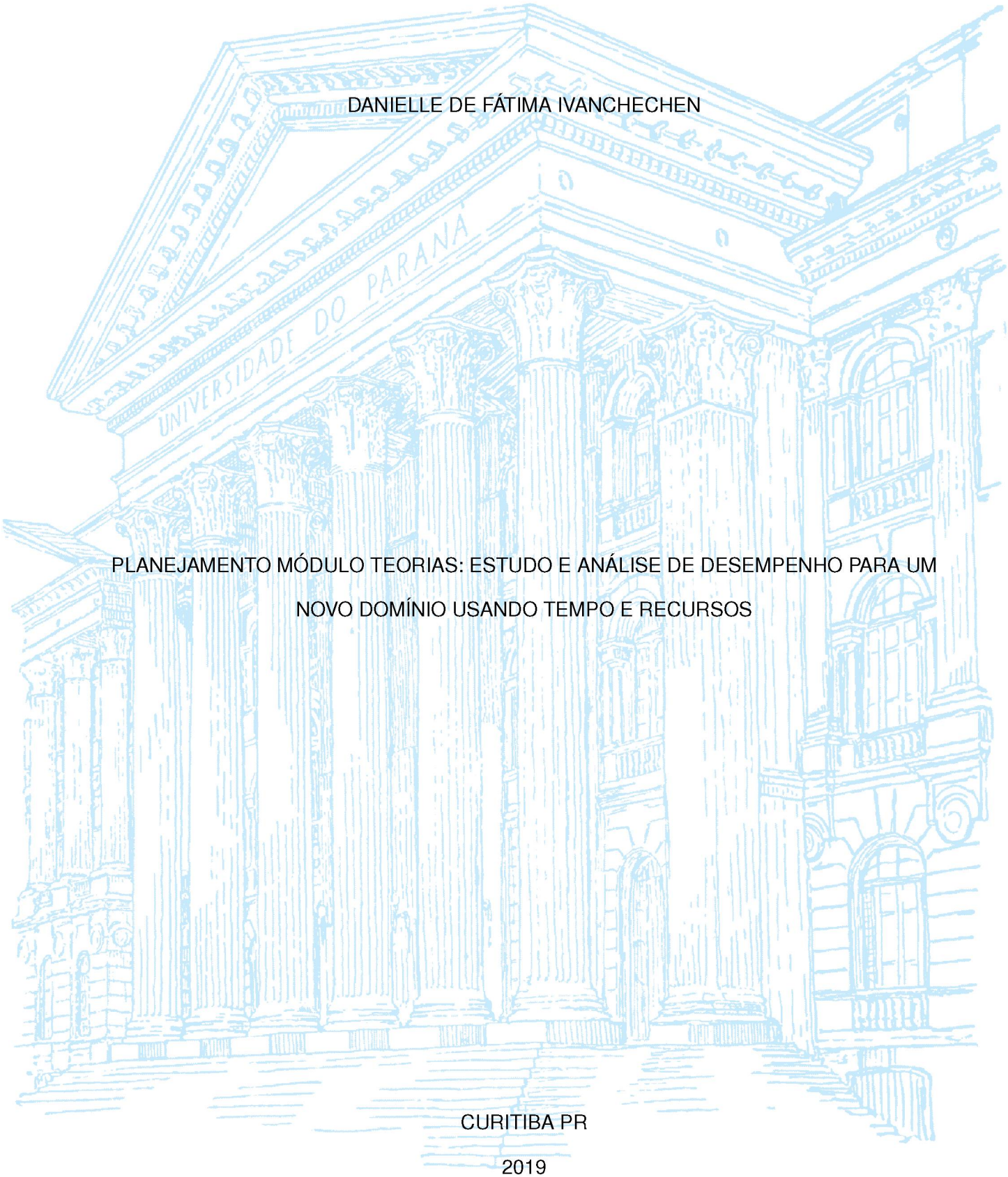
UNIVERSIDADE FEDERAL DO PARANÁ

DANIELLE DE FÁTIMA IVANCHECHEN



PLANEJAMENTO MÓDULO TEORIAS: ESTUDO E ANÁLISE DE DESEMPENHO PARA UM NOVO DOMÍNIO USANDO TEMPO E RECURSOS

CURITIBA PR

2019

DANIELLE DE FÁTIMA IVANCHECHEN

PLANEJAMENTO MÓDULO TEORIAS: ESTUDO E ANÁLISE DE DESEMPENHO PARA UM NOVO DOMÍNIO USANDO TEMPO E RECURSOS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática no Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Prof. Dr. Marcos Alexandre Castilho.

CURITIBA PR

2019

Catalogação na Fonte: Sistema de Bibliotecas, UFPR
Biblioteca de Ciência e Tecnologia

I93p Ivanchechen, Danielle de Fátima
Planejamento módulo teorias: estudo e análise de desempenho para um novo domínio usando tempo e recursos [recurso eletrônico] /Danielle de Fátima Ivanchechen. – Curitiba, 2019.

Dissertação – Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática, 2019.

Orientador: Marcos Alexandre Castilho.

1. Inteligência artificial. 2. Algorítmos. 3. Desempenho. I. Universidade Federal do Paraná. II. Castilho, Marcos Alexandre. III. Título.

CDD: 307.12

Bibliotecária: Vanusa Maciel CRB- 9/1928

MINISTÉRIO DA EDUCAÇÃO
SETOR DE CIENCIAS EXATAS
UNIVERSIDADE FEDERAL DO PARANÁ
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO INFORMÁTICA - 40001016034P5

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da dissertação de Mestrado de **DANIELLE DE FÁTIMA IVANCHECHEN** intitulada: **PLANEJAMENTO MÓDULO TEORIAS: ESTUDO E ANÁLISE DE DESEMPENHO PARA UM NOVO DOMÍNIO USANDO TEMPO E RECURSOS**, sob orientação do Prof. Dr. MARCOS ALEXANDRE CASTILHO, que após após terem inquirido a aluna e realizado a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.

A outorga do título de mestre está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

CURITIBA, 16 de Setembro de 2019.

MARCOS ALEXANDRE CASTILHO
Presidente da Banca Examinadora (UNIVERSIDADE FEDERAL DO PARANÁ)

GUSTAVO ALBERTO GIMENEZ LUGO
Avaliador Externo (UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ)

FABIANO SILVA
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

MINISTÉRIO DA EDUCAÇÃO - UNIVERSIDADE FEDERAL DO PARANÁ - Setor de Ciências Exatas - Coordenação de Pós-Graduação em Informática

AGRADECIMENTOS

Agradeço, primeiramente, a Deus, por sempre estar ao meu, mesmo nos momentos em que eu não percebia Sua presença. Agradeço pela força e coragem recebidas para terminar essa pesquisa e texto. Agradeço à todas as oportunidades que Ele me concedeu durante esse período de estudo. Sem Ele, nada seria possível.

Agradeço aos meus pais, Teresinha e Irineu, pois sempre me apoiaram e jamais me deixaram desistir. Também agradeço ao meu irmão, Daniel, pelos momentos de descontração e ajuda para formatar as tabelas.

Aos meus amigos, que sempre estiveram ao meu lado e me ajudaram muito nessa jornada, em especial, Cibelle e Rafael.

Agradeço, de maneira especial, ao Vinícius, pelo apoio e compreensão. Além da ajuda para montar algumas figuras do texto.

Agradeço, também, ao Professor Dr. Marcos Alexandre Castilho, pela orientação e paciência durante esses anos.

Finalmente, a todos os amigos e colegas que contribuíram de alguma forma para que este trabalho fosse concretizado.

Muito obrigada!

A Universidade Pública faz ciência.
A ciência produz conhecimento.
O conhecimento destrói os mitos.
#OrgulhoDeSerUFPR
#UniversidadePública #EuDefendo

RESUMO

À medida que a pesquisa de planejamento se torna mais relevante para aplicações no mundo real, aumentam as demandas de poder expressivo na linguagem de modelagem. Em particular, existe um novo formalismo de planejamento em que se utiliza noções de teorias para aumentar o poder da modelagem. Este trabalho mostra como o Planejamento Módulo Teorias (PMT) cumpre esse papel, analisando o seu desempenho em comparação com o planejador Metric-FF. Também, apresenta-se um novo domínio em que utiliza-se tempo e recursos simultaneamente, o qual mostra como o PMT pode resolver problemas que vão além dos domínios existentes.

Palavras-chave: SAT, SMT, Planejamento, Planejamento Módulo Teorias

ABSTRACT

As the search for plans becomes more relevant to applications in the real world, it increases the demands for expression in the modeling language. In particular, there is a new planning formalism for the use of new theories to increase the power of modeling. This work shows how the Planning Modules Theory (PMT) fulfills this role by analyzing its performance compared to the Metric-FF planner. Also, a new domain is presented in which time and resources are used simultaneously , which shows that the PMT is able to solve problems that are outside the of current techniques.

Keywords: SAT, SMT, Planning, Planning Modules Theory

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

IA	Inteligência Artificial
PMT	Planning Modulo Theories
SAT	Satisfazibilidade
SMT	Satisfiability Modulo Theories
PDDL	Planning Domain Definition Language
STRIPS	STanford Research Institute Problem Solver
ADL	Action Description Language
MDDL	Module Definition Description Language
CDDL	Core Domain Description Language

SUMÁRIO

1 INTRODUÇÃO

Em muitas situações do dia a dia o ser humano se depara com a necessidade de tomar decisões com a finalidade de alcançar uma meta. Esse processo faz com que seja necessário a criação de um plano de ações que o leve ao seu objetivo. Esse processo é chamado de planejamento e é possível automatizá-lo em Inteligência Artificial (IA).

Há muitos anos o estudo de planejamento tem sido um dos principais objetivos de pesquisa em IA. Russell e Norvig [RN03] citam que um dos principais motivos do interesse pela área é o fato dela combinar dois conceitos em IA: a busca e a lógica. Assim, um planejador, que é um sistema implementado para encontrar um plano de ações, pode ser visto tanto como um algoritmo que busca por uma solução quanto um algoritmo que de maneira construtiva prova a existência (ou não) de uma solução.

Até o início da década de 1970, os problemas de planejamento eram resolvidos por meio de prova de teoremas e lógica clássica. Achar o plano compreendia em provar um teorema e resgatar os passos para a obtenção da prova [Gre69]. Essa abordagem de representação apresenta um impedimento, que é a necessidade de tratar o problema de persistência [MH69].

A primeira proposta de solução integrando um sistema formal alternativo à lógica e um algoritmo correspondente foi apresentado por Fikes e Nilsson em 1971, chamado STRIPS (*STanford Research Institute Problem Solver*) [FN71].

A linguagem STRIPS é composta por um modelo formal para representar as ações e os estados do mundo e um algoritmo que trata o problema como uma busca no espaço de estados. A representação STRIPS é derivada do cálculo proposicional e usa uma representação de literais de primeira ordem, que devem ser básicos e livres de funções. Um problema de planejamento escrito em STRIPS apresenta duas partes: a descrição do domínio e o descrição do problema. Essa linguagem permite uma representação simples do problema, por isso, surgiu a necessidade de uma linguagem mais expressiva, merecendo um destaque, a linguagem ADL (*Action Description Language*) [Ped89].

A linguagem ADL apresenta melhorias em relação a linguagem STRIPS, como a inclusão de efeitos condicionais, a permissão de disjunção nos objetivos e assume a hipótese do mundo aberto, ou seja, assume a suposição de que o que não é conhecido como verdadeiro é desconhecido.

Com o objetivo de ter uma especificação padrão para representar os problemas de planejamento e permitir uma comparação justa, no caso de competições, foi desenvolvida em 1998 uma nova linguagem chamada PDDL (*Planning Domain Definition Language*) [MGH+98]. A linguagem PDDL é uma combinação das linguagens STRIPS e ADL, com tarefas hierárquicas, recursos, tempo.

Podemos formular o problema de planejamento de forma simples como descrito por Weld [Wel99]:

1. uma descrição do estado inicial do mundo em alguma linguagem formal;

2. uma descrição do objetivo do agente (isto é, qual comportamento é desejado) em alguma linguagem formal;

3. uma descrição das possíveis ações que podem ser realizadas (novamente, em alguma linguagem formal).

De acordo com a modelagem do problema, os planejadores dividem-se em duas categorias de planejamento: clássico e não clássico.

No planejamento clássico, o domínio do problema é totalmente conhecido, finito, estático, suas ações não tem duração, entre outros aspectos. Ao se modelar um problema de planejamento clássico, muitas vezes é preciso abandonar os detalhes, como recursos, tempo, mundo não observável, entre outros fatores. Isso faz-se necessário devido a complexidade de tempo de se encontrar uma solução. Muito já foi desenvolvido para resolver e melhorar a eficiência desses planejadores. Alguns exemplos são o planejamento por análise de grafos [BF97], por satisfazibilidade [KS99, KS$^+$92], rede de planos [Sil05], redes de Petri [SCK00] e busca heurística [BG98, HN01].

Ainda que o planejamento clássico resolva uma grande variedade de problemas ele ainda é limitado e não é realista. Para ir além do domínio proposicional e para que os estudos possam avançar no sentido de resolver problemas reais, embora saiba-se antecipadamente que os algoritmos propostos terão que lidar com fatores de tempo e espaço exponenciais, ocorre o planejamento não clássico. Esse planejamento nasce da necessidade de adicionar noções de tempo, quantidades, espaço, entre outros, a fim de possibilitar aos modelos o tratamento de outros problemas em que esses fatores são fundamentais. Assim, o planejamento não clássico pode ser classificado de acordo com o problema a ser tratado. Alguns exemplos são: o planejamento temporal [BK00]; hierárquico [TEHN96]; por checagem de modelos [CRT98]; em domínios não determinísticos; multiagentes, com recursos, etc.

Até o momento, cada problema de planejamento não clássico deve ter um algoritmo específico, ou seja, um algoritmo com sua própria sintaxe e integração para tratar o problema em questão. Dessa forma, um planejador que trata um problema de planejamento em que o tempo é um fator fundamental não tratará de forma correta um problema em que a quantidade de um recurso é indispensável. Encontrar uma maneira de tratar esses problemas diretamente pode ser interessante.

Em 2012, Gregory e colegas [GLFB12] propuseram um novo formalismo para trabalhar com planejamento não clássico, no qual permite-se que o planejamento seja ampliado em um caminho modular, de modo que o planejador trate de forma passível de interação os problemas de planejamento. Esse formalismo é chamado de Planejamento Módulo Teorias, em inglês, *Planning Modulo Theories* (PMT).

O planejamento clássico compartilha muito com o problema de satisfazibilidade (SAT), que também se preocupa com avaliações de variáveis proposicionais. Uma pesquisa que tem sido objeto de estudos recentes na comunidade SAT é a extensão da linguagem proposicional para linguagens mais expressivas, chamada Satisfazibilidade Módulo Teorias ou, em inglês, *Satisfiability Modulo Theories* (SMT). Dessa forma, o SMT pode ser visto como o campo de pesquisa envolvido com a verificação da satisfazibilidade de lógicas mais expressivas baseado em teorias lógicas de interesse. O problema SMT pode ser representado da seguinte maneira [GLFB12]:

- Instância: fórmula φ, incluindo símbolos constantes, predicados e de função, e uma teoria T, definindo os significados dos símbolos;
- Pergunta: φ é satisfazível, sujeito às interpretações dos símbolos impostos pela teoria T?

À medida que a pesquisa de planejamento se torna mais relevante para aplicações no mundo real, aumentam as demandas de poder expressivo na linguagem de modelagem. O SAT respondeu a esse problema ao permitir a extensão com teorias no SMT. O PMT trata o problema

de planejamento de forma análoga ao SMT, no qual permite a exploração de técnicas básicas do planejamento juntamente com a decomposição de problemas que precisam de técnicas mais especializadas. Consequentemente, esse formalismo trata de forma direta os problemas de planejamento, de modo que não seja necessário o desenvolvimento de um algoritmo específico para cada problema.

Grande parte dos problemas do mundo real impõem limitações de recursos, como limitações de energia, de dinheiro, de combustível, de materiais, entre outros e limitações de tempo. Dessa forma, o uso de um planejador para tratar esses problemas é fundamental para encontrar um plano que maximize ou minimize o valor total de um determinado recurso ou o tempo gasto.

A partir disto, o objetivo desta pesquisa é estudar e explicar esse novo formalismo de planejamento, o PMT. Apresentaremos um planejador que resolve diretamente problemas PMT, o PMTPlan, desenvolvido por Gregory e colegas [GLFB12]. Também explicaremos a nova linguagem apresentada para tratar esse problema. Com o uso do PMTPlan, demonstraremos a eficácia desse formalismo ao trabalhar com problemas não clássicos. Conjuntamente, analisaremos como o planejador comporta-se com problemas em que o uso de tempo e recursos são usados simultaneamente.

Dessa forma, este trabalho está organizado da seguinte maneira: o capítulo 2 traz o problema de planejamento, a representação do conhecimento dentro do planejamento clássico (este apresentado de maneira superficial, pois não é de interesse aprofundá-lo, já que existem muitas pesquisas nessa área) e dentro do planejamento não clássico, apresentando de forma mais aprofundada o planejamento temporal e com recursos. No capítulo 3 é apresentada uma introdução sobre Satisfazibilidade Booleana e conceitos sobre SMT (formalismo no qual o PMT é baseado). O capítulo 4 apresenta conceitos e definições sobre o PMT, assim como a descrição de problemas em PMT e alguns resultados que envolvem recursos, estes que foram mostrados em [**?**]. O capítulo 5 apresenta nossos resultados, baseados no tratamento de problemas que envolvem tempo e recurso. E, por último, o capítulo 6 traz nossas conclusões e trabalhos futuros a cerca do PMT.

2 PLANEJAMENTO NÃO CLÁSSICO

Planejamento, segundo o dicionário Michaelis [dic19], significa a determinação das ações para atingir metas estipuladas. De maneira informal, em Inteligência Artificial, o planejamento é uma área de estudo que visa encontrar de forma automatizada a melhor sequência de ações que seja capaz de alcançar o estado objetivo proposto partindo de um estado inicial. Esse é um problema de alta complexidade computacional e pertence a classe PSPACE-Completo [Byl94]. Mesmo assim, os algoritmos atuais resolvem muitas instâncias de planejamento.

Neste capítulo apresenta-se o problema de planejamento na seção 2.1. As seções 2.2 e 2.3 trazem a representação do conhecimento dentro dos planejamentos clássico e não clássico, respectivamente, na qual esta última foca nas restrições de recursos e de tempo.

2.1 O PROBLEMA DE PLANEJAMENTO

O ser humano no seu dia a dia realiza o ato de planejar quando deseja alcançar algum objetivo. Deste modo, define uma sequência de ações que seja capaz de partir de uma situação atual e chegar a um objetivo. O problema de planejamento em IA consiste em automatizar esse processo do ser humano. Pode-se modelar e resolver diversos problemas do mundo real como, por exemplo, o problema de transporte logístico, transporte aéreo e satélites, planejamento para resposta a incêndio, resgate, jogos de estratégia, segurança/controle de redes.

Para solucionar um problema do mundo real de forma automática é necessário ignorar os eventos externos que não geram impacto, de modo a simplificar a sua descrição. Isso porque os problemas tendem a ser complexos e não apresentam uma única descrição, o que torna a sua análise difícil. Assim a descrição concentra-se apenas nos eventos que causam algum impacto no problema.

O problema de planejamento é baseado na ideia da hipótese de mundo fechado, ou seja, as variáveis que não são afetadas explicitamente por uma ação são assumidas inalteradas.

