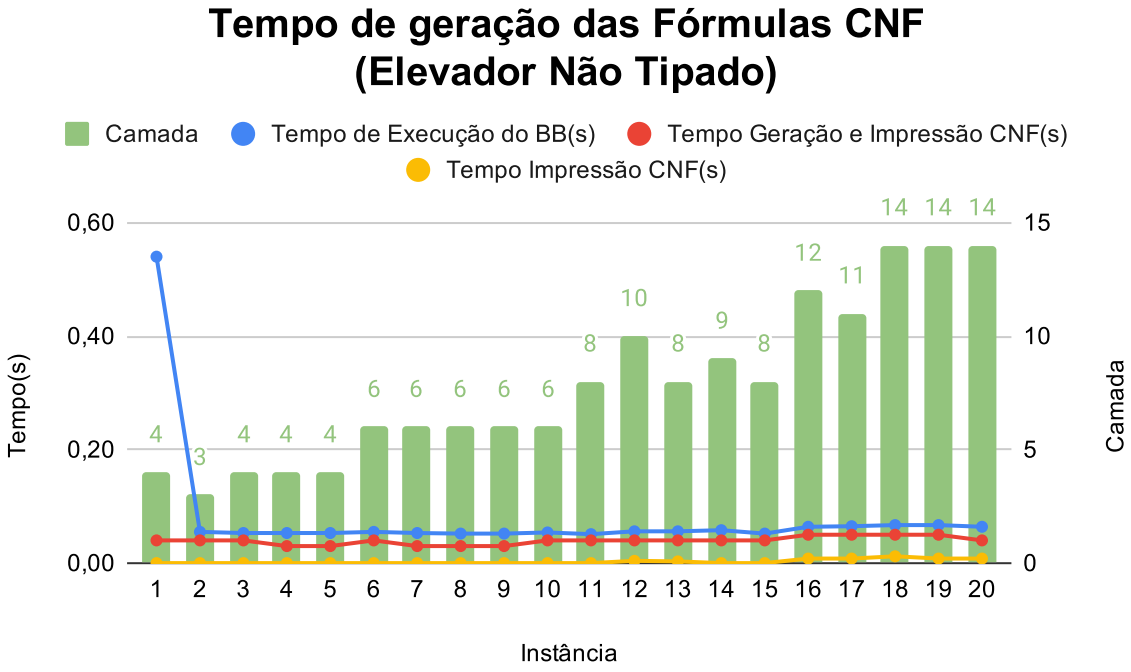

Figura 19 – Tempo de impressão da fórmula CNF em cada instâcia do domínio do Elevador Não Tipado