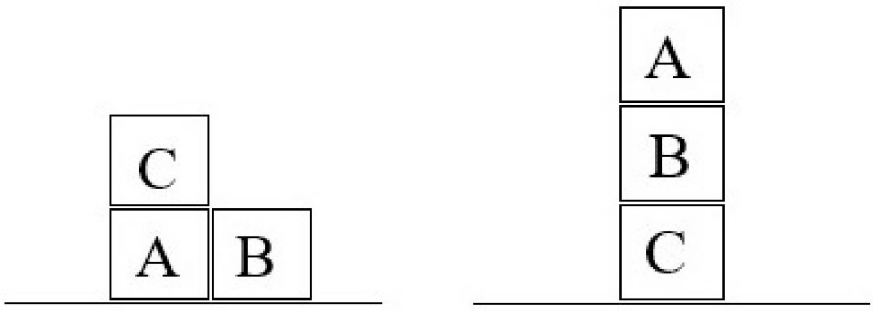
Um exemplo clássico é o problema do mundo dos blocos, descrito como um mundo no qual existe uma certa quantidade de blocos empilhados de maneira qualquer sobre uma mesa. O desafio consiste em encontrar uma sequência de ações, definidas no escopo do problema, que levem os blocos empilhados de uma maneira qualquer a ficarem empilhados de um modo desejado.

Vamos assumir como exemplo uma instância desse problema.

Sejam três blocos, A, B, e C, sobre uma mesa inicialmente dispostos como mostra a Figura 2.1(a). O objetivo é rearranjá-los conforme ilustra a Figura 2.1(b).

Assim, um problema de planejamento possui os estados inicial e objetivo e as ações disponíveis. Um estado representa uma situação do mundo do problema e a mudança de um estado para o outro é realizada pela aplicação de uma ação. Uma ação é composta de pré-condições, que especifica como o estado deve estar antes de aplicá-la (o bloco C deve estar livre para que se possa empilhá-lo sobre a MESA), e de pós-condições, no qual descreve o estado após a aplicação da ação (após empilhar o bloco C sobre a MESA o bloco A fica livre). A sequência de ações (ou movimentos), como empilha C sobre a mesa, empilha B sobre C e empilha A sobre B, ou qualquer outra sequência que gera uma solução para o problema, é chamada plano.

No conjunto de ações disponíveis para o planejamento, pode existir subconjuntos de ações que podem ser realizadas ao mesmo tempo. Então, um plano pode ser classificado como



(a) Estado inicial (b) Estado Objetivo

Figura 2.1: Mundo dos Blocos

parcialmente ordenado, no qual contém ações que podem ser realizadas ao mesmo tempo, e ordenado, no qual não contém ações que podem ser executadas ao mesmo tempo.

De modo geral, um problema de planejamento possui as seguintes definições:

- Literais: Define-se V como um conjunto de variáveis de estado positivas. O conjunto de literais sobre o conjunto V é $L = V \cup \{\neg v | v \in V\}$;
- Predicado: Um predicado $R(v)$ é uma expressão que define uma propriedade R para um conjunto de literais v, tal que $R(v)$ é verdadeiro ou falso;
- Estado: Um estado é a descrição do mundo em dado instante de tempo, isto é, um conjunto de variáveis de estado verdadeiras naquele instante;
- Ação: Uma ação é um operador que transforma um estado do mundo em um outro. Essa transformação é dada pela condição $a = \langle Cond, Eff \rangle$, no qual *Cond* é o conjunto de predicados que compõem as condições e *Eff* é uma dupla $\langle Add \cup Del \rangle$, em que *Add* o conjunto de literais positivos e *Del* o conjunto literais negativos, que compõem os efeitos da ação;
- Plano: Um plano P é um conjunto parcialmente ordenado de ações que leva à solução de um dado problema, ou seja, $P = \{a_1, a_2, ..., a_n\}$, tal que a_i é uma ação.

Um sistema implementado para encontrar um plano de ações a ser executado é chamado de planejador e tem como entrada uma descrição dos estados inicial e objetivo e a descrição das ações disponíveis. Essas descrições são codificadas em linguagens formais, como a linguagem STRIPS [FN71], ADL[Ped89] ou PDDL [MGH+98].

Voltemos ao exemplo do mundo dos blocos. O planejador recebe como entrada a descrição dos estados inicial e objetivo e das ações, como apresentado abaixo:

```
estado inicial: A sobre a MESA
                B sobre a MESA
                C sobre A
estado objetivo: C sobre a MESA
                 B sobre C
                 A sobre B

ações:
    empilha: blocox de origem sobre blocoy
             pré-condições: blocoy estar livre; blocox estar livre
             pós condições: blocox sobre blocoy; origem ficar livre
```

A saída do planejador será o conjunto de ações como

```
empilha C sobre a MESA
empilha B sobre C
empilha A sobre B
```

De acordo com a descrição do domínio do problema, o planejamento se divide em duas categorias: planejamento clássico e não-clássico.

2.2 PLANEJAMENTO CLÁSSICO

O planejamento clássico se refere, de maneira geral, a um planejamento para problemas limitados e restritos. Essa classe de problemas também é conhecida na literatura como planejamento STRIPS [GNT04].

A abordagem clássica apresenta as seguintes restrições [GNT04]:

Postulado 2.2.1. ***Finito:*** *o ambiente deve possuir um conjunto finito de estados;*

Postulado 2.2.2. ***Totalmente conhecido:*** *o ambiente deve ser completamente observável, isto é, tem-se um conhecimento completo sobre o estado;*

Postulado 2.2.3. ***Determinístico:*** *se uma ação é aplicável a um estado, ela resulta em um único outro estado;*

Postulado 2.2.4. ***Estático:*** *não possui dinâmica interna, ou seja, deve ficar no mesmo estado até que uma ação seja aplicada;*

Postulado 2.2.5. ***Objetivos restritos:*** *os objetivos são especificados como um estado objetivo explícito ou como um conjunto de estados objetivos;*

Postulado 2.2.6. ***Planos sequenciais:*** *a solução para o problema deve ser uma sequência de ações finitas e linearmente ordenadas;*

Postulado 2.2.7. ***Tempo implícito:*** *as ações não tem duração, ou seja, são transições de estado instantâneas;*

Postulado 2.2.8. ***Planejamento offline:*** *o planejador não está preocupado com qualquer mudança externa que possa ocorrer no ambiente no momento em que executa o planejamento.*

Bylander mostra que o planejamento clássico é classificado como PSPACE-Completo [Byl94], no qual demonstra a grande dificuldade em lidar com esse tipo de problema que oferece grandes desafios apesar das suas restrições.

Estudos sobre planejamento clássico vêm desde os primeiros dias da IA, com desenvolvimentos que revolucionaram o campo a partir dos anos 90. Existem muitos algoritmos para a resolução desse problema como o algoritmo de planejamento de duas fases GRAPHPLAN [BL99] e seus descendentes IPP [KNHD97], STAN [LF99] e SGP [WAS98], algoritmos baseados em compilação para o SAT, como o planejador BLACKBOX [KS98], o planejador MEDIC [EMW97], implementado em Lisp, o SATPLAN [KS+92], entre outros.

Também existem vários métodos para resolver um problema de planejamento como: rede de planos [Sil05], redes de Petri [SCK00], busca heurística [BG98, HN01], estruturas convertidas para instâncias SAT [KS99, KS+92] e grafo de planos [BF97].

Dada a farta literatura, neste trabalho não entraremos diretamente nesse assunto. Interessados podem consultar, não exclusivamente, [GNT04, RN03, NS+72, Wel99, McD97, KMS96], além dos trabalhos já citados nesse texto.

2.3 PLANEJAMENTO NÃO CLÁSSICO

Apesar do planejamento clássico resolver uma grande variedade de problemas e ser bastante útil para estudar as lógicas e aspectos computacionais, ele ainda é limitado e não é realista. Adicionar noções temporais aos modelos, por exemplo, possibilita o tratamento de outros problemas em que esse fator é fundamental.

No momento em que algum dos postulados do 2.2.1 ao 2.2.8 é violado ocorre um Planejamento Não Clássico. De acordo com o postulado violado o planejamento não clássico pode ser dividido em vários tipos. Por exemplo, se o postulado 2.2.7 é violado temos o planejamento temporal [BK00], no qual as ações possuem duração e, por isso, os efeitos não são aplicados de forma instantânea no mundo. Da mesma forma, são exemplos de planejamento: hierárquico [TEHN96], por checagem de modelos [CRT98], em domínios não determinísticos, multiagente, com recursos, entre outros.

Dado o interesse desse trabalho, vamos concentrar-nos apenas no planejamento temporal e com recursos.

2.3.1 Planejamento Temporal e com Recursos

A maioria dos problemas do mundo real impõe restrições de tempo e de recursos como energia, dinheiro, combustível ou materiais. No planejamento temporal as ações possuem duração e seu o objetivo é a geração de um plano com a menor duração ou um plano que atenda uma determinada restrição de tempo. Assim, diferente do planejamento clássico, o número de ações do plano deixa de ser métrica, em que pode ocorrer a geração de um plano com uma quantidade maior de ações.

Já os recursos são utilizados para expressar quantidades numéricas de um determinado item. Por exemplo, uma empresa de entregas tem número limitado da capacidade de um caminhão e precisa entregar pacotes em vários pontos, então, é necessário uma variável que armazene a quantidade de pacotes dentro do caminhão. A abordagem de planejamento com recursos é comum em problemas de manufatura e de logística. Assim, as variáveis de estado armazenam valores numéricos ao invés de apenas valores booleanos, como acontece no planejamento clássico. Com o uso de recursos, o planejador pode tentar encontrar um plano que maximize ou minimize o valor total de um determinado recurso, no qual fornece uma maneira poderosa de podar o espaço de pesquisa e orientar o planejador em direção a um plano bem-sucedido.

As técnicas mais importantes para resolver problemas que envolvem métricas são baseadas em programação linear. Programas Mistos Integrais, em inglês *Mixed-Integer Programs* (MIPs), em que as variáveis podem ser restritas a valores inteiros, são resolvidas por uma combinação de busca e pelo uso de relaxamentos como programas lineares [CFLS08]. Van Den Briel e colaboradores em 2007 [VDBBKV07], Long e Fox em 2003 [LF03] e outros pesquisadores [SD05, KW00, WW01] exploraram a programação linear no planejamento.

Uma das melhores heurísticas para o problema de planejamento que usam métricas é o relaxamento prolongado de Hoffmann, implementado no planejador Metric-FF [Hof03], em que foi proposto uma maneira de planejar domínios com recursos usando busca no espaço de estados de encadeamento direto.

Um outro planejador, proposto por Wolfman, em 1999, é o LPSAT LCNF [WW99], um resolvedor de satisfazibilidade integrado com um algoritmo Simplex. Dessa forma, o LPSAT [ZKC01] explora o espaço de busca proposicional, atualizando o conjunto de requisitos métricos gerenciados pelo resolvedor de programa linear, enquanto o Simplex notifica o LPSAT se esses requisitos se tornarem insatisfatórios.

Muitos trabalhos e várias técnicas já foram abordados para outros problemas de planejamento não clássico, como planejamento temporal. Porém, o problema de planejamento de recursos só começou a receber atenção a partir dos anos 90, nos trabalhos de Frederking e Muscettola [FM92], Drabble e Tate [DT94] e Labone e Ghallab [LG95].

O estudo do problema de planejamento com recursos ainda é um desafio na IA, apesar dos recursos serem de grande importância para a resolução dos problemas poucas heurísticas foram propostas e os planejadores não são muito eficazes na solução deles.

Labone e Ghallab em [LG95] definem um recurso como qualquer substância ou conjunto de objetos cujo custo ou disponibilidade provoque restrições sobre as ações que os utilizam. Desta forma, os recursos podem ser classificados da seguinte forma, como mostrado em [Lab03]:

- **Recurso de Reservatório:** é um recurso de múltiplas capacidades que pode ser consumido e/ou produzido pelas atividades. Um recurso de reservatório tem uma capacidade máxima inteira e pode ter um nível inicial, por exemplo, um tanque de combustível.

- **Recurso Discreto:** é um tipo especial de recurso de reservatório. Esse tipo de recurso é usado durante algum intervalo de tempo (uma quantidade de recurso é consumida no tempo de início da atividade e a mesma quantidade é liberada no seu tempo final), por exemplo, um grupo de trabalhadores cuja disponibilidade pode mudar com o tempo.

- **Recurso Unário:** é um recurso discreto com capacidade de unidade, ou seja, todas as atividades que requerem o mesmo recurso unário precisam ser totalmente ordenadas, um exemplo é o caso de uma máquina que pode processar apenas uma operação por vez. Recursos unários são os recursos mais simples e mais estudados em IA.

Para a representação de recursos, cada ação tem um conjunto de restrições de recursos exigidos pela ação e cada restrição especifica um tipo de recurso como, por exemplo, os pacotes, a quantidade necessária e se é consumível (por exemplo, não existe mais pacotes para entrega) ou reutilizável (um caminhão está com sua capacidade total ocupada, mas ficará disponível quando efetuar uma entrega).

2.4 CONCLUSÃO

Neste capítulo introduzimos o conceito de problema de planejamento. De forma breve, apresentamos o planejamento clássico, que resolve problemas restritos e limitados, no qual foi fornecido algumas referências para possível aprofundamento no assunto. Por meio das restrições apresentadas chegamos ao planejamento não clássico, que ocorre no momento em que uma das restrições é violada. Com isso cada problema envolve o uso de planejadores mais específicos, o que justifica o nosso estudo em PMT, que será apresentado no capítulo 4.

3 SATISFAZIBILIDADE MÓDULO TEORIAS

Satisfazibilidade (SAT) é um problema que consiste em decidir se existe uma valoração que torna uma fórmula em lógica proposicional verdadeira. Como o problema Satisfazibilidade Módulo Teorias (SMT) está ligado ao problema SAT.

Neste capítulo apresenta-se uma introdução sobre Satisfazibilidade Booleana na seção 3.1. O restante do capítulo concentra-se na explicação e apresentação de conceitos sobre SMT.

3.1 SATISFAZIBILIDADE BOOLEANA

Satisfazibilidade (SAT) é o problema de decidir se uma fórmula em lógica proposicional é satisfazível, ou seja, é o problema de se encontrar uma valoração que torne a fórmula verdadeira.

Em lógica proposicional as fórmulas são baseadas em proposições, que podem ver-dadeiras ou falsas (nunca ambas). Por exemplo, a afirmação "O homem é mortal" e "A neve é branca" são proposições. A partir das proposições podemos construir sentenças que são formadas pela combinação de proposições usando conectivos lógicos, $\wedge$ (e), $\vee$ (ou), $\neg$ (negação), $\rightarrow$ (se ... então) e $\leftrightarrow$ (se e somente se).

Uma variável proposicional é denotada por letras latinas minúsculas $p, q, r, s, ...$ ou $x_1, x_2, ...$ e indicam as proposições (átomos ou fórmulas atômicas), por exemplo, a proposição "O homem é mortal" pode ser representada pela variável proposicional p. Uma variável proposicional pode receber os valores verdadeiro ou falso.

Uma fórmula proposicional pode ser definida recursivamente assim:

- Uma variável proposicional, chamada átomo, é uma fórmula;
- Se α é uma fórmula, $\neg\alpha$ também é uma fórmula;
- Sendo α e β fórmulas, então $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$, $(\alpha \rightarrow \beta)$ e $(\alpha \leftrightarrow \beta)$ também serão fórmulas;
- Todas as fórmulas são geradas pela aplicação das regras anteriores.

Uma fórmula possui literais. Um literal l é uma variável p ou a sua negação $\neg p$. Um literal puro é um literal que aparece apenas em uma forma em toda a fórmula. Uma cláusula c é uma disjunção de literais. Uma cláusula unitária é uma cláusula que contém apenas um literal.Uma cláusula é dita satisfeita se, pelo menos, um de seus literais assumir o valor verdadeiro e não satisfeita se todos os seus literais assumirem o valor falso e uma fórmula é dita satisfeita se todas as cláusulas forem verdadeiras.

Fórmulas da forma $(\alpha \wedge \beta)$ e $(\alpha \vee \beta)$ são denominadas, respectivamente, conjunção e disjunção. Uma conjunção é verdadeira quando todos os literais forem verdadeiros e falsa quando pelo menos um de seus literais for falso. Uma disjunção é verdadeira quando pelo menos um de seus literais é verdadeiro e falsa quando todos os literais forem falsos.

Qualquer fórmula pode ser convertida para a Forma Normal Conjuntiva (CNF), do inglês, *Conjuctive Normal Form*, classe de fórmulas da lógica proposicional representada por uma conjunção de disjunções e que utiliza apenas os operadores lógicos de: disjunção ($\vee$), conjunção ($\wedge$) e negação ($\neg$).Então, uma fórmula CNF é uma conjunção de cláusulas.

A fórmula

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

é um exemplo de uma fórmula que está na Forma Normal Conjuntiva.

A Forma Normal Conjuntiva facilita o processo de implementação dos resolvedores SAT, por isso, a maioria dos resolvedores precisa que a fórmula de entrada, ou instância do problema, esteja nessa formato.

A conversão para CNF envolve a aplicação de equivalências lógicas como as leis de De Morgan, lei da dupla negação e a propriedade distributiva. Com essas aplicações obtém-se uma nova fórmula equivalente à anterior. O uso dessas equivalências pode gerar fórmulas com tamanho exponencial de cláusulas em relação ao número de variáveis [Rib11]. Para contornar esse problema, usa-se um método chamado SAT-equivalência que consiste em adicionar novas variáveis à fórmula para provocar o seu aumento de forma linear em relação ao número de variáveis.

A verificação da satisfazibilidade é muito importante em várias áreas da Ciência da Computação, no qual se busca a solução de vários problemas práticos da área, como o planejamento automático, verificação de software e teste de circuitos [MS08].

Por exemplo, a verificação de equivalência de circuitos é um dos problemas que pode ser representado na forma normal conjuntiva e solucionado pelos resolvedores SAT. Outro problema no campo dos circuitos é a detecção de falhas [MS08].

Outra aplicação que pode ser solucionada com SAT é o problema de planejamento em Inteligência Artificial [KS$^+$92], que consiste em decidir se um estado objetivo pode ser alcançado a partir de um estado inicial. Resumidamente, a ideia de usar SAT em planejamento é converter a instância do problema em uma instância SAT e resolvê-lo com um método de verificação de satisfazibilidade.

SAT foi o primeiro problema a ser provado NP-Completo [Coo71], ou seja, ainda não é conhecido um algoritmo que o resolva em tempo polinomial. Mesmo assim, muitos trabalhos conseguiram resultados relevantes desde os anos 90 [MSS99, MMZ$^+$01, ES03].

A maioria dos trabalhos sobre SAT se baseiam no algoritmo *Davis-Putnam-Logemann-Loveland* (DPLL) [DP60, DLL62]. O algoritmo DPLL [DLL62], foi proposto por Martin Davis, George Logemann e Donald W. Loveland, em 1962 e é um aprimoramento do algoritmo apresentado em 1960, por Martin Davis e Hilary Putnam [DP60].

Seja uma fórmula φ e seu conjunto de cláusulas ψ, o algoritmo DPLL é definido com as seguintes regras [Rib11]:

- Tautologia: Remove de ψ todas as cláusulas que são tautologias. O conjunto de cláusulas resultante é insatisfazível apenas se ψ também for;
- Cláusula Unitária: Se existe alguma cláusula unitária em φ dada pelo literal l, remova de ψ todas as cláusulas que contenham l. Se ψ for vazia, então φ é satisfazível;
- Literal Puro: Se um literal l em ψ é puro, ou seja, se não existe em ψ a sua negação $\neg l$, remova de ψ todas as cláusulas que contenham l;

- Ramificação: Se nenhuma das regras anteriores for aplicável, ψ é reescrita da seguinte forma:

  $(A_1 \vee l) \wedge \cdots \wedge (A_m \vee l) \wedge (B_1 \vee \neg l) \wedge \cdots \wedge (B_n \vee \neg l) \wedge R$, onde:

  l é um literal;

  A_i, B_i e R são livres de l e $\neg l$, assim temos os conjuntos $\psi_1 = A_1 \wedge \cdots \wedge A_m \wedge R$ e $\psi_2 = B_1 \wedge \cdots \wedge B_n \wedge R$;

  ψ é insatisfazível se, e somente se, ψ_1 e ψ_2 forem insatisfazíveis.

O ponto crucial desse algoritmo está na regra de ramificação, que é o momento em que é selecionado uma variável da fórmula e é escolhido um valor verdade para ela (verdadeiro ou falso). Com essa valoração, o algoritmo propaga esse valor para os dois conjuntos de cláusulas, sendo para um conjunto definida como verdadeira e, para o outro, falsa.

A codificação do algoritmo DPLL, para a regra da ramificação, não é feita seguindo a definição, pois para cada escolha de variável o conjunto de cláusulas dobra de tamanho, portanto, as implementações são baseadas nos conceitos de nível e retrocesso.

Uma codificação genérica do DPLL, apresentado em [Rib11] pode ser visto abaixo:

```
1
2  //Procedimento DPLL
3  //Entrada: Fórmula em CNF
4  //Saída: Informação SAT ou INSATISFAZÍVEL
5  enquanto 1==1 faça
6     se Decide() == OK então
7        BCP()
8        se BCPRetornouConflito() == SIM então
9           se Diagnostico() == NaoPodeSerResolvido então
10             retorna INSATISFAZÍVEL
11          fim
12          senão
13             RETROCESSO()
14          fim
15       fim
16    fim
17    senão
18       retorna SAT
19    fim
20 fim
```

O algoritmo faz uma busca pelo espaço das possíveis valorações das variáveis. Com a função `Decide()`, um valor verdade é escolhido para uma variável ainda não valorada em cada passo da busca. A função `BCP()`, *Boolean Constraint Propagation*, faz uma propagação desse valor verdade para o restante da fórmula. Se o BCP retornou conflito, ou seja, alguma cláusula, com a propagação do valor, se tornou falsa, a função `Diagnostico` faz a análise desse conflito e identifica em qual nível de decisão um erro foi cometido e, então, faz um retrocesso, desfazendo todas as valorações do nível atual até o nível onde o conflito foi encontrado.

A partir do algoritmo DPLL, muitas estratégias de otimização foram apresentadas. O primeiro resolvedor a contribuir com a otimização do DPLL foi o SATO [Zha97], que otimizou o processo de avaliar as consequências de uma valoração na fórmula. A otimização foi obtida a partir do modo em que as cláusulas foram representadas. Essa representação envolve o uso de um vetor, onde dois marcadores são colocados no início e final do vetor, e são deslocados uma posição cada vez que o literal indicado pelo marcador for valorado como falso. Essa

representação apresenta uma melhoria no desempenho já que não é mais necessário avaliar todas as cláusulas a cada valoração.

Depois do SATO, outro resolvedor que contribuiu para a melhoria do DPLL foi o GRASP [MSS99], que introduziu o conceito de retrocesso não cronológico ao algoritmo. Esse conceito fez com que a revisão da valoração que gerou o conflito em uma cláusula fosse analisada de forma mais inteligente, ou seja, ao invés de voltar apenas um nível de decisão e trocar o valor, o algoritmo analisa o motivo da cláusula se tornar falsa e volta todos os níveis até a causa do conflito.

Embora a proposta do GRASP fosse boa, o uso do BCP clássico fez com que outros resolvedores da época tivessem resultados melhores. A partir desse problema, surgiu um novo algoritmo, chamado ZCHAFF. A maior contribuição desse algoritmo foi a otimização do BCP, onde as cláusulas passaram a ser analisadas somente quando entram em um estado crítico de valoração, ou seja, quando todos os literais da cláusula estão valorados como falso e apenas um deles ainda não foi valorado.

Apesar da grande aplicabilidade e avanços dos resolvedores SAT, a lógica proposicional não é expressiva o suficiente para representar muitos outros problemas do mundo real, sendo assim importante a utilização de fórmulas mais expressivas. Uma das abordagens de resolução de fórmulas mais expressivas é chamada de Satisfazibilidade Módulo Teorias ou, em inglês, *Satisfiability Modulo Theories* (SMT).

3.2 SATISFAZIBILIDADE MÓDULO TEORIAS

Satisfazibilidade Módulo Teorias ou, em inglês, *Satisfiability Modulo Theories* (SMT) é o campo de pesquisa envolvido com a verificação da satisfazibilidade de lógicas mais expressivas, como lógica de primeira ordem, livres de quantificadores, baseada em teorias lógicas de interesse [GBT09, BSST09], tais como: igualdade e funções não interpretadas, aritmética, vetores de bits, *arrays* e listas [Seb07]. Essas teorias serão apresentadas na subseção 3.2.3.

As primeiras referências ao SMT podem ser encontradas por meados dos anos 70 e início dos anos 80, nos trabalhos de Nelson e Oppen, quando determinaram a satisfação de uma conjunção de literais com base no algoritmo de fecho de congruência [NO80] e apresentaram um simplificador para uso na manipulação e verificação de programas [NO79]; Shostak, também apresentou um algoritmo para o raciocínio sobre igualdade [Sho78], um procedimento para uma extensão da aritmética Presburger [Sho79] e um método para obter fórmulas decodificadas com combinação de teorias [Sho84]; Boyer e Moore, estudaram a integração de um procedimento aritmético linear em um provador de teorema heurístico [BM85]. Nesses trabalhos, procedimentos de métodos formais foram usados.

Os resolvedores SMT[1] tiveram um grande desenvolvimento na década de 90 utilizando tecnologia SAT. Esses avanços podem ser vistos nos trabalhos de Armando e colaboradores [ACG99], que estudaram o problema de consistência para um conjunto de restrições temporais disjuntivas, propondo dois procedimentos baseados em SAT, e nos de Bryant e colaboradores [BGV99], que apresentaram um procedimento que traduz a fórmula original em uma fórmula na lógica proposicional, interpretando a fórmula sobre um domínio de vetores de bits, por exemplo.

[1] Os procedimentos de SMT (de decisão ou não) são geralmente referidos como resolvedores SMT, fazendo uma referência aos resolvedores SAT.

3.2.1 Definições e notações básicas

Nesta subseção serão apresentados alguns conceitos e notações usados para definir uma fórmula em SMT, conforme apresentado por Barrett e colaboradores em [BSST09].

As variáveis de uma fórmula SMT estão ligadas à uma teoria lógica de interesse; o conjunto de variáveis e símbolos da teoria usada na fórmula é denominada assinatura e é representada por Σ. Exemplos de teorias podem ser vistos na Subseção 3.2.3. Os conectivos lógicos são os mesmos conectivos das fórmulas SAT: $\wedge$ (e), $\vee$ (ou) e $\neg$ (negação). Um átomo é uma variável da fórmula, valorado conforme a teoria de interesse. Um literal é um átomo ou a sua negação. Um módulo é uma fórmula constituída por variáveis e símbolos de Σ. Uma cláusula é uma disjunção de literais ou módulos. Uma cláusula unitária é uma cláusula composta por apenas um literal. Uma fórmula CNF é uma conjunção de zero ou mais cláusulas. Portanto, uma fórmula SMT é composta por uma fórmula SAT com módulos.

A fórmula

$$\neg(a \geq 2 \vee a \geq 5) \wedge (a \geq 2 \vee a \geq 4)$$

é um exemplo de uma fórmula φ em SMT, onde as desigualdades fazem parte do módulo de aritmética e a é um átomo de φ. Se φ for interpretada de acordo com a teoria aritmética inteira, um modelo μ para φ poderia ser a sendo igual a 5. Porém, para essa teoria, o modelo $(a = 5)$ não satisfaz φ, já que, na fórmula, a não pode ser maior ou igual a 5. Um modelo que satisfaz φ é $a = 4$.

As fórmulas recebem um valor verdade de acordo com o modelo da teoria de interesse. Um modelo μ é uma interpretação para a fórmula, ou seja, uma valoração para cada átomo da fórmula.

3.2.2 Resolução de Instâncias SMT

Conforme a lista de participantes da competição de resolvedores SMT, a SMT-COMP[2], que ocorre todos os anos, podemos encontrar mais de 20 resolvedores SMT, sejam eles de código aberto ou proprietário. Cada um desses resolvedores possui abordagens e métodos diferentes para a resolução das fórmulas SMT.

Os resolvedores SMT são uma extensão dos resolvedores de satisfazibilidade booleana (SAT). Então uma fórmula SMT pode ser reduzida à uma fórmula SAT. Para essa redução é necessário descobrir as incompatibilidades entre os átomos das duas fórmulas e eliminá-las para que ela seja equi-satisfazível, ou seja, para que os valores verdade das duas fórmulas, SAT e SMT, sejam os mesmos para todas as interpretações tanto da fórmula SAT quanto da fórmula SMT.

De acordo com o momento em que essas incompatibilidades são adicionadas à estrutura booleana do problema, existem duas abordagens de implementação para o SMT: a abordagem ansiosa (*eager*) e a abordagem preguiçosa (*lazy*).

3.2.2.1 Abordagem Ansiosa

A abordagem ansiosa consiste em transformar toda a fórmula SMT em um fórmula proposicional equi-satisfazível em um único passo, gerando, assim, um problema SAT. A partir dessa fórmula proposicional usa-se um resolvedor SAT para analisá-la, não sendo preciso desenvolver um resolvedor específico para a teoria.

[2] http://www.smtcomp.org/

Existem dois métodos principais para a transformação da fórmula SMT em fórmula SAT: o método de codificação de pequenos domínios (SD) ou, em inglês, *Small Domain Encoding* e o método de codificação direta, em inglês, *Direct Encoding* [BSST09].

No método de codificação de pequenos domínios, dada uma fórmula φ, primeiro calcula-se o limite polinomial do tamanho da solução S e, então, busca-se uma solução satisfatória para φ nesse espaço. No entanto, esse método não é válido para as fórmulas encontradas na prática, pois, para problemas envolvendo milhares de restrições e variáveis, as fórmulas booleanas geradas são muito grandes, ficando além da capacidade até dos melhores resolvedores SAT. Para casos especiais, esse método pode ser prático.

Já o método de codificação direta, também chamado de *Per-Constraint Encoding*, opera em três passos: primeiro, substitui-se cada restrição da fórmula como uma nova variável boleana, gerando uma fórmula boleana φ_1. Segundo, gera-se uma fórmula φ_2 que restringe os valores das variáveis de φ_1, para preservar a informação da fórmula original. Por último, chama-se um resolvedor SAT para a fórmula $\varphi_1 \wedge \varphi_2$. Esse método gera um número pequeno de restrições de transitividade e o problema SAT resultante pode ser facilmente resolvido. Além disso, diversas otimizações heurísticas são possíveis com base na estrutura booleana da fórmula [Str02].

A Figura 3.1 apresenta um esquema de codificação para a abordagem ansiosa. O codificador recebe como entrada uma fórmula SMT e a transforma para uma fórmula proposicional, essa nova fórmula é enviada para um resolvedor SAT que vai analisá-la, informando se a fórmula é satisfazível ou não.

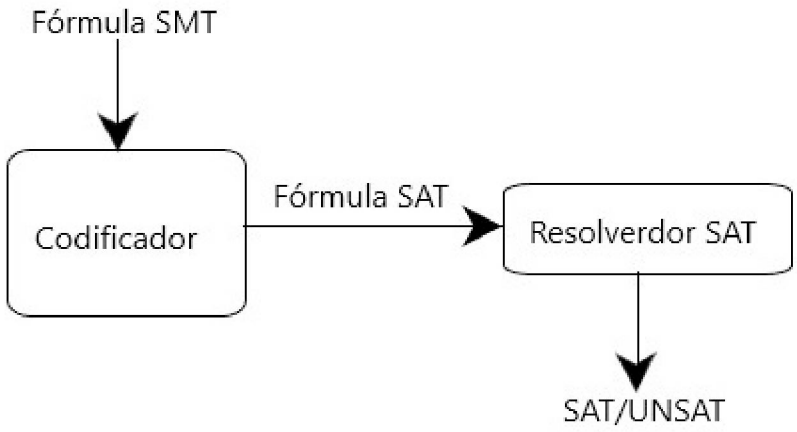


Figura 3.1: Esquema de codificação para a abordagem ansiosa. Adaptado de [Bru].

Embora existam resolvedores eficientes para uma série de teorias importantes [BT17], essa abordagem é pouco utilizada, já que o processo de tradução da fórmula pode gerar uma outra fórmula maior, tornando impraticável a representação computacional, devido a grande quantidade de memória requerida para essa representação.

3.2.2.2 Abordagem Preguiçosa

Outro modo de resolução de instâncias SMT é a abordagem preguiçosa, que consiste em integrar procedimentos de decisão para teorias, também chamados de resolvedores de teorias ou *T-Solvers* [BSST09], integrado com um resolvedor SAT.

Os resolvedores SMT apresentam os seguintes passos básicos:

1. Extração da estrutura;
2. Encontrar um modelo candidato com um resolvedor SAT. Caso não exista modelo, retornar UNSAT;
3. Verificar esse modelo em um resolvedor de teorias, caso o modelo seja satisfazível, retornar SAT. Senão, voltar ao passo 2.

A Figura 3.2 mostra um esquema geral de codificação para a abordagem preguiçosa. Nesse esquema é possível ver a integração entre o resolvedor SAT e o resolvedor de teorias.

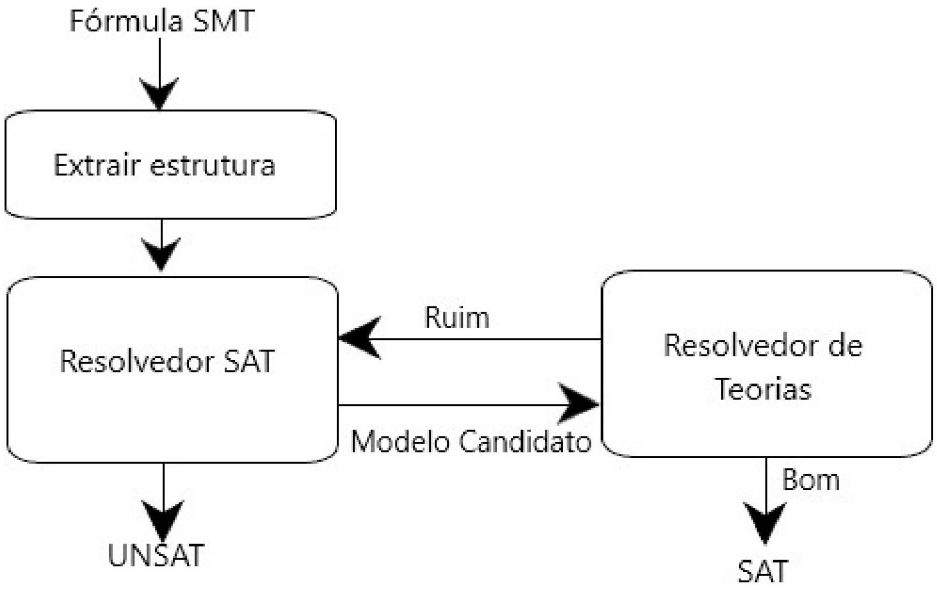


Figura 3.2: Esquema de codificação para a abordagem preguiçosa. Adaptado de [Bru] .

O primeiro passo dessa abordagem é extrair a estrutura, ou seja, transformar a fórmula φ em SMT em uma fórmula booleana φ^B (nesse passo pode ser necessário a conversão da fórmula resultante para CNF). Podemos ver um exemplo dessa extração com a fórmula

$$\neg(a \geq 2 \vee a \geq 5) \wedge (a \geq 2 \vee a \geq 4),$$

onde os módulos $a \geq 2, a \geq 5$ e $a \geq 4$, podem ser substituídos pelas variáveis proposicionais p, q e r, respectivamente, resultando na fórmula

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

A partir de φ^B, é chamado um resolvedor SAT que gerará um modelo μ^B. Gerado μ^B, o resolvedor de teorias verificará se este modelo é bom ou ruim para φ, ou seja, se, através de μ^B, existe um modelo μ que satisfaça φ. Caso o resolvedor de teorias não encontre μ que satisfaça φ, retorna para o resolvedor SAT e este busca outro modelo μ^B e o processo é repetido.

Por exemplo, para a fórmula apresentada acima, o modelo:

$$\mu^B = \{p = 1, q = 0, r = 0\}$$

torna φ^B insatisfazível, então, outro modelo é gerado para φ^B:

$$\mu^B = \{p = 0, q = 0, r = 1\}$$

esse modelo torna φ^B satisfazível, então, a partir de $\{p = 0, q = 0, r = 1\}$, temos:

$$\mu = \{a \geq 2 = 0, a \geq 5 = 0, a \geq 4 = 1\}$$

que torna φ satisfazível, onde o valor de a é 4.

Relações entre os modelos:

- Se μ é um modelo para φ, então μ^B é um modelo para φ^B;
- se μ^B não é um modelo para φ^B, então não existe um modelo μ que satisfaça φ;
- Podem existir vários modelos μ^B para φ^B que não mapeiam nenhum modelo μ para φ.

Resolvedor de teorias: Um resolvedor de teorias, ou *T-Solver*, é um procedimento que, dado um conjunto de literais γ em Σ, decide se γ é satisfazível para a teoria. Um bom resolvedor de teorias possui as seguintes características [BSST09, Seb07]:

- Geração de modelo: produzir um modelo da teoria que assuma a consistência de γ. Isso pode exigir esforço extra de computação.
  Exemplo: Seja a fórmula $(2a - b > 3) \wedge (c - 2b \leq 2) \wedge (b = d + 4)$. O resolvedor de teorias pode retornar como modelo o conjunto $\{a = 4, b = 4, c = 0, d = 0\}$;
- Geração de conjunto de conflitos: caso γ seja insatisfazível, um subconjunto λ de γ que gerou o conflito é produzido e $\neg\lambda$ é adicionado ao conjunto. Isso gera uma nova cláusula na fórmula proposicional, forçando o resolvedor SAT a fazer um *backtrack* ao encontrar esse conflito.
  Exemplo: Considere a fórmula $(a = 4 \vee a < 4) \wedge (a = 4 \vee a > 4) \wedge (\neg(a < 4) \vee a = 4$ e sua fórmula proposicional $(p \vee q) \wedge (r \vee p)$. Suponha que um modelo SAT para a fórmula proposicional seja $\{p = 1, q = 1, r = 1\}$. Para a fórmula SMT esse modelo é UNSAT, retornando o conjunto de conflitos $\lambda = \{(a = 4 \vee a < 4), (a = 4 \vee a > 4)\}$;
- Incrementação: lembrar os seus estados anteriores, evitando refazer verificações.
  Exemplo: Dada uma fórmula e suas cláusula $c_1, c_2, ..., c_n$:
  `T-Solver` (c_1) – SAT
  `T-Solver` $(c_1 \cup c_2)$ – SAT
  `T-Solver` $(c_1 \cup c_2 \cup c_3)$ – SAT;
- *Backtrackability*: desfazer as etapas, de forma eficiente, até chegar ao estado onde aconteceu um conflito
  Exemplo: Dada uma fórmula e suas cláusula $c_1, c_2, ..., c_n$:
  `T-Solver` (c_1) – SAT
  `T-Solver` $(c_1 \cup c_2)$ – SAT
  `T-Solver` $(c_1 \cup c_2 \cup c_3)$ – UNSAT
  desfaz c_3 e c_2
  `T-Solver` $(c_1 \cup c'_2)$ – SAT
  `T-Solver` $(c_1 \cup c'_2 \cup c'_3)$ – SAT;

- Dedução de literais: ser capaz de identificar as consequências de seu conjunto γ atual, ou seja, deduzir o valor de um literal a partir do valor dos outros literais.
  Exemplo [Seb07]: Seja a fórmula $(\neg(2a - b > 2) \wedge (3c - 2a \leq 3) \wedge \neg(3c - b \leq 6) \wedge (b = 3d + 4))$, onde o resolvedor de teorias retorna SAT para o conjunto $(\neg(2a - b > 2) \wedge \neg(3c - b \leq 6) \wedge (b = 3d + 4))$, retornando a dedução $\{\neg(2a - b > 2), \neg(3c - b \leq 6)\} \models \neg(3c - 2a \leq 3)$;

- Dedução de igualdades de interface: ser capaz de identificar as igualdades entre as variáveis e termos que ocorrem em γ.
  Exemplo: Dada a fórmula $(a \geq 0) \wedge (a \leq 1) \wedge (b = 0) \wedge (c = 1)$, pode deduzir a disjunção de igualdades $(a = b) \vee (a = c)$.