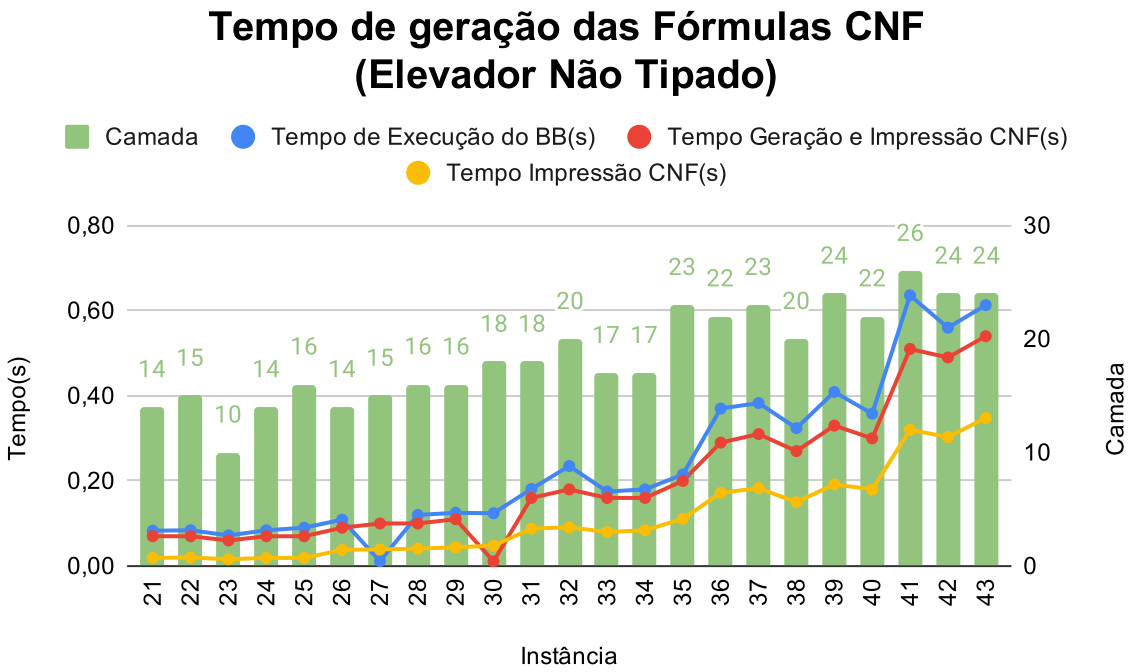

Figura 20 – Tempo de impressão da fórmula CNF em cada instâcia do domínio do Elevador Não Tipado

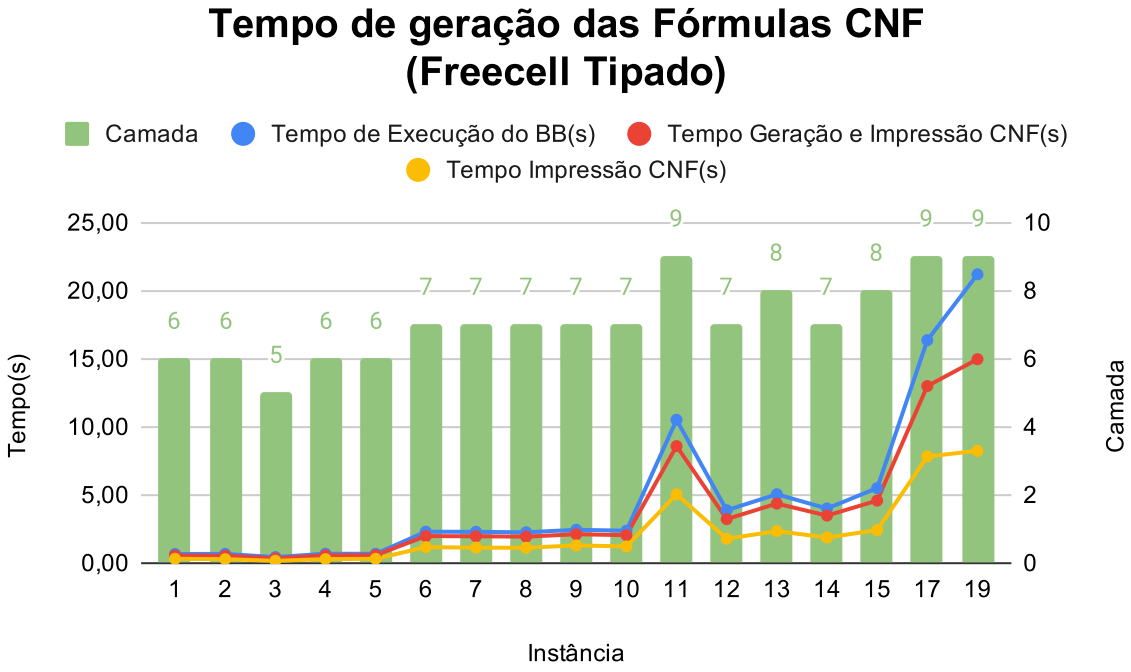

Figura 21 – Tempo de impressão da fórmula CNF em cada instâcia do domínio do Freecell Tipado