O Algoritmo Simplex e o Union-Find são exemplos de resolvedores de teorias para teoria aritmética e para a teoria de igualdades, respectivamente.

Uma das técnicas de implementação para se integrar um resolvedor SAT com um resolvedor de teorias se baseia no algoritmo DPLL. Essa técnica pode utilizar dois esquemas: *offline* e *online*.

O esquema de integração *offline* [BDS02, FJOS03], também chamado de abordagem "lemas sob demanda", ou, em inglês, *"lemmas on demand"* [DMRS02] é o método mais simples e consiste em alimentar o DPLL com a fórmula booleana φ^B para decidir se é satisfazível ou não.

Caso seja insatisfazível, a fórmula SMT φ também será. Caso contrário, retorna para o resolvedor de teorias um conjunto μ^B de literais que satisfaz a φ^B. Se μ^B for satisfazível em φ, então φ é satisfazível, caso contrário, $\neg\mu^B$ é adicionado a μ^B e o processo é repetido. Nesse método, o processo de verificar se a valoração encontrada também satisfaz φ só é feita após o resolvedor SAT encontrar uma valoração que satisfaz φ^B.

Um exemplo da resolução *offline* pode ser visto abaixo com a fórmula SMT φ:

$$\neg(a \geq 2 \vee a \geq 5) \wedge (a \geq 2 \vee a \geq 4),$$

Primeiramente, φ é convertida para uma fórmula SAT φ^B:

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

φ^B é submetida ao resolvedor SAT. Retornando o modelo μ^B de φ^B:

$$p = 0, q = 1, r = 1$$

verifica se μ^B satisfaz φ:

$$\neg(a \geq 2), (a \geq 5), (a \geq 4)$$

como não satisfaz, μ^B é aprendido e φ^B é novamente submetida ao resolvedor SAT. Retornando o modelo μ^B:

$$p = 1, q = 1, r = 0$$

verifica se μ^B satisfaz φ:

$$(a \geq 2), (a \geq 5), \neg(a \geq 4)$$

como não satisfaz, μ^B é aprendido e φ^B é novamente submetida ao resolvedor SAT. Retornando o modelo μ^B:

$$p = 1, q = 0, r = 0$$

verifica se μ^B satisfaz φ:

$$(a \geq 2), \neg(a \geq 5), \neg(a \geq 4)$$

μ^B satisfaz φ e o algoritmo é finalizado.

Apesar da sua simples implementação, não sendo necessário muitas alterações no resolvedor SAT, esse esquema não apresenta bons resultados, principalmente, para os resolvedores competitivos.

O algoritmo abaixo, adaptado de [Seb07], apresenta o funcionamento da ideia do esquema *offline*:

```
1  //Procedimento esquema offline
2  //Entrada: fórmula SMT φ
3  //Saída: SAT ou INSATISFAZÍVEL
4
5     φ^p = Teoria_para_Booleano (φ)
6     enquanto (DPLL(φ^p,μ^p) == SAT) faça
7        se (T-solver(Booleano_para_Teoria (μ^p)) == SAT) então
8           retorna SAT
9        φ^p = φ^p ∧ ¬μ^p
10    fim
11    retorna INSATISFAZÍVEL
```

Já o esquema *online* [BBC+06, FJOS03, ABC+02, ACG99] é mais sofisticado e é capaz de processar sua entrada de forma progressiva, ou seja, apresenta valorações parciais μ^P para o resolvedor de teorias. Caso μ^P seja insatisfazível, ela é aprendida e o resolvedor SAT faz um retrocesso afim de encontrar outra valoração que possa satisfazer φ. Nessa abordagem o aprendizado de conflitos ocorre durante a resolução da fórmula no resolvedor SAT, poupando trabalho do resolvedor que não precisa trabalhar em uma valoração que não irá satisfazer φ. Embora seja mais complexo, a maioria dos resolvedores SMT utiliza esse esquema para aproveitar os bons resultados, como o CVC4 [BCD+11], por exemplo.

Um exemplo da resolução *online* pode ser visto abaixo com a fórmula SMT φ:

$$\neg(a \geq 2 \vee a \geq 5) \wedge (a \geq 2 \vee a \geq 4),$$

Primeiramente, φ é convertida para uma fórmula SAT φ^B:

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

φ^B é submetida ao resolvedor SAT. Valora a variável p como falsa e verifica se essa valoração satisfaz φ:

$$\neg(a \geq 2)$$

Valora a variável r como verdadeira e verifica se essa valoração satisfaz φ:

$$\neg(a \geq 2), (a \geq 4)$$

Como não satisfaz, valora a variável p como verdadeira e verifica se essa valoração satisfaz φ:

$$(a \geq 2)$$

Valora a variável q como falsa e verifica se satisfaz φ:

$$(a \geq 2), \neg(a \geq 5)$$

Essa valoração satisfaz φ e o algoritmo é finalizado.

Um esquema de um mecanismo online DPLL, chamado *T*-DPLL, dirigido por conflitos, adaptado de [BSST09], é apresentado abaixo:

```
1  //Procedimento esquema online
2  //Entrada: fórmula SMT φ e conjunto vazio de literais μ
3  //Saída: SAT ou INSATISFAZÍVEL
4
5     se (pre_processo(φ, μ) == CONFLITO) então
6        retorna INSATISFAZÍVEL
7     φ^B = Teoria_para_Booleano(φ)
8     μ^B = Teoria_para_Booleano(μ)
9     enquanto (1) faça
10       decide_próximo_literal(φ^B,μ^B)
11       enquanto (1) faça
12          status = BCP(φ^B,μ^B)
13          se (status == SAT) então
14             μ = Booleano_para_Teoria (μ^B)
15             retorna SAT
16          fim
17          senão se (status == CONFLITO) então
18             nível = analisa_conflito(φ^B,μ^B)
19             se (nível == 0) então
20                retorna INSATISFAZÍVEL
21             senão
22                retrocesso(nível,φ^B,μ^B)
23          fim
24       fim
25    fim
26 }
```

O algoritmo acima recebe como entrada uma fórmula SMT φ e um conjunto inicialmente vazio de literais μ de φ. A função `pre_processo` simplica φ e modifica μ, se for necessário. Se a fórmula resultante não for satisfazível, retorna INSATISFAZÍVEL. As funções `Teoria_para_Booleano` e `Booleano_para_Teoria` mapeiam a teoria para booleano e booleano para a teoria, respectivamente. A função `decide_próximo_literal` seleciona o próximo literal para ser valorado e a função `BCP` propaga esse valor e retorna "SAT"ou *"CONFLITO"*. Se houver conflito, a função `analisa_conflito` detecta o conjunto que causou o erro e faz um retrocesso.

A abordagem preguiçosa é a mais predominante para a resolução de problemas SMT, por ser mais flexível e geral. Muitos dos resolvedores conhecidos utilizam essa abordagem, como, por exemplo: ArgoLib [MJ04], CVC3 [BT07], Fx7 [ML06], Math- SAT [BBC$^+$05], Yices [DdM06] e Z3 [DMB08].

Uma outra alternativa para a resolução de problemas SMT é combinar as abordagens ansiosa e preguiçosa. Por exemplo, a expansão de Ackermann [Ack54] é uma forma de combinar as duas abordagens (ver seção 12.7 de [BSST09]).

3.2.3 Teorias SMT

As teorias de interesse são tradicionalmente definidas como conjuntos de tipos e funções que associam um tipo às variáveis de uma fórmula SMT [BFT17, Cok13].

Para auxiliar a pesquisa e o desenvolvimento do SMT, em 2003, Clark Barrett, Pascal Fontaine e Cesare Tinelli criaram uma biblioteca chamada SMT-LIB [BFT16]. Um dos objetivos

da iniciativa SMT-LIB é definir uma lista de teorias de interesse, começando com um pequeno número de teorias e, aos poucos, acrescentando outras [BFT17].

A atual versão da biblioteca SMT-LIB distingue as teorias básicas das teorias combinadas. As teorias básicas, como a teoria dos números reais, são bem definidas na biblioteca, enquanto as teorias combinadas (aquelas obtidas da combinação de diferentes teorias) são definidas em termos das teorias básicas por meio de um operador de combinação modular [BFT17].

As teorias básicas apresentadas na SMT-LIB são as seguintes: *ArraysEx*, *FixedSizeBitVectors*, *Core*, *Ints*, *Reals, Reals_Ints* e *FloatingPoint*.

- A teoria *ArrayEx* define o tipo e funções para ler e escrever elementos de matrizes. Esse tipo, definido como matriz, recebe dois parâmetros, o primeiro é o índice e o segundo o tipo dos elementos da matriz. Essa teoria também possui duas funções, a função `select`, que extrai os valores da matriz e a função `store`, que cria uma nova matriz com algum elemento modificado em um determinado índice.

  Um exemplo para essa teoria é a fórmula $(i = j \wedge a[j][j] = 2) \Rightarrow a[i][i] = 2$;

- A teoria *FixedSizeBitVectors* descreve o comportamento dos vetores de bit, contendo operações para manipular, combinar e extrair os vetores. Os vetores de bits são uma representação binária sem sinal de um número inteiro não negativo, com o bit menos significativo do lado direito. Uma observação importante é que algumas operações dessa teoria interpretam os vetores de bit como um número natural. Essa teoria apresenta um tipo, chamado `_BitVec` m, onde m é um número maior que 0. Literais binários e hexadecimais são definidos para obter a quantidade de dígitos no vetor. A teoria *FixedSizeBitVectors* apresenta várias funções para trabalhar com os vetores de bit, entre elas a função `concat`, que concatena dois vetores de bit, a função `_extract`, que extrai um sub vetor, do índice i ao índice j e a função `bvult`, para comparar. Outras funções são, as funções unárias, `bvnot` e `bvneg`, e as funções binárias, `bvand`, `bvor`, `bvadd`, `bvmul`, `bvudiv`, `bvurem`, `bvshl` e `bvlshr`.

  A fórmula $(b >> i)\&1 = 1$ é um exemplo para a teoria de vetores de bit;

- A teoria *Core* define os elementos e operações básicas da lógica booleana. Apresenta o tipo `Bool`. O conjunto {`true`, `false`} dos valores booleanos são tratados como constantes internas. As funções definidas nessa teorias são: a operação `not`, para a negação de uma variável, as funções associativas, `and`, `or`, `xor` e `=>`, a função de igualdade (`chainable`), entre os elementos, a função `distinct` de desigualdade (`pairwise`) e a função `ite` (*if-then-else*).

  A fórmula $p \wedge \neg p$ é um exemplo para essa teoria.

- A teoria *Ints* apresenta os elementos básicos da aritmética inteira. Define o tipo `Int` e denota o conjunto dos números inteiros como constantes internas da teoria. As funções de adição ($+$), subtração ($-$), multiplicação ($*$), módulo (`mod`), divisão inteira (`div`) e valor absoluto (`abs`) são definidos nessa teoria. O símbolo $-$ é usado tanto para a operação de subtração (com dois argumentos) quanto para a negação (com um argumento). Também define as funções de comparação, que retornam um booleano, como a função de maior ($>$), menor ($<$), maior igual ($>=$) e menor igual ($<=$).

  Um exemplo para a teoria de aritmética de números inteiros é a fórmula $(4y + 3x > 4) \vee (x - 3 < 3)$

- A teoria *Reals* define os elementos e operações usados no conjunto dos números reais. Um tipo `Real` é definido. Numerais e decimais são definidos como reais. Também são apresentadas as funções de adição ($+$), subtração ($-$), multiplicação ($*$) e divisão ($/$), além das funções de comparação, como a função de maior ($>$), menor ($<$), maior igual ($>=$) e menor igual ($<=$).

  A fórmula $(1.5y + 3.0x > 1.5) \vee (x - 2.5 < 2.5 + y)$ é um exemplo para a teoria dos reais.

- A teoria *Reals_Ints* é uma combinação da teoria dos números inteiros e da teoria dos números reais, porém não é a mesma coisa que a união de ambas. Essa teria apresenta dois tipos, o tipo `Int`, para os inteiros e o tipo `Real`, para os reais. Os numerais são definidos como constantes internas inteiras e os decimais como constantes internas reais. As funções de negação ($-$) adição ($+$), subtração ($-$), multiplicação ($*$), módulo (`mod`), divisão inteira (`div`), valor absoluto (`abs`), maior ($>$), menor ($<$), maior igual ($>=$) e menor igual ($<=$) são definidas, nessa teoria, para argumentos do tipo inteiro e as funções de adição ($+$), subtração ($-$), multiplicação ($*$), divisão ($/$), maior ($>$), menor ($<$), maior igual ($>=$) e menor igual ($<=$) são definidas para argumentos do tipo real. Nessa teoria também estão definidas funções que mapeiam argumentos do tipo `Int` para argumentos do tipo `Real` e vice-versa, são elas, `to_int` e `to_real`, respectivamente. A função `is_int` mapeia um argumento do tipo `Real` para um argumento do tipo `Bool`.

  A fórmula $(1y + 3x > 1.5) \vee (x - 2.5 < 2 + y)$ é um exemplo para essa teoria.

- A teoria *FloatingPoint* define os elementos e operações para a aritmética de números em ponto flutuante. Define dois tipos, o tipo `RoundingMode`, para o arredondamento de números e o tipo `Real`, para números reais. Literais binários e hexadecimais, do tipo `_BitVec` são definidos para obter a quantidade de dígitos no vetor. As funções de valor absoluto, negação, adição, subtração, multiplicação, divisão, raiz quadrada, sobra, mínimo e máximo são definidas, além de funções de comparações, classificação e conversão de tipos.

  A fórmula $(1y + \sqrt{3x} > 1.5) \vee (x - 2.5 < 2 + \sqrt{y})$ é um exemplo para essa teoria.

3.2.4 Lógicas SMT

Uma lógica define, por meio de uma ou mais teorias, todos os tipos, funções e símbolos constantes que compõem o conjunto inicial de definições dentro da lógica, juntamente com algumas restrições sobre os tipos de expressões. Por exemplo, a lógica QF_LIA inclui as teorias *Core* e *Ints*. Essas teorias definem muitas operações usuais para booleanos e inteiros, apesar disso, a lógica QF_LIA não permite expressões quantificadas e permite apenas uma aritmética linear. Tais restrições são definidas porque existem procedimentos de decisão que podem resolver problemas de satisfação nessas lógicas.

As lógicas da SMT-LIB foram nomeadas usando grupos de letras que evocam as teorias usadas pelas lógicas e algumas restrições importantes em suas linguagens, com as seguintes convenções [BFT16]:

- QF para fórmulas livres de quantificadores;
- A ou AX para a teoria *ArraysEx*;

- BV para a teoria *FixedSizeBitVectors*;
- FP para a teoria *FloatingPoint*;
- IA para a teoria *Ints*;
- RA para a teoria *Reals*;
- IRA para a teoria *Reals_Ints*;
- IDL para lógicas de diferenças para inteiros;
- RDL para lógicas de diferenças para racionais;
- L antes IA, RA ou IRA para o fragmento linear dessas aritméticas;
- N antes IA, RA ou IRA para o fragmento não linear dessas aritméticas;
- UF para a extensão que permite símbolos de classificação e função livres.

A Figura 3.3 apresenta uma visão geral das lógicas existentes na atual versão da SMT-LIB.

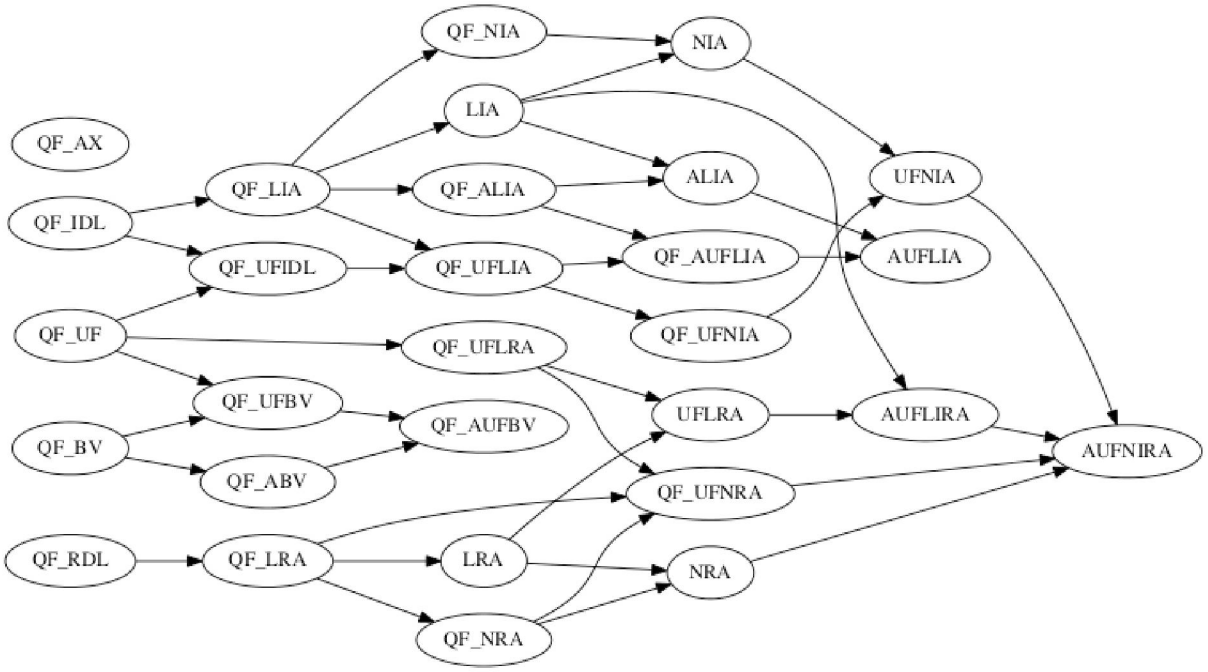


Figura 3.3: Visão geral das lógicas existentes na atual versão da SMT-LIB [BFT16].

Arquivos de texto que definem as lógicas e as teorias citadas nesse texto estão disponíveis no site SMT-LIB [BFT16].

3.3 CONCLUSÃO

Este capítulo apresentou conceitos importantes sobre satisfazibilidade, trazendo um breve histórico do SAT, apresentando alguns resolvedores e a motivação para que surgisse o SMT. Também apresentou definições de SMT, sua trajetória, descrições sobre a resolução de fórmulas SMT e os principais métodos de resolução utilizados nos resolvedores modernos, além de uma explicação breve sobre as teorias de interesse do SMT.

4 PLANEJAMENTO MÓDULO TEORIAS

Muito esforço foi gasto para ampliar o alcance do planejamento, a fim de poder ir além do domínio proposicional, no qual a inclusão de fatores como tempo, quantidade, espaço, etc. torna-se possível. Assim, cada problema de planejamento deve ter um algoritmo personalizado, com sua própria sintaxe e integração para tratá-lo. Motivados com isso, em 2012, Gregory e colegas, em [GLFB12], propuseram um novo formalismo para o planejamento, o Planejamento Módulo Teorias, em inglês, *Planning Modulo Theories* (PMT). Ao alcance do nosso conhecimento, poucas informações sobre o PMT foram encontradas na literatura.

Desta forma, este capítulo é baseado no artigo *Planning Modulo Theories : Extending the Planning Paradigm* [GLFB12]. Apresenta-se o problema PMT na seção 4.1 e a descrição do problema na seção 4.2. Na seção 4.3 apresenta-se o planejador PMTPlan, suas heurísticas e desempenho para problemas com recursos, testados pelos próprios autores. Para um melhor entendimento do funcionamento desse planejador, essa seção também mostra com exemplo prático as sequências de passos para a busca do plano. E, por último, na seção 4.4, apresenta-se um novo domínio, criado pela autora desta dissertação, denominado Problema Temporal dos Jarros, no qual se faz uso de duas noções de planejamento: tempo e recursos.

4.1 O PROBLEMA PMT

O PMT é uma estrutura modular inspirada no SMT (ver capítulo 3) que permite que o planejamento clássico seja ampliado em um caminho modular de forma semelhante ao SMT. Isso significa que módulos com funções que operam sobre as novas teorias adicionadas ao planejador serão criados e usados em ações de planejamento. Do mesmo modo, como visto no capítulo 2, o PMT também assume a hipótese de mundo fechado.

Estudos para incluir predicados e funções e ampliar o alcance do planejamento clássico vem desde os anos 1990. Podemos ver nos trabalhos de Currie e Tate [CT91] com o planejamento hierárquico e Bacchus e Kabanza [BK00] com a pesquisa guiada por verificação de regras. Ambos com o uso de ligações semânticas. Geffner [Gef00] propôs em 2000 o STRIPS funcional, que é muito semelhante ao PMT. Apesar do STRIPS funcional ser uma boa ideia, ele não fazia das teorias um parâmetro explícito das codificações e não foi continuado. Dornhege e colegas [DEK$^+$09] em 2009 propuseram uma extensão do PDDL com módulos de ligações semânticas.

Já alguns pesquisadores observaram a adequação que o SMT tem para lidar com as teorias, em que trata o problema de planejamento como planejamento clássico, complementa-o com as teorias específicas e usa-se a compilação do SMT. Alguns exemplos de trabalhos são: o LP-SAT [WW99], criado por Wolfman e Weld em 1999, o TM-LPSAT [SD05] de Shin e Davis criado em 2005 e o TLP-GP [MR08].

A definição de PMT estende o problema de planejamento clássico de forma análoga à maneira como o SMT estende o SAT. De maneira geral, um problema PMT contém as seguintes definições [GLFB12]:

- Estado: um estado é uma avaliação sobre um conjunto finito de variáveis, V, no qual cada variável, $v \in V$, tem um domínio correspondente de valores possíveis, D_v. A expressão $s[v]$ denota o valor que o estado s atribui à variável v e $s[v = x]$ é o estado que é idêntico a s, exceto que ele assinala o valor x para a variável v;

- Espaço de estado: para um conjunto de variáveis V é o conjunto de todas as avaliações sobre V;
- Sentença de primeira ordem sobre um espaço de estados S módulo T:

  é uma sentença de primeira ordem sobre as variáveis do espaço de estado, símbolos constantes, símbolos de função e símbolos predicados, em que T é uma teoria que define os domínios das variáveis e as interpretações para as constantes, funções e predicados;
- Termo sobre S módulo T: é uma expressão construída com os símbolos definidos por S e T;
- Ação: uma ação, A, para um espaço de estado S módulo T, é uma função de transição de estado, compreendendo:
  - Uma sentença de primeira ordem sobre S módulo T, Pre_A (pré-condição de A);
  - Um conjunto, Eff_A (efeitos da ação A), de atribuições a um subconjunto das variáveis do estado S, no qual cada configuração de uma variável para um valor é definido por um termo sobre S módulo T.
- Aplicação de um ação: uma ação, A, para o espaço de estado S módulo T, é aplicável (ou executável) em um estado $s \in S$ se T, $s \models Pre_A$ (ou seja, a teoria, juntamente com a avaliação s, satisfaz a pré-condição de A). Após a aplicação de A, o estado s é atualizado pelas atribuições em Eff_A para as variáveis que eles afetam. Todas as outras variáveis permanecem inalteradas.
- Plano:

  é uma sequência de ações $\langle a_1, ..., a_n \rangle$, de modo que a_i é aplicável no estado s_{i-1} e s_i é o resultado da aplicação a_i para $s_i - 1$, onde $s_0 = IeT, s_n| = G$.
- Um problema PMT, para uma teoria T, é uma tupla ‹S, A, I, G›, onde:
  - S é um espaço de estados no qual todas as variáveis dos domínios são definidos em T;
  - A é uma conjunto de ações para S módulo T;
  - I é uma avaliação inicial (o estado inicial);
  - G é uma sentença de primeira ordem sobre S módulo T (o objetivo).

4.2 DESCREVENDO PROBLEMAS PMT

Para suportar a descrição dos domínios PMT, Gregory e colegas [GLFB12] propuseram uma variante modular do PDDL, que visa a adição de teorias.

Os arquivos do módulo contêm declarações das funções que manipulam as teorias e estão escritas em MDDL (Module Definition Description Language). As definições das variáveis e ações que modelam um domínio de planejamento são fornecidas em arquivos principais, que estão escritos em Core Domain Description Language (CDDL). A ideia dessa nova linguagem modular se assemelha ao SMT-lib [BFT16], pois permite adicionar novas teorias, em que novos tipos, com funções e variáveis da teoria lógica de interesse são proporcionados.

Cada módulo pode definir um único tipo de teoria com constantes e funções. As funções podem ser sobre o novo tipo e também sobre os tipos definidos em outros módulos.

A título de exemplo, o código a seguir apresenta a descrição de um módulo, denominado *set*, o qual representa a teoria de conjuntos polimórficos.

```
(define (module set)
    (:type set of a')
    (:functions
        (construct-set  ?x+ - a')                       - set of a'
        (empty-set)                                     - set of a'
        (cardinality    ?s - set of a')                 - integer
        (member         ?s - set of a' ?x - a')         - boolean
        (subset         ?x - set of a' ?y - set of a')  - boolean
        (union          ?x - set of a' ?y - set of a')  - set of a'
        (intersect      ?x - set of a' ?y - set of a')  - set of a'
        (difference     ?x - set of a' ?y - set of a')  - set of a'
        (add-element    ?s - set of a' ?x - a')         - set of a'
        (rem-element    ?s - set of a' ?x - a')         - set of a'
    )
)
```

Uma das funções definidas nesse módulo é `construct-set`, que constrói um novo conjunto, o parâmetro usado nessa função é `?x+ - a'`, que quer dizer que a função leva uma ou mais constantes do tipo `a'`. Como é útil ter número variável de argumentos em chamadas de funções, é usada a sintaxe '+'. Outras funções são: `cardinality`, que retorna um valor do tipo inteiro (teoria já definida em um outro módulo separado) que representa a quantidade de elementos no conjunto. A função `member`, verifica se determinada variável está presente no conjunto. Já a função `subset` confere se um conjunto é subconjunto. As funções `union`, `intersect` e `difference` retornam, nesta ordem, a união, intersecção e diferença de dois conjuntos. Enquanto que as funções `add-element` e `rem-element` retornam um conjunto com um novo elemento adicionado ou removido, respectivamente.

Um arquivo CDDL, que descreve o domínio do problema, contém um cabeçalho, que define nomes e valores, os módulos, que são necessários para aquele domínio e as funções, que definem variáveis de estado para o domínio. Nesse arquivo também são descritas as ações do planejador. As ações, assim como em PDDL, são definidas em duas partes: uma lista de pré-condições e uma lista de efeitos.

Para melhor entender cada parte dos arquivos CDDL, será usado como exemplo um domínio para o Problema dos Jarros [Ric85]. Este desafio envolve um conjunto finito de jarros com capacidades inteiras conhecidas. Inicialmente, cada jarro contém uma quantidade inteira definida de líquido, não necessariamente igual à sua capacidade. A partir disso a proposta é alcançar um estado objetivo, especificado em termos da quantidade de líquido que deve estar presente alguns jarros, através de passos de transferir líquidos de um jarro para outro.

A seguir é apresentado uma descrição de um cabeçalho de um domínio para um problema dos jarros em CDDL.

```
(define (domain jugpour)
  (:types
   jug
  )
  (:modules integer set)
  (:functions
     (quantity ?t - jug) - integer
     (space    ?t - jug) - integer
     (capacity ?t - jug) - integer
  )
```

Nesse cabeçalho estão especificados o tipo jarro (*jug*), os módulos de inteiros (*integer*) e conjuntos (*set*), este apresentado acima, e as funções de *quantity* para especificar a quantidade existente no jarro, de *space* para informar o espaço restante no jarro e *capacity* para indicar a capacidade do jarro.

Um exemplo de uma ação para um problema dos jarros em CDDL é mostrado a seguir. Como exemplo de descrição de uma ação, o trecho logo a seguir descreve a ação de encher um jarro. As demais ações podem ser vistas no apêndice A.

```
(:action fill-jug
  :parameters
   (?t - jug)
  :precondition
   (true)
  :effect
   (
    (assign (quantity ?t) (capacity ?t))
    (assign (space ?t) 0)
   )
)
```

Consideremos como exemplo uma instância do problema dos jarros no qual existem 4 jarros, inicialmente vazios. Os trechos de código abaixo apresentam a descrição dessa instância em CDDL. O objetivo é deixar algum dos jarros com quantidade total de 1 litro.

Primeiramente, é descrito o cabeçalho dessa instância. São requeridos quatro objetos do tipo jarros (*jugs*).

```
(define (problem jugs)
(:domain jugpour)
(:objects
 jug1 jug2 jug3 jug4 - jug
)
```

O trecho abaixo representa o estado inicial da instância e é ilustrado na Figura 4.1.

Os 4 jarros, enumerados do 1 ao 4, com capacidade de 3, 6, 9 e 23 litros, respectivamente, estão inicialmente vazios. O espaço livre é igual a capacidade.

```
(:init
         (assign (quantity jug1) 0)
         (assign (quantity jug2) 0)
         (assign (quantity jug3) 0)
         (assign (quantity jug4) 0)
         (assign (space jug1) 3)
         (assign (space jug2) 6)
         (assign (space jug3) 9)
         (assign (space jug4) 23)
         (assign (capacity jug1) 3)
         (assign (capacity jug2) 6)
         (assign (capacity jug3) 9)
         (assign (capacity jug4) 23)
)
```

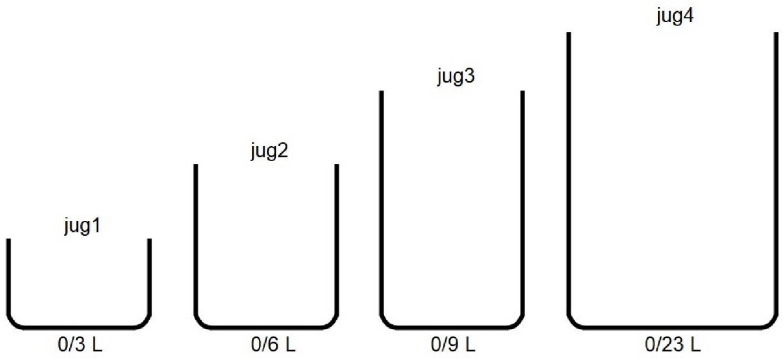


Figura 4.1: Estado inicial.

E, por último, a descrição do estado objetivo da instância e ilustrado na Figura 4.2. Um dos jarros deve estar com a quantidade de 1 litro.

```
(:goal
        (member (construct-set (quantity jug1) (quantity jug2)
        (quantity jug3) (quantity jug4) ) 1)
)
)
```

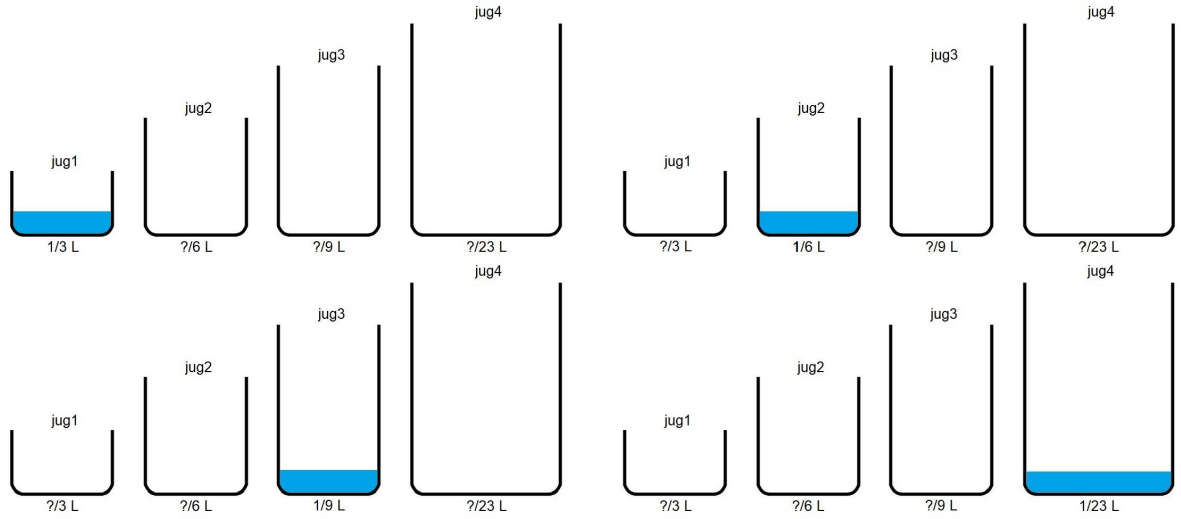


Figura 4.2: Subconjunto de possíveis estados objetivos.

4.3 O PLANEJADOR PMTPLAN

O PMTPlan é um planejador que está em fase experimental e foi proposto por Gregory, Derek, Maria Fox e Beck [GLFB12].

Esse planejador trata o problema PMT diretamente, então, o problema e o domínio da situação já são especificados no formato PMT, como explicado na seção 4.2.Está escrito em Java e está dividido em duas partes principais: *Modules* e *Core*.

A parte denominada *Modules* é encarregada dos módulos das teorias e contém os arquivos, denominados sub-resolvedores, responsáveis pela análise da estrutura de cada módulo. Enquanto a parte *Core* é responsável pela busca do plano. Este acessa cada parte do *Modules*.

A Figura 4.3 apresenta uma esquematização do PMTPlan.

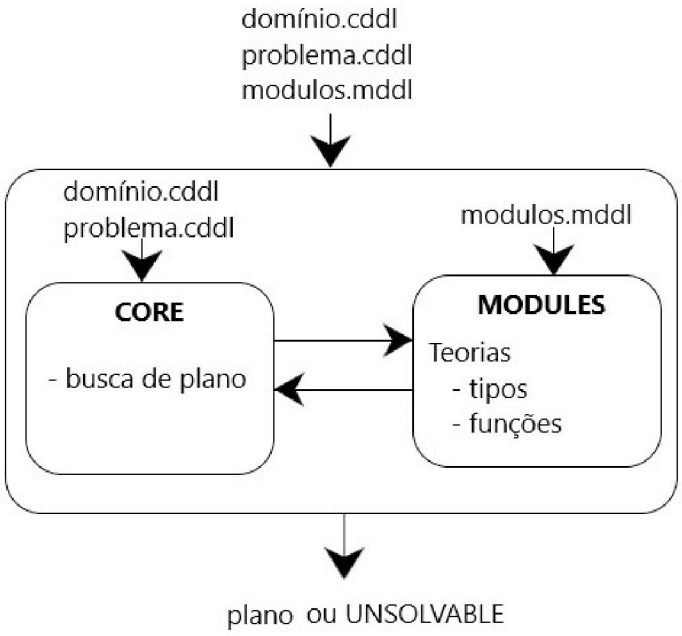


Figura 4.3: Esquematização do PMTPlan

Na Figura 4.3 tem-se como entrada do planejador dois arquivos CDDL, que descrevem o domínio e a instância do problema. Também, arquivos MDDL contendo a descrição de cada teoria utilizada para a busca do plano.

Assim que é inicializado, a primeira tarefa do algoritmo é fazer uma análise sintática dos arquivos de entrada, abstraindo e decompondo em partes para a estruturação da busca pela solução do problema.

Após essa análise, o próximo passo do PMTPlan é executar a busca em profundidade com a utilização de heurísticas pelo plano solução utilizando para a procura funções das teorias especificas na entrada do algoritmo. A saída do algoritmo será um plano solução ou um comunicado de que não existe solução para o problema.

Pode-se ver esse funcionamento no pseudo-código na página a seguir:

```
1
2  PMTPlan(domain, problem, modules ){
3
4     read(domain, problem, modules);
5
6
7     /*Faz o parser dos arquivos*/
8     MDDLparser.parseModuleFile(modules);
9     unground = CDDLparser.parseFiles(domain, problem);
10
11    /* Separa os módulos */
12    for (modules : ModuleStore.getModules())
13
14    /* Inicializa a teoria de interesse, o estado inicial,
15    o conjunto de ações, o estado objetivo */
16    ground = unground.ground();
17    initialState = ground.getInitialState();
18    initialState.setActions(ground.actions);
19    abstraction = IdentityAbstraction.getInstance();
20    concreteState = new(initialState, abstraction);
21    concreteState.goal = ground.goal.project(abstraction);
22    initialState.goal = ground.goal;
23
24    /*Faz o cruzamento de ações para o estado inicial*/
25    completeFoldingRPG = new(initialState,
26          OverGeneralRelaxedAbstraction.getInstance());
27    completeFoldingRPG.getFinalLayer();
28    initialState.actions = completeFoldingRPG.getActionSet();
29    action.isApplicable(initialState,
30                IdentityAbstraction.getInstance()));
31    Collections.shuffle(initialState.actions);
32
33    /*Faz a busca pelo plano*/
34    busca = BestFirstSearch(initialState, IdentityAbstraction.getInstance(),
35                      OverGeneralRelaxedAbstraction.getInstance());
36
37
38    /*confere se o estado objetivo foi alcançado*/
39    goalS = busca;
40    if (goalS == null){
41       "UNSOLVABLE";
42    }
43    else
44       return getPlan(goalS);
45
46 }
```

4.3.1 Abstrações e Heurísticas para o Planejador

Uma das técnicas mais fortes no planejamento moderno é o uso de problemas relaxados para fornecer uma estimativa do comprimento do plano, fazendo uso de heurísticas. Uma heurística simples é o h_{max} que pode ser usado para determinar o plano mais curto no estado relaxado alcançável [GH00].

Também, como Haslum e Geffner [GH00] observaram, o h_{max} é equivalente a h_1. Constrói-se uma análise de alcançabilidade relaxada do estado e é avaliado até que as metas sejam satisfeitas. Se as metas ou as pré-condições são sentenças proposicionais arbitrárias sobre as variáveis de estado, então, alcançar um comportamento monótono requer uma modificação dessa abordagem. Na análise de alcançabilidade, considera-se que as variáveis proposicionais adotam a lógica multi-valorada, ou seja, existem mais de dois valores verdade, $\{T, F, TouF\}$. O estado inicial em uma análise de alcançabilidade começa com cada variável designada $TouF$ de acordo com seu valor (verdadeiro ou falso) no estado avaliado. As sentenças são avaliadas neste domínio com uma interpretação modificada dos conectivos lógicos.