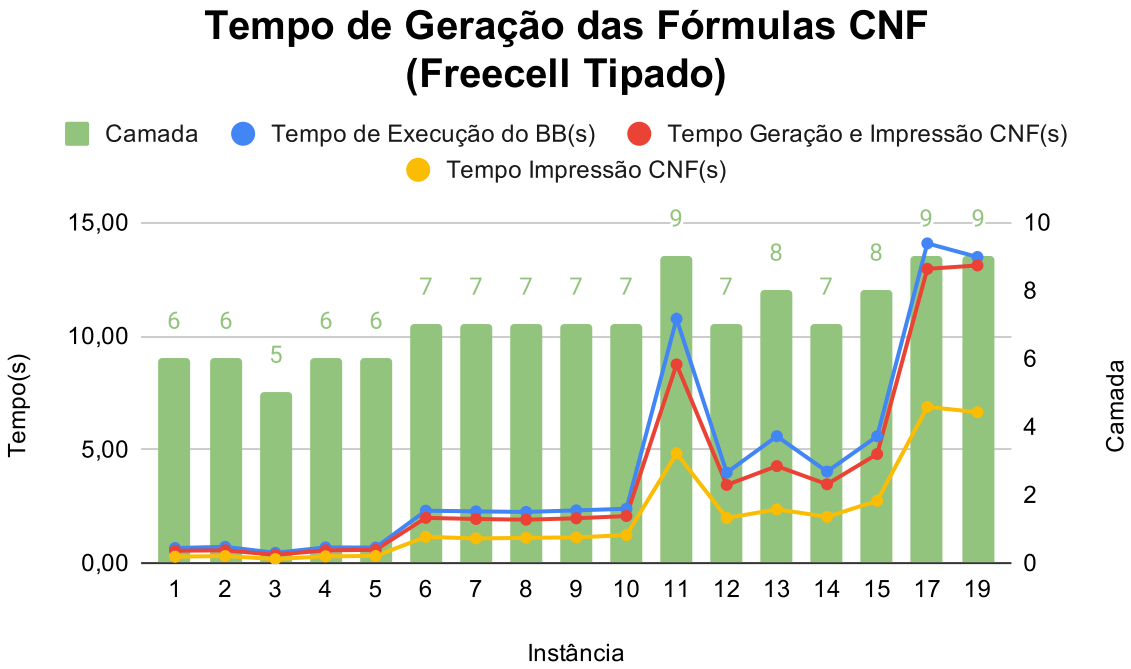

Figura 22 – Tempo de impressão da fórmula CNF em cada instância do domínio do Freecell Tipado

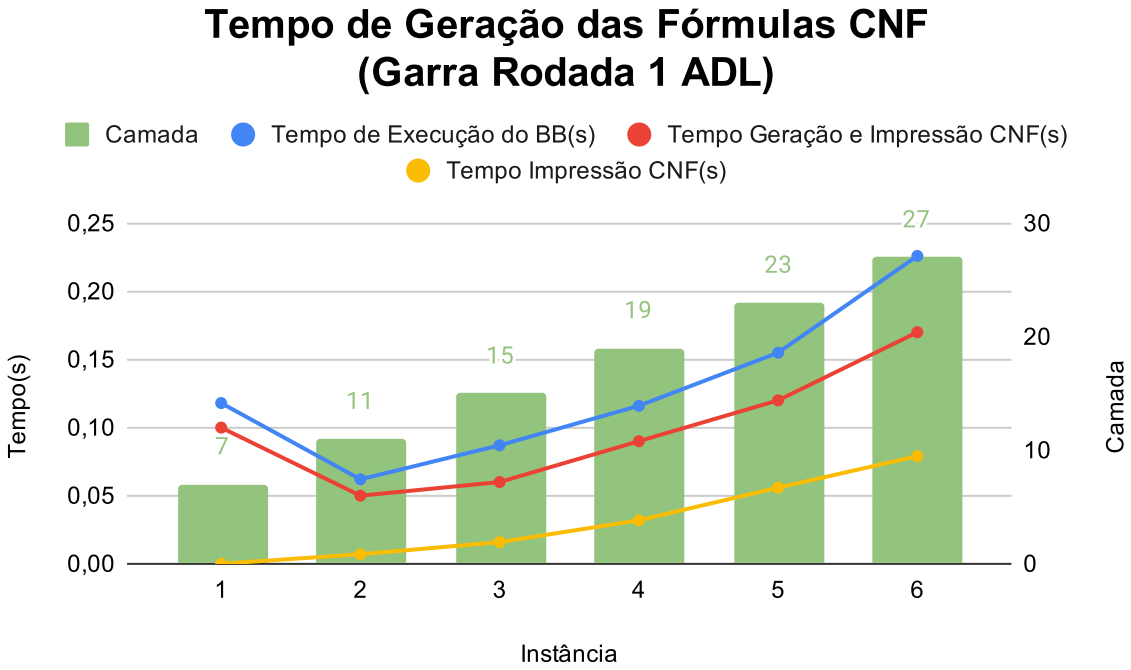

Figura 23 – Tempo de impressão da fórmula CNF em cada instância do domínio da Garra Rodada 1 ADL

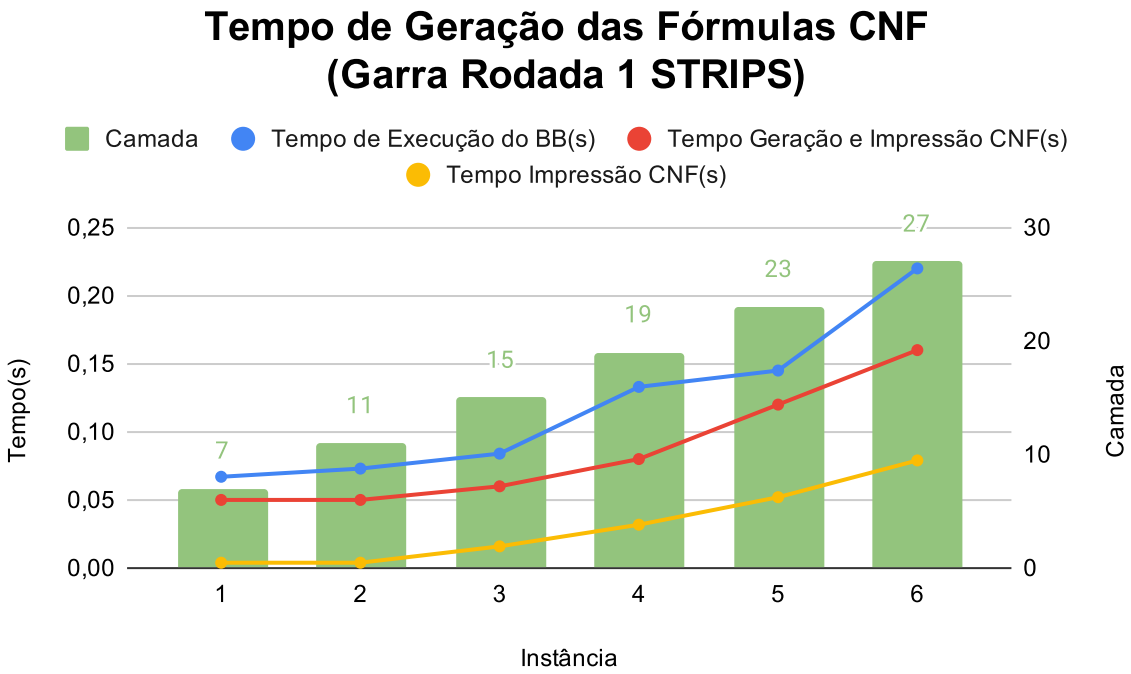

Figura 24 – Tempo de impressão da fórmula CNF em cada instâcia do domínio da Garra Rodada 1 STRIPS