Isso serve para motivar as seguintes definições, que generalizam as ideias: a noção de uma abstração de cada valor do domínio, como $\{T, F, TouF\}$, descrito acima; uma abstração da teoria usada em um modelo de PMT para o domínio $\{T, F, TouF\}$, a definição estendida dos conectivos lógicos; Por último, um operador, chamado de operador de dobra (*folding*), para combinar valores abstratos prévios com novos valores durante a atualização de variáveis, que é o papel desempenhado por disjunção no relaxamento e união de exclusão para o domínio $\{\{true\}, \{false\}, \{true, false\}\}$.

Assim, temos duas definições apresentadas em [GLFB12]:

Definição 4.3.1. ***Abstração do Domínio:*** *Diante de um problema de PMT, $P = \langle S, A, I, G\rangle$, onde S é um conjunto de avaliações para variáveis $\langle v_1, ..., v_n\rangle$, uma abstração de domínio de P é uma espaço de estados abstraído, $A(S) = A(D_{v_1}) \times ... \times A(D_{v_n})$, onde:*

- *$A(D_{v_i}$ é um tipo chamado de abstração de D_{v_i} e há uma função associada $\texttt{abstract} :: D_{v_i} \rightarrow A(D_{v_i})$;*
- *para $s \in S$, $\texttt{abstract}(s)$ é definido como sendo o resultado da aplicação $\texttt{abstract}$ para cada valor atribuído por s;*
- *o estado inicial abstrato é a avaliação $\texttt{abstract}(I)$;*
- *$A(T)$ é uma abstração de T que define os comportamentos das funções, predicados e constantes de T interpretados sobre os tipos $A(D_{v_1}), ..., A(D_{v_n})$;*
- *para qualquer sentença S e estado $s \in S$, tal que T, $s \models S$, $A(T)$, $\texttt{abstract}(s) \models S$;*
- *cada $A(D_{v_i})$ está associado a um operador de dobra $\oplus$, tal que, para qualquer x, $y \in A(D_{v_i})$, qualquer sentença S e qualquer estado $s \in A(S)$, se $A(T)$, $s[v_i = x] \models S$ ou $A(T)$, $s[v_i = y] \models S$ então $A(T), s[v_i = x \oplus y] \models S$.*

Definição 4.3.2. ***Análise de Alcançabilidade:*** *Para uma abstração de domínio de um problema PMT, é uma sequência de estados abstratos, $s_0, ..., s_k$, onde s_0 é o estado inicial abstrato e, para cada $i = 1, ..., k$, $s_i = \texttt{apply}(\cup_{As.t.s_{i-1}, A(T) \models PreA} Eff_A, s_{i-1})$, onde:*

- *$\texttt{apply}(es, s)$ = $\texttt{doEach}(es, s, s)$;*

- $doEach(\{e\} \cup es, s', s) = doEach(es, doOne(e, s', s), s)$;
- $doOne(v := x, s', s) = s'[v = s'[v] \oplus evaluate(xs)]$.

A penúltima parte da definição 4.3.1 faz da abstração um relaxamento, enquanto a última parte assegura que a operação de dobramento fornece a mono-tonicidade necessária na análise de alcançabilidade definida na definição 4.3.2. Exemplos de planejadores que ilustram essas definições são o MetricFF [Hof03] para a definição 4.3.1 e o LPRPG [CFLS08] para a definição 4.3.2.

No PMTPlan, várias abstrações de domínio foram exploradas. Como a abstração de identidade, que deixa o espaço inalterado, a abstração enumerada, na qual um tipo t, é associado ao tipo abstrato $P(t)$, descrita para o tipo booleano, a abstração finita, na qual um valor limite é adicionado a um subconjunto finito de valores (conjunto base) e abstração de limites, que usa um espaço de valor abstrato que representa limites no intervalo de valores possíveis do tipo base.

Assim, o PMTPlan usa duas abstrações: a abstração nível de plano, para o espaço no qual um plano é procurado e outra, abstração heurística, para a análise de alcançabilidade. Essas abstrações são construídas automaticamente.

4.3.2 Comportamento do PMTPlan

Para um melhor entendimento do PMTPlan, aqui apresenta-se de forma instrutiva o comportamento do planejador ao realizar a busca do plano. Para isso, considera-se como exemplo uma instância do problema do jarros, apresentado na Seção 4.2, para mostrar a árvore de busca e as mudanças de estados aplicados ao problema. Lembrando que esta instância possui 4 jarros, de capacidade 3, 6, 9 e 23, respectivamente, inicialmente todos estão vazios.

Como saída, o PMTPlan gera um plano de tamanho 6 com as seguintes ações:

```
fill-jug jug3
fill-jug jug2
transfer3 jug3 jug4
transfer3 jug2 jug4
fill-jug jug3
transfer1 jug3 jug4
```

A árvore de busca para a esta instância pode ser vista na Figura 4.4.

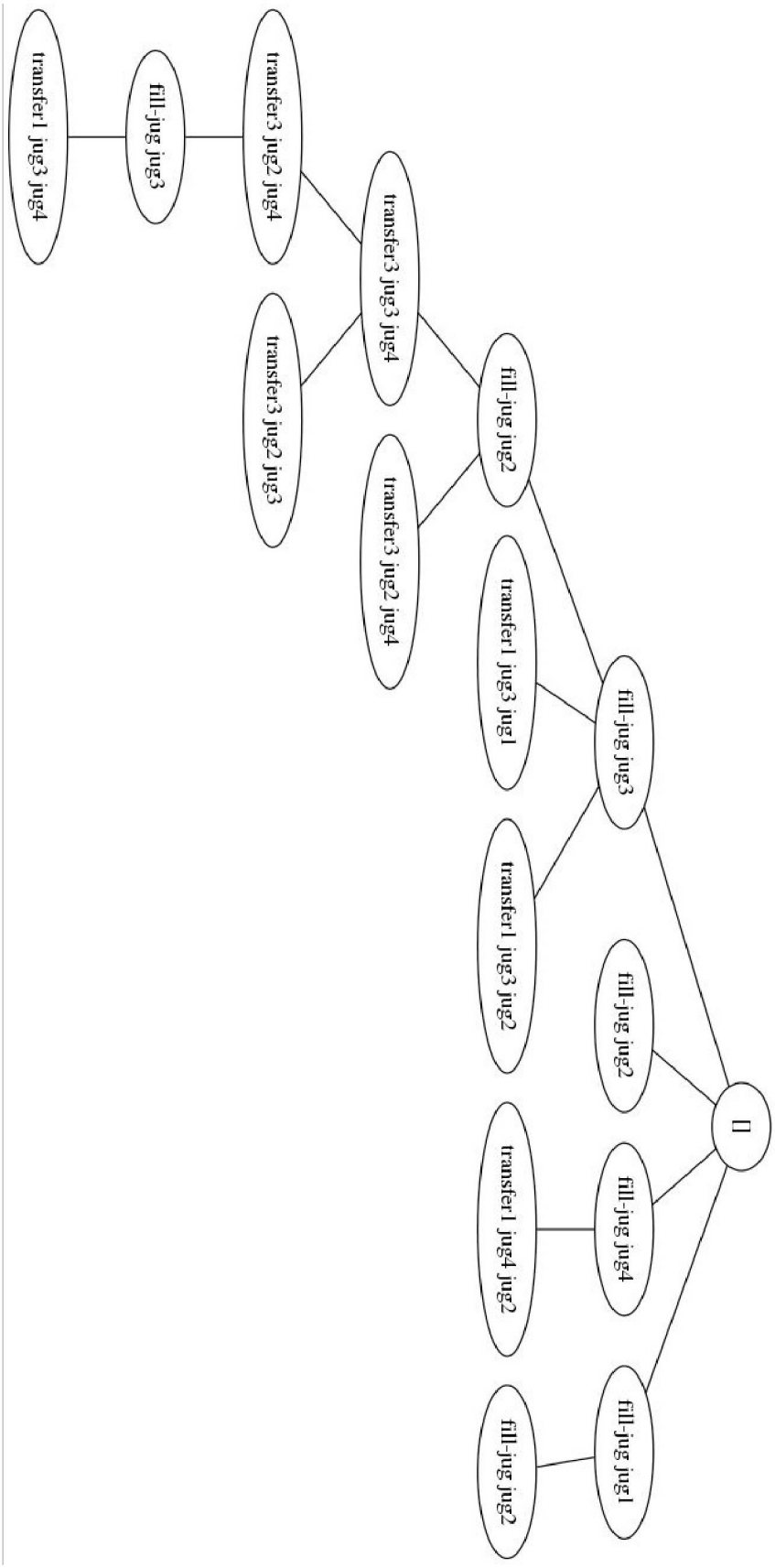


Figura 4.4: Árvore de busca para a instância com um conjunto de 4 jarros

A sequência de estados e suas mudanças correspondentes podem ser vistas na Figura 4.5

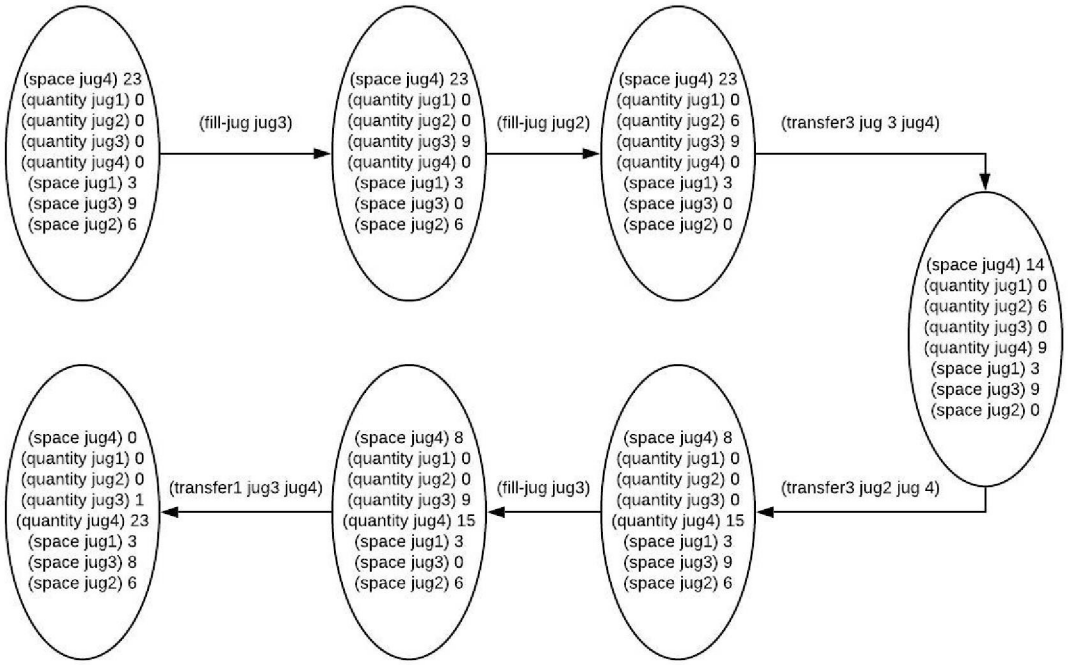


Figura 4.5: Sequência de estados solução gerados para a instância com um conjunto de 4 jarros

4.3.3 Análises de Desempenho

Gregory e colegas, em seu artigo "*Planning Modulo Theories: Extending the Planning Paradigm*"[GLFB12], apresentaram algumas análises de desempenho para PMTPlan. Neste trabalho traz-se dois desses problemas de planejamento, o problema dos "Caminhões de Entrega"e o problema dos "Contadores de Histórias", para a análise e comparação com o planejador MetricFF [Hof03], o qual é considerado um bom planejador para problemas com domínios numéricos.

O primeiro problema apresenta resultados para o domínio dos caminhões de entrega, em inglês *Dump-trucks*. Nesse problema, o objetivo é entregar todos os pacotes em menos passos. Para isso, os autores consideraram um conjunto de dois caminhões, dois locais e a quantidade de pacotes a entregar é variável. Pode-se ver os resultados na Tabela 4.1, no qual o PMTPlan lida muito bem com a expansão dos nós.

Tabela 4.1: Análise de desempenho para o problema Caminhões de Entrega [GLFB12].

	PMTPlan		**MetricFF**	
Pacotes	**Nós**	**Tempo**	**Nós**	**Tempo**
10	1247	11.44	54658	1.07
12	1855	16.78	84759	2.83
15	10933	52.60	271705	47.90
17	20247	156.83	551430	215.14
20	49414	1095.77		

O segundo experimento que os autores apresentaram foi com o problema chamado domínio dos contadores de histórias, em inglês *Storrytellers Domain*. Para essa comparação considera-se dois objetivos diferentes: saturação e igualdade. Ambas as metas exigem que o público ouça pelo menos metade das histórias. Os problemas de saturação exigem que todas as histórias tenham sido ouvidas por pelo menos um dos públicos, enquanto os problemas de igualdade exigem que todos os públicos ouçam as mesmas histórias.

Para esse problema foi considerado dois públicos e cinco contadores de histórias. O número de histórias foi variado. As tabelas 4.2 e 4.3 mostram o desempenho dos planejadores para os objetivos de saturação e igualdade, respectivamente.

Tabela 4.2: Análise de desempenho para o problema Contadores de Histórias - Saturação [GLFB12].

	PMTPlan		**MetricFF**	
Histórias	**Nós**	**Tempo**	**Nós**	**Tempo**
10	20	0.67	18	0.00
12	11	0.57	26	0.01
14	8	0.60	33	0.01
16	14	0.68	59	0.02
18	28	0.68	32	0.00
20	22	0.68	105	0.02
50	23	0.83	46872	6.48
60	34	0.99	701345	208.40
70	38	0.98		
100	25	0.82		

Tabela 4.3: Análise de desempenho para o problema Contadores de Histórias - Igualdade [GLFB12].

	PMTPlan		**MetricFF**	
Histórias	**Nós**	**Tempo**	**Nós**	**Tempo**
5	13	0.71	14	0.01
6	12	0.64	14	0.13
7	4	0.55	33	2.56
8	8	0.70	24	48.04
9	4	0.62		
10	4	0.57		
14	11	0.72		
18	4	0.61		
20	4	0.59		

O MetricFF tem uma melhor execução com as instâncias de saturação do que nas instâncias de igualdade porque evita lidar explicitamente com conjuntos. O público acumula histórias monotonicamente, então o problema só diz respeito ao simples contador para saber se todas as histórias foram ouvidas por todas os públicos. Ao raciocinar sobre se dois públicos ouviram as mesmas histórias, é impossível evitar o raciocínio explícito sobre conjuntos.

Os dados mostram que a abordagem do PDDL é inviável para instâncias maiores, enquanto o PMTPlan funciona igualmente bem nesses problemas, saturação e igualdade (os planos são muito semelhantes para todas essas instâncias do problema porque a codificação do problema lida diretamente com os conjuntos de histórias).

4.4 O PROBLEMA TEMPORAL DOS JARROS: UM NOVO DOMÍNIO PMT

Nesta seção, apresenta-se um novo domínio para mostrar a eficiência do PMT em lidar com problemas que vão além dos domínios proposicionais. Em especifico esse domínio lidará simultaneamente com fatores de tempo e recursos. Como não foi encontrado domínios na literatura que tratem dessa composição, propõe-se um novo domínio com base em um já existente.

Esse domínio que propõem-se é baseado no conhecido problema dos jarros. A proposta é acrescentar uma nova dimensão ao domínio que trata apenas a questão de recursos. Assim, esse problema terá duas dimensões: recursos e tempo. Deste modo, o objetivo desse problema será encher os jarros da melhor forma possível com o menor tempo. Denomina-se este problema de Problema Temporal dos Jarros.

Assume-se o exemplo da Seção 4.2 para demonstrar as mudanças na modelagem do problema. Também, admite-se que cada litro transferido, colocado ou retirado de cada jarro leva 1 segundo para acontecer.

Seja o arquivo de instância do problema. Ao novo cabeçalho adiciona-se um novo objeto, act1, para garantir que cada ação execute e termine. Isso impede que ações sejam executadas aos mesmo tempo.

```
(define (problem jugs)
(:domain jugpour)
(:objects
jug1 jug2 jug3 jug4 - jug
 act1 - act
)
```

Ao estado inicial define-se o objeto act1 com o comando (assign (free-act act1) true). Esse comando indica que o objeto act1 está livre. O restante permanece igual.

```
(:init
(assign (quantity jug1) 0)
(assign (quantity jug2) 0)
(assign (quantity jug3) 0)
(assign (quantity jug4) 0)
(assign (space jug1) 3)
(assign (space jug2) 6)
(assign (space jug3) 9)
(assign (space jug4) 23)
(assign (capacity jug1) 3)
(assign (capacity jug2) 6)
(assign (capacity jug3) 9)
(assign (capacity jug4) 23)
(assign (free-act act1) true)
)
```

A descrição do estado objetivo permanece igual ao conhecido problema dos jarros em CDDL, como visto na Seção 4.2.

Para a modelagem do domínio do problema acrescenta-se o módulo temporal e a função (free-act ?a - act) - boolean no cabeçalho. Essa função indica se existe ação sendo executada ou não.

```
(define (domain jugpour)
(:types
    jug
    act
)
(:modules integer set temporal )
(:functions
    (quantity  ?t - jug) - integer
    (space     ?t - jug) - integer
    (capacity  ?t - jug) - integer
    (free-act  ?a - act) - boolean
)
```

Como exemplo de descrição de uma ação, o trecho abaixo descreve a ação de encher um jarro. Veja a modelagem completa do Problema Temporal dos Jarros no apêndice B.

```
(:action fill-jug
    :parameters(
       ?t - jug
       ?a - act
    )
    :precondition(
        (duration-of (this) (capacity ?t))
        (< (quantity ?t) (capacity ?t))
        (at-start (free-act ?a))
    )
    :effect(
        (at-start (assign (free-act ?a) false))
        (assign (quantity ?t) (capacity ?t))
        (assign (space ?t) 0)
        (at-end (assign (free-act ?a) true))
    )
)
```

Note a criação do parâmetro a. Esse parâmetro é necessário para controlar a duração de cada ação.

Para calcular o tempo é necessário que o objeto a se inicie livre. Assim, o estabelece como não-livre. Deste modo, a duração para encher o jarro é a capacidade do mesmo. Após o término da ação, marca-se o objeto a como livre, para que outra ação possa ser iniciada.

4.5 CONCLUSÃO

Neste capítulo apresentamos a descrição de uma nova abordagem para o planejamento, chamado de Planejamento Módulo Teorias (PMT). Essa nova abordagem surgiu a partir da ideia do SMT em criar módulos para resolução dos problemas e da dificuldade de resolver problemas mais reais sem a necessidade da criação de um algoritmo para cada tipo de problema de planejamento. Mostramos o planejador proposto para lidar com essas questões e assim, pudemos perceber seu melhor desempenho em lidar com problemas em que o fator recurso é importante. No próximo capítulo apresentaremos outros experimentos e comparações com o uso de recursos e apresentaremos os resultados realizados com o novo domínio criado neste capítulo.

5 EXPERIMENTOS E RESULTADOS

Vimos no capítulo 4 que o planejador PMTPlan lida muito bem com problemas de planejamento com recursos.

Neste capítulo apresenta-se alguns experimentos e resultados com o uso do mesmo planejador. Na seção 5.1 apresenta-se experimentos feitos com o domínio dos jarros apenas utilizando. Nesta seção, também apresenta-se os experimentos feitos com o domínio de alocação de alunos para professores.