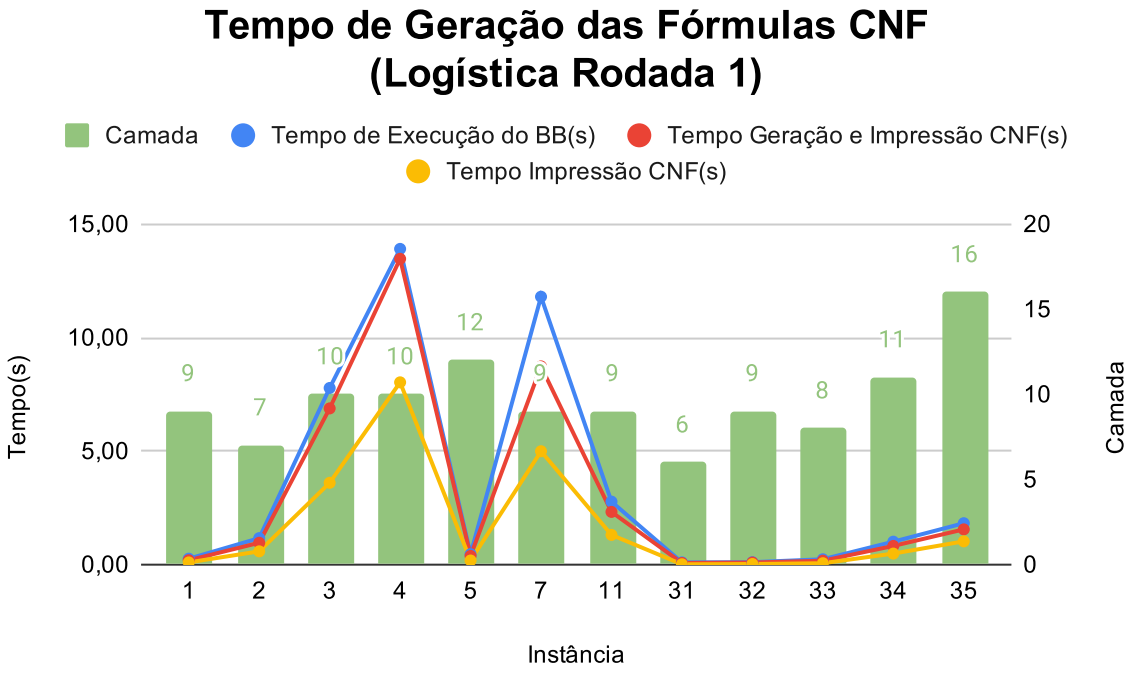

Figura 25 – Tempo de impressão da fórmula CNF em cada instâcia do domínio da Logística Rodada 1

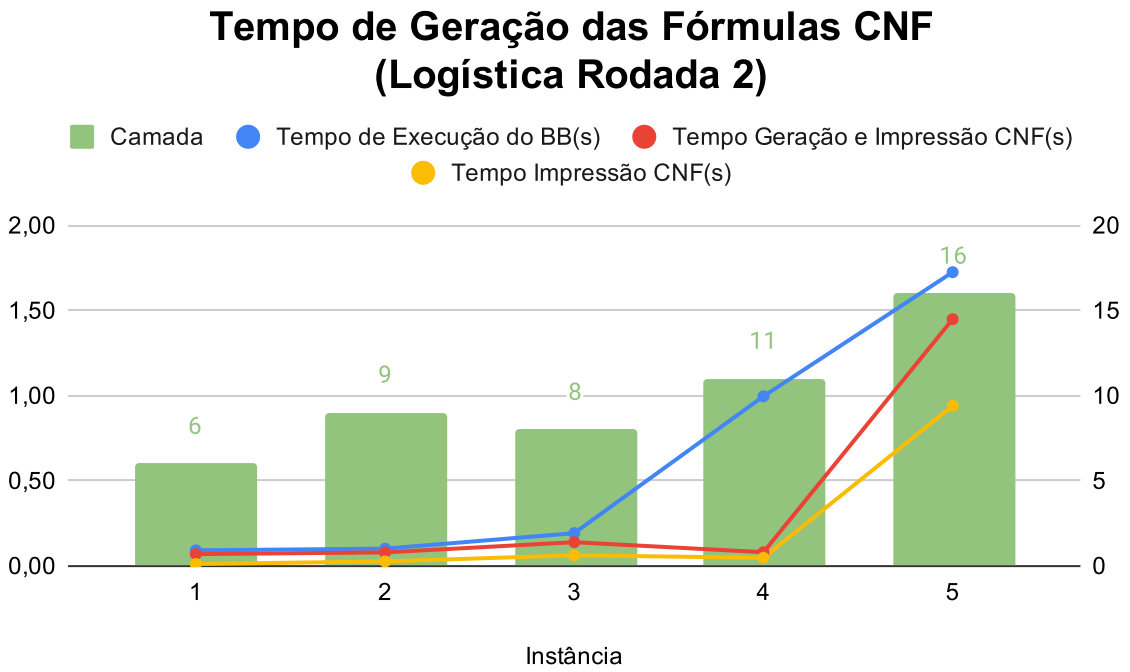

Figura 26 – Tempo de impressão da fórmula CNF em cada instâcia do domínio da Logística Rodada 2

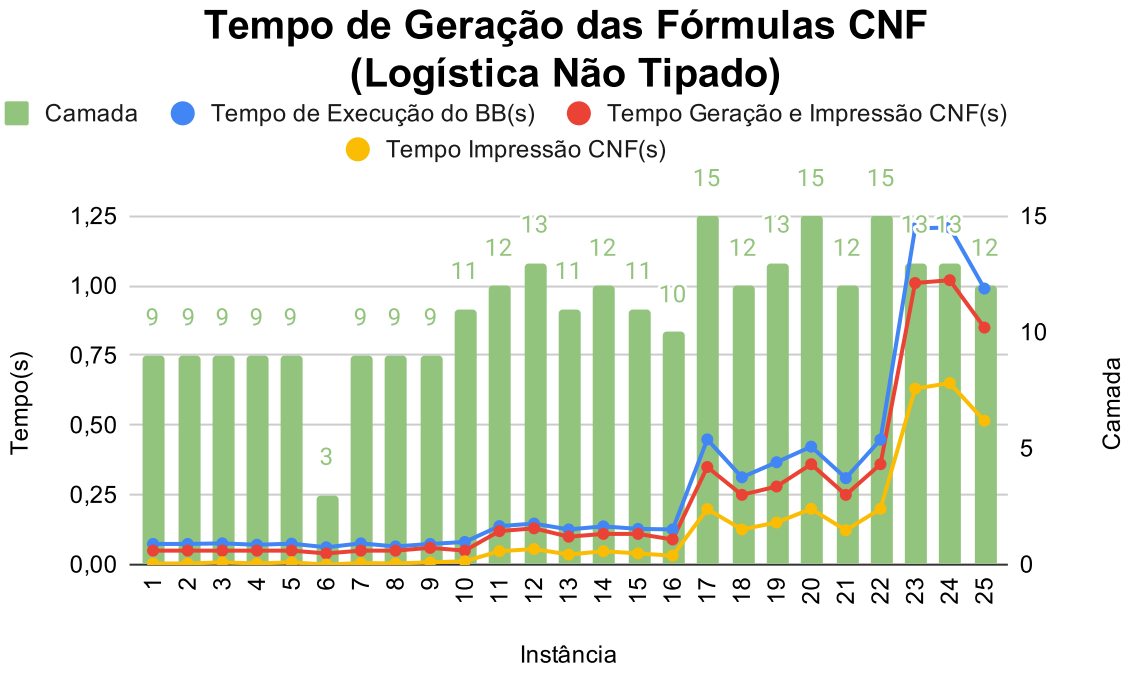

Figura 27 – Tempo de impressão da fórmula CNF em cada instâcia do domínio da Logísitica Não Tipado