Em ambos os testes é feito uma comparação dos resultados do PMTPlan com o planejador MetricFF. Este planejador foi escolhido por ser considerado um bom planejador para problemas com domínios numéricos, além da agilidade motivada pelos domínios existentes, visto que alguns já estavam disponíveis tanto em PDDL quanto em CDDL.

A seção 5.2 apresenta os experimentos feitos com o domínio proposto na seção 4.4, o qual utiliza dois fatores para o planejamento: recursos e tempo.

Para a realização dos experimentos foi utilizado uma máquina com 8GiB de memória e processador Intel Core i3.

5.1 EXPERIMENTOS COM DOMÍNIO DE RECURSOS

O primeiro experimento é com o domínio do Problema dos Jarros, este já definido na seção 4.2.

A Tabela 5.1 apresenta os resultados para esse problema e está disposta em dois blocos principais identificando o planejador usado. A primeira coluna de cada bloco apresenta a quantidade de nós criados durante a busca pelo plano, a segunda o tamanho do plano encontrado e a terceira tempo total, em segundos, de execução. Cada linha dessa tabela representa os dados obtidos para instâncias com diferentes quantidades de jarros. A escolha do número de jarros para cada instância do problema se baseou nos domínios já existente.

Observa-se que para as instâncias testadas o PMTPlan apresentou resultados inferiores quando comparado ao MetricFF nas colunas 'Nós' e 'Tamanho do Plano'. Também pode-se notar que os resultados para a instância com 8 jarros foram discrepantes, possivelmente aconteceu pela forma como o planejador começa a busca, visto que as instâncias do problema seguem um padrão, o que justifica o resultado maior para a instância com 2 jarros.Descobrir com segurança o motivo dessa anomalia não será o foco desse trabalho, assim fica-se aberto para estudo futuro.

Nota-se que as linhas abaixo da instância com 16 jarros não apresentam resultados para o MetricFF, isso se deve ao fato de que esse planejador ultrapassou a quantidade de memória da máquina utilizada ao executar a busca pelo melhor plano.

Também, observa-se que na questão de tempo total da execução o MetricFF teve uma melhor performance, apresentando tempos menores em comparação ao PMTPlan.

Apesar do MetricFF ser mais rápido, ele não consegue resolver os problemas que foram resolvidos pelo PMTPlan. Isto mostra que o PMTPlan usa menos memória e consegue executar testes maiores, além de encontrar planos consideravelmente menores em relação ao MetricFF.

A Figura 5.6 apresenta um gráfico para a comparação do tempo total de execução entre os dois algoritmos. Para melhor análise, foram considerados resultados até a instância com 16 jarros.

Tabela 5.1: Análise de desempenho para o problema dos jarros.

	PMTPlan			MetricFF		
Nº de Jarros	**Nós**	**Tamanho do Plano**	**Tempo (seg)**	**Nós**	**Tamanho do Plano**	**Tempo (seg)**
2	17	16	1.37	18	17	0.00
4	16	6	3.70	136	17	0.00
6	24	6	7.88	582	17	0.01
8	1044	8	298.25	2516	17	0.10
10	27	6	26.05	10564	17	0.64
12	25	6	33.15	28740	17	2.73
14	26	6	60.53	73558	17	9.76
16	31	6	95.09	186206	17	35.21
18	47	6	286.23	–	–	–
20	50	6	418.63	–	–	–
22	36	6	408.08	–	–	–
24	60	6	824.07	–	–	–
26	42	6	768.76	–	–	–
28	63	6	1527.27	–	–	–
30	47	6	1645.70	–	–	–

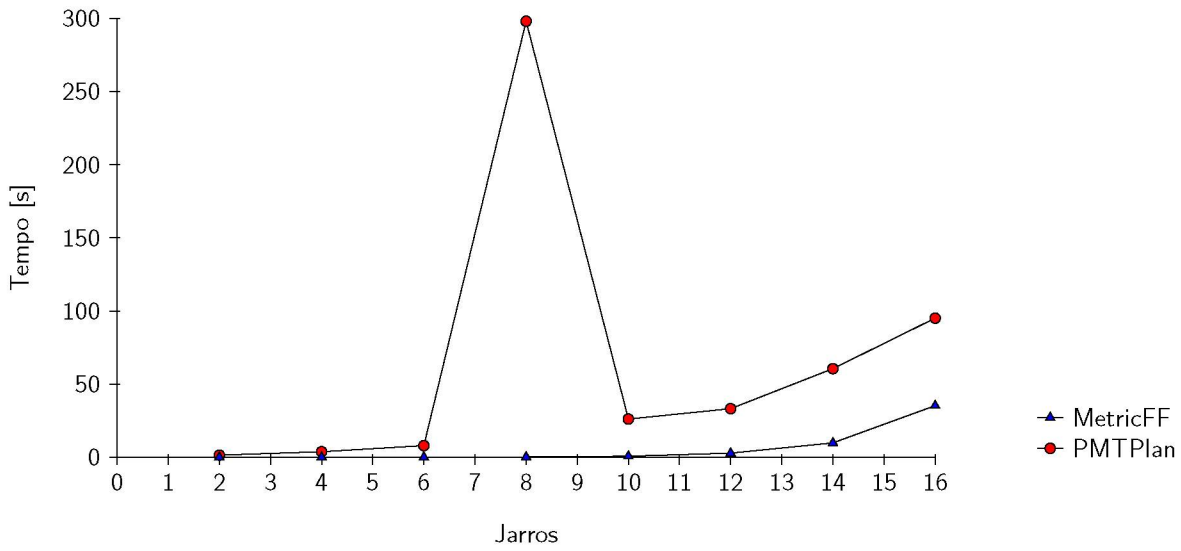


Figura 5.1: Gráfico de tempo para o domínio dos Jarros.

No gráfico, nota-se que o tempo de execução do PMTPlan encontra-se mais elevado, embora a diferença seja pequena. Novamente, no gráfico fica visível o valor discrepante, como explicado anteriormente, para a instância com 8 jarros.

Para os próximos experimentos com o uso de recurso foi utilizado o domínio chamado Professor Viajante, em inglês *travelling-teacher*. Esse domínio envolve um grupo de professores que sabem um conjunto de matérias, o objetivo é que cada aluno aprenda as mesmas matérias e que o total de matérias aprendidas seja maior que a metade das matérias disponíveis. Esse domínio explora a interação dos módulos *integer* e *set*.

A Tabela 5.2 apresenta os resultados obtidos para instâncias com um conjunto de 5 professores e 2 alunos e está disposta em dois blocos principais identificando o planejador usado.

A primeira coluna de cada bloco apresenta a quantidade de nós criados durante a busca pelo plano, a segunda o tamanho do plano encontrado e a terceira tempo total, em segundos, de execução. Cada linha dessa tabela representa os dados obtidos para instâncias com diferentes quantidades de matérias, as quais variam de 2 até 20. A escolha desses números foi baseada nos domínios existentes.

Tabela 5.2: Análise de desempenho para o problema dos professores para instâncias com 5 professores.

	PMTPlan			MetricFF		
Matérias	**Nós**	**Tamanho do Plano**	**Tempo (seg)**	**Nós**	**Tamanho do Plano**	**Tempo (seg)**
2	6	2	0.46	12	11	0.00
3	4	2	0.44	13	10	0.00
4	4	2	0.44	20	14	0.00
5	4	2	0.47	14	12	0.00
6	4	2	0.46	14	12	0.08
7	4	2	0.43	33	18	1.93
8	6	2	0.52	24	16	84.42
9	4	2	0.48	30	21	1999.26
10	4	2	0.46	–	–	–
11	4	2	0.44	–	–	–
12	4	2	0.51	–	–	–
13	7	2	0.59	–	–	–
14	3	2	0.46	–	–	–
15	4	2	0.48	–	–	–
16	4	2	0.45	–	–	–
17	5	2	0.51	–	–	–
18	6	2	0.52	–	–	–
19	3	2	0.40	–	–	–
20	5	2	0.50	–	–	–

Observa-se que, para estas instâncias do Problema dos Professores, o PMTPlan apresentou melhores resultados em relação ao MetricFF nos quesitos expansão dos nós durante a busca e no tamanho do plano encontrado. Nota-se, porém, que na coluna 'tempo' o PMTPlan também foi melhor, embora nas cinco primeiras instâncias da tabela o MetricFF tenha se mostrado mais rápido. Percebe-se, também, que para as instâncias com 8 e 9 matérias a diferença de tempo de execução do PMTPlan com o MetricFF foi discrepante. Esse fenômeno ocorre pelo fato de que o PMTPlan trata os conjuntos diretamente, enquanto o MetricFF trata cada atributo de forma isolada. Isto provoca com que as descrições de ações em PDDL tenham mais parâmetros, o que ocasiona um maior uso de memória da máquina e o leve a não conseguir executar os restantes dos experimentos, além de aumentar o tempo de execução para cada instância.

Um evento interessante é que o número de passos do plano solução é 2 para todos os exemplos, enquanto para o MetricFF o plano vai aumentando, além de ser muito maior que o do encontrado pelo PMTPlan. Isso se deve a modelagem do problema. Para esse conjunto de instâncias existem professores que ensinam um número maior que a metade de matérias disponíveis, então, o planejador resolve com apenas 2 passos: o primeiro para ensinar um aluno; o segundo passo para ensinar o outro aluno.

Por exemplo, para a instância com 10 matérias, temos a seguinte modelagem do problema:

```
(define (problem tt-5-10)
(:domain travelling-teacher)
(:objects
        t1 t2 t3 t4 t5 - teacher
        s1 s2 - student
        sk1 sk2 sk3 sk4 sk5 sk6 sk7 sk8 sk9 sk10 - skill
)
(:init
        (assign (skills t1) (construct-set sk1 sk3 ))
        (assign (skills t2) (construct-set sk8 sk9 sk1 ))
        (assign (skills t3) (construct-set sk3 sk3 sk5 sk7
                                           sk8 sk2 sk9 sk10 ))
        (assign (skills t4) (construct-set sk2 sk4 sk9 sk10
                                           sk2 sk4 sk7 sk8 ))
        (assign (skills t5) (construct-set sk5 sk6 sk7 sk1 sk6
                                           sk10 sk4 sk5 sk6 ))
        (assign (learnt s1) (empty-set))
        (assign (learnt s2) (empty-set))
        (assign (skill-set) (construct-set sk1 sk2 sk3 sk4 sk5
                                           sk6 sk7 sk8 sk9 sk10))
)
(:goal
        (= (learnt s1) (learnt s2))
        (> (cardinality (learnt s1)) (/ (cardinality (skill-set)) 2))
        (> (cardinality (learnt s2)) (/ (cardinality (skill-set)) 2))
)
)
```

E como saída do planejador, temos as seguintes ações:

```
(teach t5 s2)
(teach t5 s1)
```

Além do mais, a modelagem do domínio em CDDL é muito mais limpa e curta que a modelagem feita em PDDL, por exemplo, para esse problema, em CDDL temos apenas uma ação, enquanto a em PDDL contém quatro ações. Isso se deve ao fato de que no PMT pode-se usar as funções de conjuntos, o que facilita a modelagem e a execução do planejador.

A Figura 5.2 apresenta um gráfico para a comparação do tempo total de execução entre o PMTPlan e o MetricFF. Para melhor análise, foram considerados resultados até a instância com 9 matérias. Para o MetricFF, foram considerados resultados até a instância com 8 matérias, pelo fato de que seu resultado é discrepante e irrelevante para essa comparação, visto que com estes já é possível perceber que o PMTPlan teve uma performance melhor.

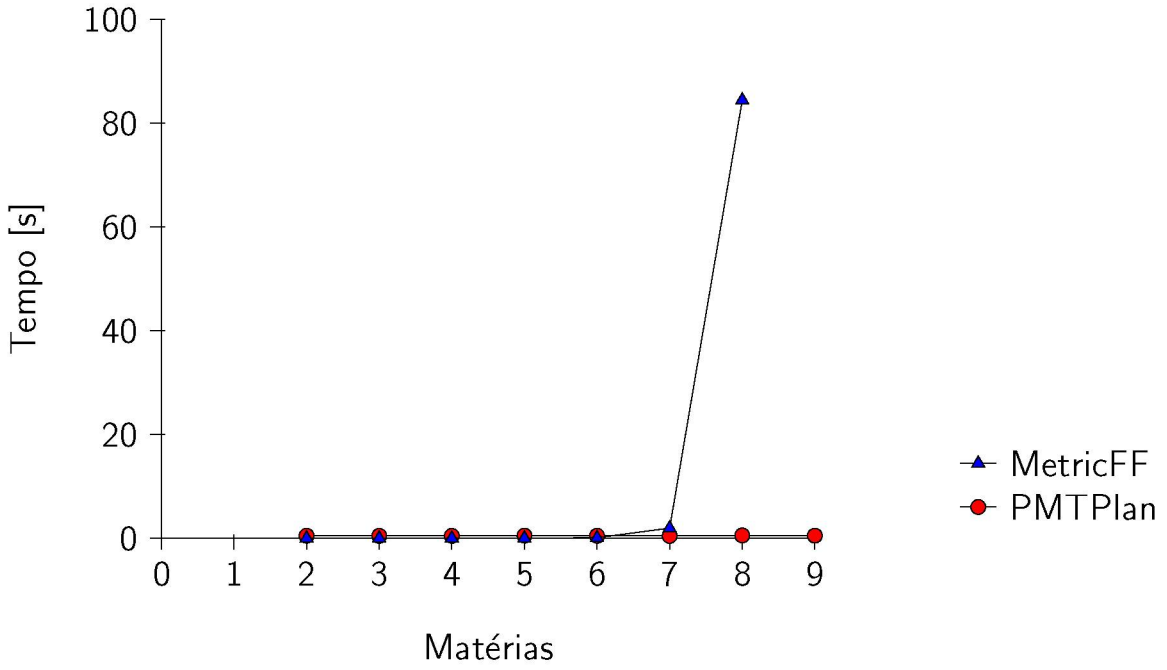


Figura 5.2: Gráfico de tempo para o domínio dos Professores para instâncias com um conjunto de 5 professores.

A Tabela 5.3 exibe os resultados obtidos para instâncias com um conjunto de 6 professores e 2 alunos e está disposta em dois blocos principais identificando o planejador usado. A primeira coluna de cada bloco apresenta a quantidade de nós criados durante a busca pelo plano, a segunda o tamanho do plano encontrado e a terceira tempo total, em segundos, de execução. Cada linha dessa tabela representa os dados obtidos para instâncias com diferentes quantidades de matérias, as quais variam de 2 até 20. A escolha desses números foi baseada nos domínios existentes.

Observa-se que, para estas instâncias do Problema dos Professores, o PMTPlan apresentou uma melhor performance em relação ao MetricFF. Embora nas cinco primeiras instâncias da tabela o MetricFF tenha se mostrado mais rápido, para os demais experimentos ele se mostrou mais lento. Percebe-se, também, que para as instâncias com 8 e 9 matérias a diferença de tempo de execução entre os dois planejadores foi discrepante. Isto ocorre por razão de que o PMTPlan trata os conjuntos diretamente, o que provoca com que um problema CDDL tenha menos de ações e parâmetros do que um problema em PDDL. Desta forma, o PMTPlan utiliza um menor uso de memória da máquina, execute os restantes dos experimentos e leve um tempo de execução menor para cada instância.

Para esse conjunto de teste, novamente temos o mesmo tamanho de plano para todos os exemplos. Vale o mesmo argumento para os resultados obtidos na Tabela 5.2.

A Figura 5.3 apresenta um gráfico para a comparação do tempo total de execução entre o PMTPlan e o MetricFF. Para melhor análise, foram considerados resultados até a instância com 9 matérias. Para o MetricFF, foram considerados resultados até a instância com 8 matérias, já que seu resultado para 9 matérias é discrepante e irrelevante para essa comparação, uma vez que com estes já é possível perceber que o PMTPlan teve uma performance melhor.

Tabela 5.3: Análise de desempenho para o problema dos professores para instâncias com 6 professores.

	PMTPlan			MetricFF		
Matérias	**Nós**	**Tamanho do Plano**	**Tempo (seg)**	**Nós**	**Tamanho do Plano**	**Tempo (seg)**
2	5	2	0.42	11	10	0.00
3	6	2	0.50	11	9	0.00
4	9	2	0.55	16	13	0.00
5	5	2	0.49	13	11	0.00
6	3	2	0.43	17	13	0.08
7	5	2	0.53	13	12	1.84
8	7	2	0.60	43	23	72.69
9	6	2	0.54	26	14	2001.34
10	8	2	0.64	–	–	–
11	6	2	0.63	–	–	–
12	7	2	0.64	–	–	–
13	8	2	0.64	–	–	–
14	9	2	0.72	–	–	–
15	8	2	0.66	–	–	–
16	6	2	0.63	–	–	–
17	6	2	0.62	–	–	–
18	7	2	0.63	–	–	–
19	6	2	0.60	–	–	–
20	8	2	0.64	–	–	–

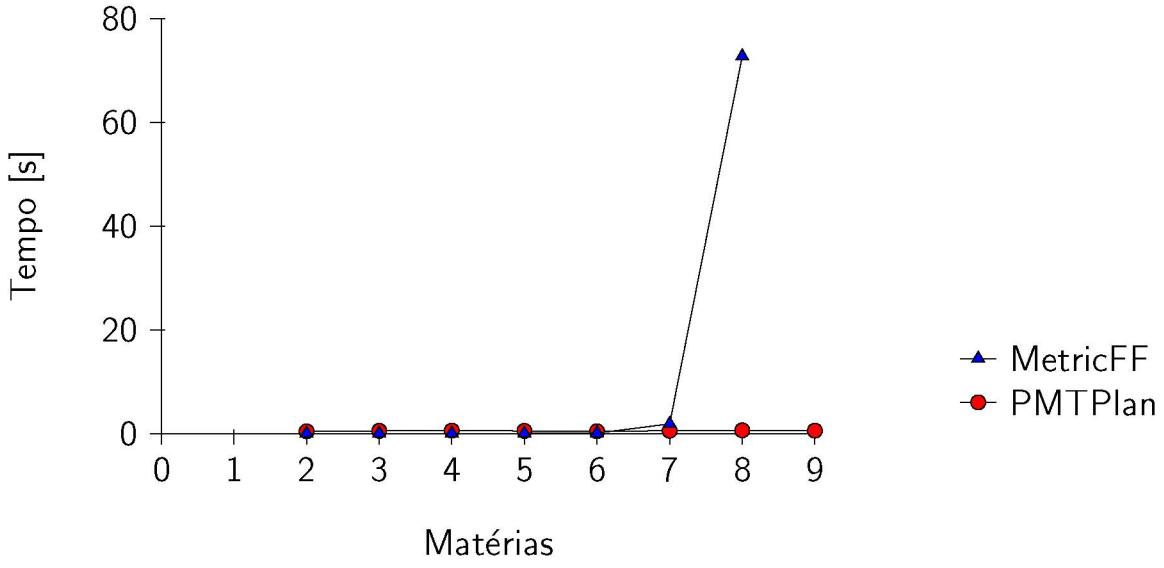


Figura 5.3: Gráfico de tempo para o domínio dos Professores para instâncias com um conjunto de 6 professores.

A Tabela 5.4 apresenta os resultados obtidos para instâncias com um conjunto de 7 professores e 2 alunos e está disposta em dois blocos principais identificando o planejador usado. A primeira coluna de cada bloco apresenta a quantidade de nós criados durante a busca pelo plano, a segunda o tamanho do plano encontrado e a terceira tempo total, em segundos, de

execução. Cada linha dessa tabela representa os dados obtidos para instâncias com diferentes quantidades de matérias, as quais variam de 2 até 20. A escolha desses números foi baseada nos domínios existentes.