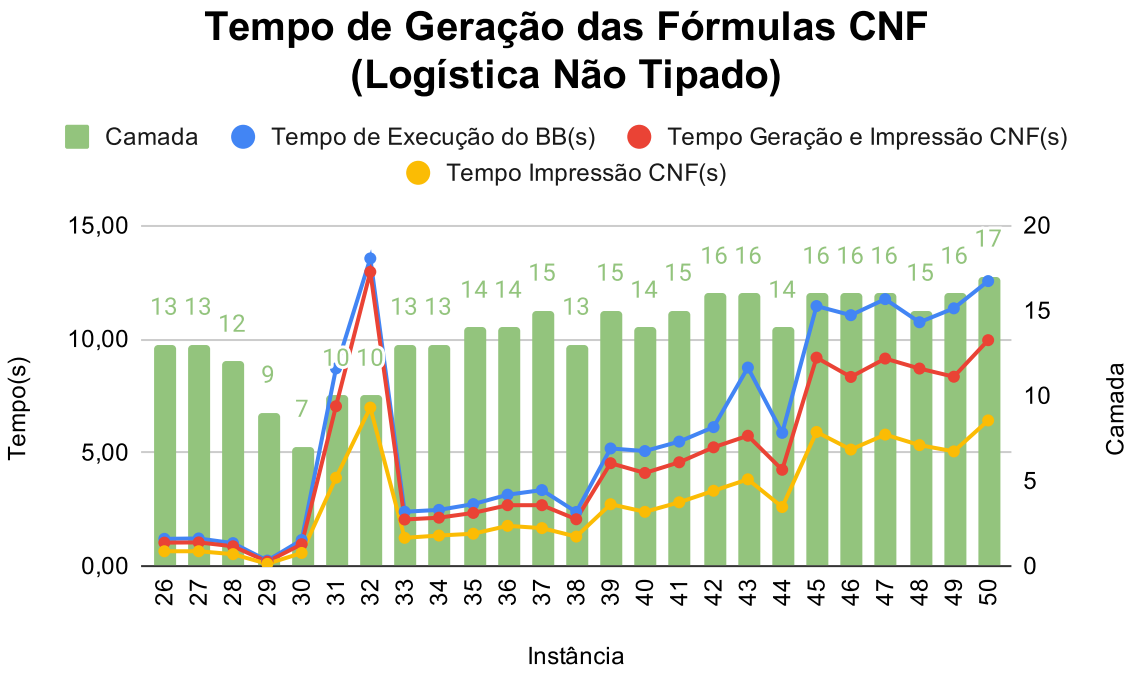

Figura 28 – Tempo de impressão da fórmula CNF em cada instâcia do domínio da Logística Não Tipado

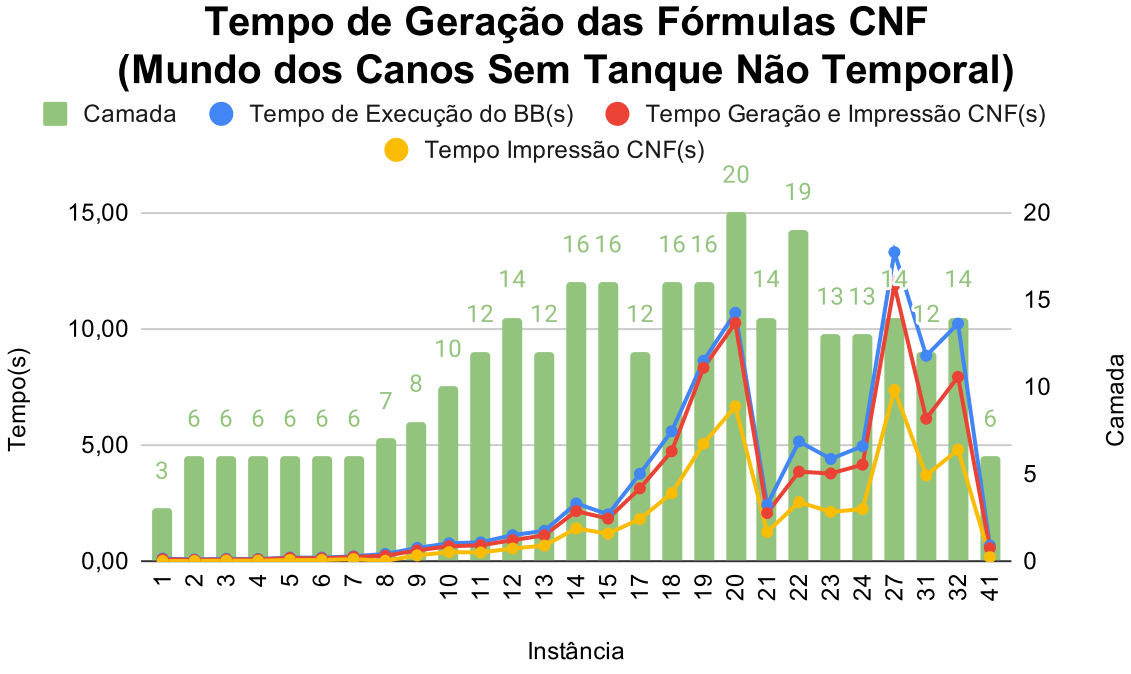

Figura 29 – Tempo de impressão da fórmula CNF em cada instâcia do domínio do Mundo dos Canos Sem Tanque Não Temporal

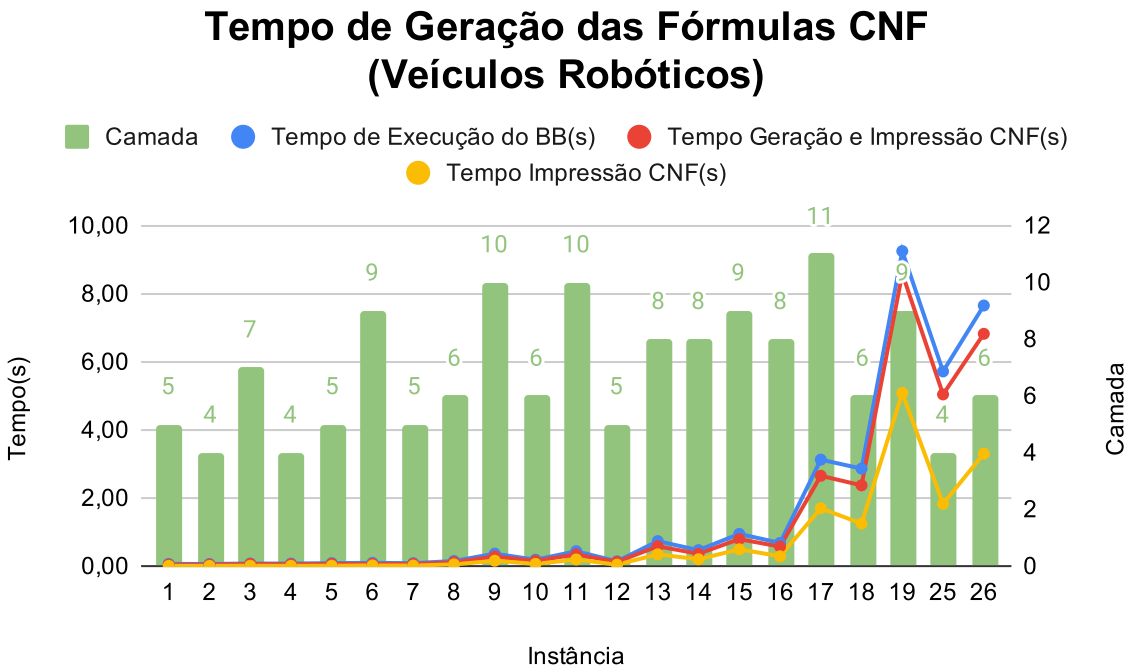

Figura 30 – Tempo de impressão da fórmula CNF em cada instância do domínio dos Veículos Robóticos