Tabela 5.4: Análise de desempenho para o problema dos professores para instâncias com 7 professores.

	PMTPlan			MetricFF		
Problema	**Nós**	**Tamanho do Plano**	**Tempo (seg)**	**Nós**	**Tamanho do Plano**	**Tempo (seg)**
2	4	2	0.39	14	11	0.00
3	3	2	0.40	18	13	0.00
4	7	4	0.55	14	13	0.00
5	8	2	0.55	15	12	0.00
6	5	2	0.50	31	18	0.09
7	3	2	0.51	36	18	2.09
8	5	2	0.56	23	18	70.68
9	5	2	0.53	40	20	1986.22
10	8	2	0.62	–	–	–
11	4	2	0.56	–	–	–
12	7	2	0.73	–	–	–
13	4	2	0.56	–	–	–
14	4	2	0.54	–	–	–
15	4	2	0.56	–	–	–
16	19	4	0.95	–	–	–
17	10	4	0.84	–	–	–
18	11	4	0.80	–	–	–
19	8	4	0.78	–	–	–
20	11	4	0.82	–	–	–

Observa-se que, para estas instâncias do Problema dos Professores, o PMTPlan, novamente apresentou uma melhor performance em relação ao MetricFF. Ainda que nas cinco primeiras instâncias experimentadas o MetricFF tenha se mostrado mais ágil, para o restante ele se mostrou mais lento. Nota-se, também, que para as instâncias com 8 e 9 matérias a diferença de tempo de execução entre os dois planejadores foi discrepante. Isto acontece pelo motivo de que o PMTPlan trata os conjuntos diretamente, em contra partida o MetricFF trata cada atributo de forma isolada. Desta maneira, isto provoca com que as descrições de ações em PDDL tenham mais parâmetros, o que ocasiona um maior uso de memória da máquina, o leva a não conseguir executar os restantes dos experimentos e aumenta o tempo de execução para cada instância.

Para esse conjunto de testes, obteve-se resultados para o tamanho de plano diferente de 2. Pode-se encontrar planos de tamanho 2 e de tamanho 4. Quando algum professor ensina um conjunto que contém mais do que a metade da quantidade de matérias, pode-se resolver o problema em apenas dois passos. Como visto na Tabela 5.2. Porém, se isso não acontece, como nas instâncias com 4, 16, 17, 18, 19 e 20 matérias, mais passos serão necessários para resolver o problema.

Por exemplo, para a instância com 4 matérias:

```
(define (problem tt-7-4)
(:domain travelling-teacher)
(:objects
     t1 t2 t3 t4 t5 t6 t7 - teacher
     s1 s2 - student
     sk1 sk2 sk3 sk4 - skill
)
(:init
        (assign (skills t1) (construct-set sk4 sk3 ))
        (assign (skills t2) (construct-set sk3 sk4 ))
        (assign (skills t3) (construct-set ))
        (assign (skills t4) (construct-set sk1 ))
        (assign (skills t5) (construct-set sk1 sk2 ))
        (assign (skills t6) (construct-set sk3 sk4 ))
        (assign (skills t7) (construct-set sk1 sk2 sk2 ))
        (assign (learnt s1) (empty-set))
        (assign (learnt s2) (empty-set))
        (assign (skill-set) (construct-set sk1 sk2 sk3 sk4))
)
(:goal
        (= (learnt s1) (learnt s2))
        (> (cardinality (learnt s1))
            (/ (cardinality (skill-set)) 2))
        (> (cardinality (learnt s2))
            (/ (cardinality (skill-set)) 2))
    )
)
```

Como saída do planejador temos a seguinte sequência de ações:

```
(teach t2 s2)
(teach t1 s1)
(teach t4 s2)
(teach t4 s1)
```

A Figura 5.4 apresenta um gráfico para a comparação do tempo total de execução entre o PMTPlan e o MetricFF. Para uma melhor análise, foram considerados resultados até a instância com 9 matérias. Para o MetricFF, foram considerados resultados até a instância com 8 matérias, já que seu resultado para 9 matérias é discrepante e irrelevante para essa comparação, dado que com estes já é possível perceber que o PMTPlan teve uma performance melhor.

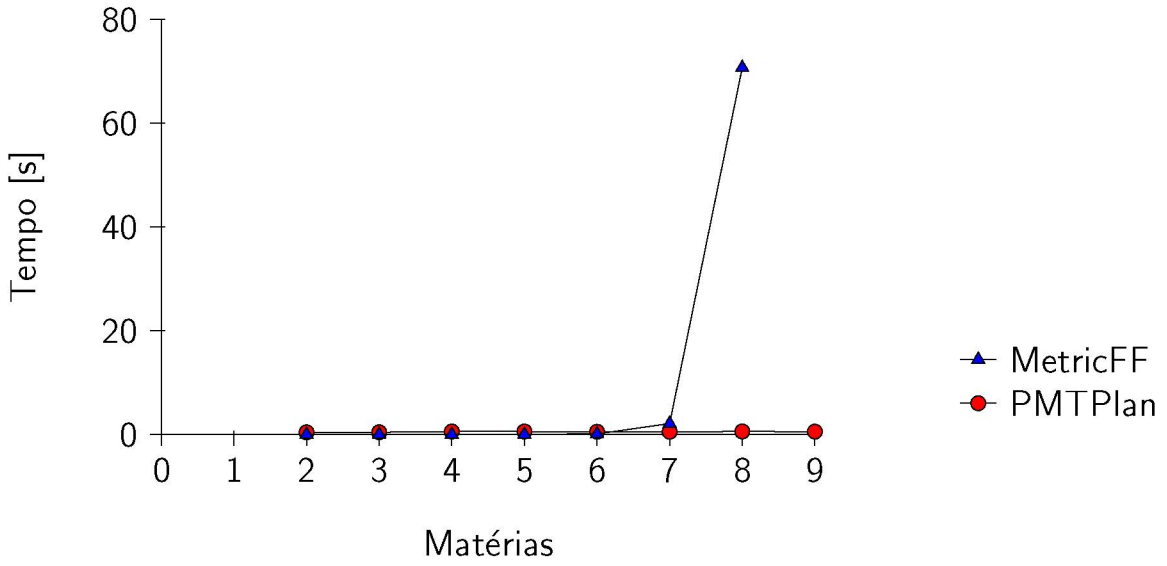


Figura 5.4: Gráfico de tempo para o domínio dos Professores para instâncias com um conjunto de 6 professores.

5.2 EXPERIMENTOS COM DOMÍNIO DE TEMPO E RECURSOS

Nesta seção mostra-se os experimentos feitos utilizando o domínio modelado pela autora dessa dissertação e que foi visto na Seção 4.4. Ao limite do nosso conhecimento não foi encontrado outro planejador para fazer a comparação da eficiência com esse novo desafio.

A Tabela 5.5 traz os resultados obtidos com esses experimentos e está preparada da seguinte forma: a primeira coluna apresenta a quantidade de jarros usada em cada instância, seguindo o padrão do experimento anterior a quantidade de jarros varia de 2 até 18, buscando quantidades pares. A segunda coluna apresenta a quantidade de nós criados durante a busca e escalonamento do plano, a terceira o tamanho do plano encontrado e a quarta o tempo total, em segundos, de execução.

Tabela 5.5: Análise de desempenho para o problema dos jarros com noções temporais.

	PMTPlan		
Nº de Jarros	**Nós**	**Tamanho do Plano**	**Tempo (seg)**
2	78	16	4.03
4	23	5	6.58
6	51	5	19.05
8	29	5	22.31
10	48	5	63.64
12	46	5	96.78
14	83	5	381.64
16	57	5	416.03
18	80	5	882.03

Visto que esse domínio baseou-se em um existente, no qual acrescentou-se noções temporais, pode-se comparar esses dois domínios.

Observa-se que comparado com a Tabela 5.1 houve um aumento significativo no número de nós criados durante a busca e pode ser visto na Figura 5.5. Isso acontece devido ao uso do fator tempo, então, o planejador precisa que as ações demarquem tempo de início e tempo de finalização. Assim cada ação é transformada em duas: uma demarcando seu início e outra demarcando o seu fim. Ao final da execução da busca as ações voltam ao normal para o escalonamento de tempo. Desta maneira, com o aumento da criação de nós, o tempo total de execução também aumenta e pode ser visto na Figura 5.6.

A Figura 5.5 apresenta o gráfico de comparação de expansão de nós para os dois domínios: Problema dos Jarros e Problema Temporal dos Jarros. Os resultados para a instância de 8 jarros foram desconsiderados para esse gráfico, visto que, para o domínio que utiliza recursos, o resultado foi discrepante, o que deixa o gráfico ilegível para análise.

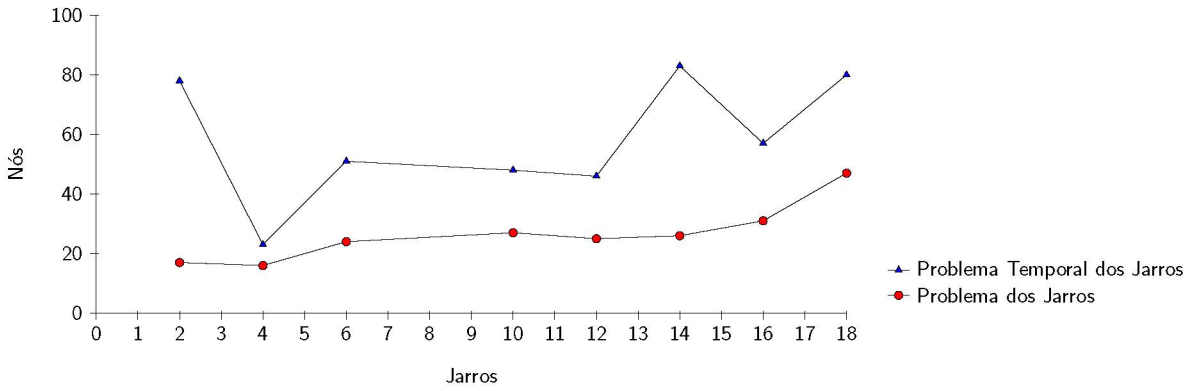


Figura 5.5: Gráfico de nós expandidos para o domínio dos jarros e o domínio temporal dos jarros.

Pode-se perceber que a quantidade de nós expandidos de um domínio para o outro aumenta de forma parecida.

A Figura 5.6 apresenta o gráfico de comparação de tempo total de execução para os dois domínios: Problema dos Jarros e Problema Temporal dos Jarros. Os resultados acima de 200 segundos foram cortados do gráfico, afim de deixá-lo legível para a comparação.

Da mesma maneira que analisado na Figura 5.5, observa-se que o tempo de execução dos dois domínios cresce de forma semelhante.

Dessa forma, com esses experimentos assegura-se que o PMTPlan é capaz de lidar com esse tipo de desafio, onde recursos e tempo são usados no mesmo problema, além de encontrar bons planos para a resolução das instâncias do problema dos jarros com noções temporais, baseado nas métricas de de expansão de nós, tamanho do plano e tempo de execução.

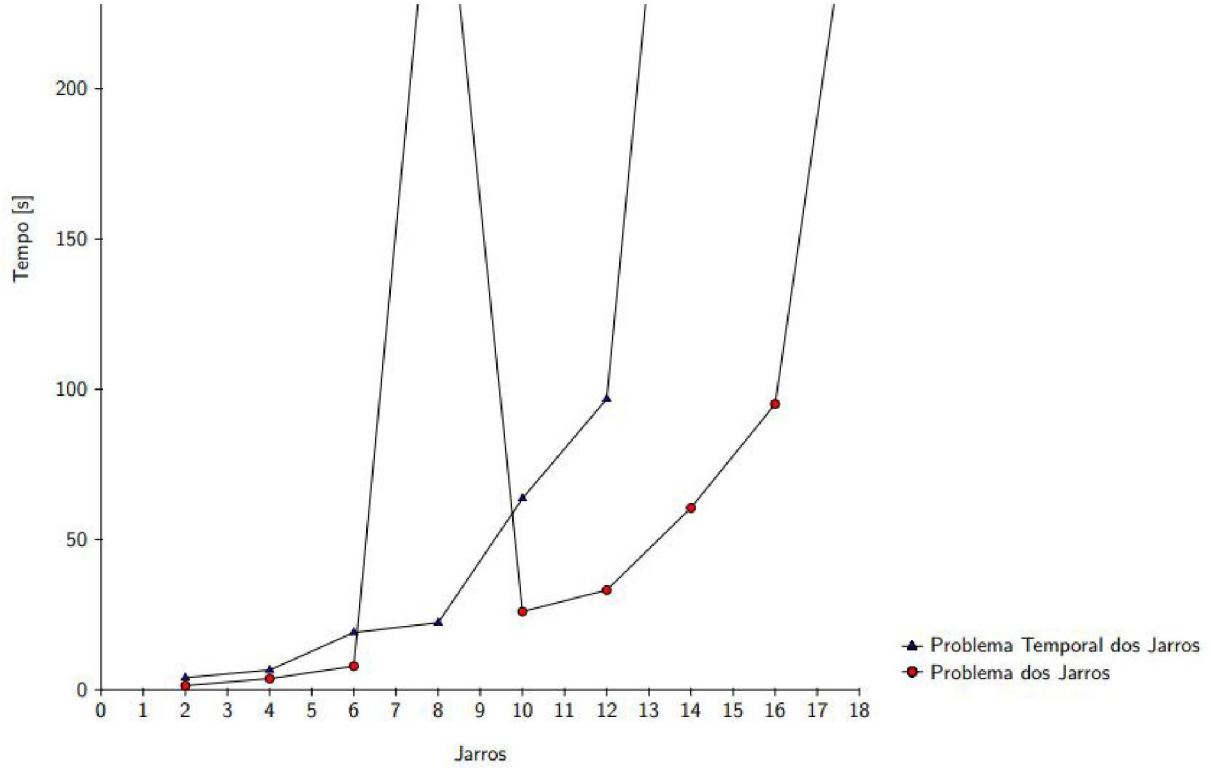


Figura 5.6: Gráfico de tempo total de execução para o domínio dos jarros e o domínio temporal dos jarros.

5.3 CONCLUSÃO

Neste capítulo foram apresentados os experimentos e resultados obtidos ao utilizar o planejador PMTPlan. Em comparação com outro planejador, o MetricFF, que é considerado um bom planejador para problemas com recursos, foi possível demonstrar que o PMTPlan, mesmo sendo um planejador em fase experimental, foi eficiente ao resolver essa classe de problemas. Também, com a utilização de um novo domínio, denominado Problema Temporal do Jarros, foi demonstrado o comportamento e a eficiência do PMTPlan em lidar com problemas em que o uso de tempo e recursos são empregados simultaneamente.

6 CONCLUSÃO

Neste trabalho foi estudado um novo formalismo para o planejamento, o qual é chamado Planejamento Módulo Teorias. Foi mostrado um planejador que lida diretamente com esse formalismo, o PMTPlan, que foi proposto to por Gregory, Derek, MariaFox e Beck [GLFB12]. Também foi desenvolvido e apresentado um novo domínio para experimentos nesse planejador, o qual utiliza duas noções de planejamento não clássico: tempo e recursos.

Para o devido estudo, foi introduzido de maneira geral o conceito de problema de planejamento e planejamento clássico. Esse planejamento resolve problemas restritos e limitados, no qual foi fornecido algumas referências para possível aprofundamento no assunto, já que o foco deste trabalho não foi trabalhar com o esse tópico.

Por meio das restrições apresentadas no problema de planejamento clássico, pode-se chegar ao planejamento não clássico, que ocorre no momento em que uma ou mais dessas restrições são violadas. Focou-se em dois tipos específicos de planejamento não clássico: o com recursos e com tempo.

A partir dessas informações, chegou-se a conclusão que para resolver cada tipo de problema é necessário desenvolver e utilizar planejadores mais específicos. Isso, então, justifica o estudo em PMT.

Dado que o PMT surgiu a partir da ideia do SMT, este último sento uma extensão do SAT, houve a necessidade de estudar e introduzir este assunto no trabalho. Assim, foi apresentado conceitos importantes sobre satisfazibilidade, oferecendo um breve histórico desse tema, para, então, alcançar a motivação do surgimento do SMT.

Este trabalho contém um bom texto em português a cerca do SMT, no qual definições, conceitos, trajetória, descrições sobre a resolução de fórmulas SMT e os principais métodos de resolução utilizados nos resolvedores modernos foram apresentados. Além de uma explicação breve sobre as teorias de interesse do SMT.

Após essa fundamentação teórica sobre as áreas que abrangem o PMT, pode-se estudar e apresentar essa nova abordagem. Lembrando que o PMT surgiu a partir da ideia do SMT, pode-se entender que o propósito principal desse planejamento é criar módulos para resolução dos problemas, o qual permite que o planejamento possa ser ampliado a um caminho modular, para resolver problemas que utilizam tempo e recursos simultaneamente. Sendo assim, o PMT trata de forma direta problemas não clássicos sem a necessidade de um algoritmo desenvolvido para tal. Com a criação do PMT também foi necessário criar uma nova linguagem para a descrição de cada problema e de cada módulo de teoria a ser utilizada. Com isso, foi apresentado com exemplo prático um domínio escrito nessa linguagem, explicando cada parte dos arquivo a serem utilizados.

Para os experimentos, foi utilizado um planejador que resolve diretamente problemas PMT, o PMTPlan, desenvolvido por Gregory e colegas para resolver esse tipo de problema. Como não foi identificado na literatura domínios para esse problema, foi proposto um novo domínio, utilizando como base o problema dos jarros, em que o uso de noções temporais foi adicionado ao problema. Denominou-se esse domínio como Problema Temporal dos Jarros.

A partir disso, foi demonstrado o comportamento e a eficiência do PMTPlan em lidar com problemas em que o uso de tempo e recursos são usados simultaneamente. Comparado com os resultados obtidos com o domínio do Problema dos Jarros, o qual utiliza apenas recursos, o domínio temporal apresentou resultados semelhantes, em que a quantidade de nós e tempo total de execução do algoritmo foram o dobro do primeiro experimento em questão. Esse resultado

deve-se ao fato de que para executar ações com noções temporais, o algoritmo desmembra tais ações em duas, designando para cada o momento de início e fim. Assim, observou-se que os resultados para esse problema foram satisfatórios.

Também foi demonstrada a eficiência do PMTPlan, mesmo sendo um planejador em fase experimental em comparação com outro planejador, o MetricFF, considerado um bom planejador para problemas numéricos, ao resolver problemas clássicos de planejamento com recursos. Nesses experimentos, observou-se que o PMTPlan tem uma boa performance em termos de memória utilizada, pois a expansão de nós no PMTPlan é menor em relação ao MetricFF, além de encontrar um plano solução menor. No quesito tempo, o PMTPlan se saiu melhor quando utilizado um domínio em que teorias de inteiros e conjuntos interagem. Isto se deve ao fato do PMTPlan lidar diretamente com conjuntos, fazendo com que o domínio tenha menos ações e parâmetros.

Além do mais, a modelagem de domínios em CDDL é muito mais limpa e curta do que a modelagem feita em PDDL. Isso se deve ao fato de que no PMT pode-se usar as funções de conjuntos, o que facilita a modelagem e a execução do planejador.

Além deste texto, deixamos como contribuição o artigo " *Analysis of Planning Performance Module Theories for a New Domain Using Resources and Time*"[IC19].

Em trabalhos futuros planeja-se estudar alguns pontos que ficaram em aberto neste trabalho, tais como: analisar o por quê dos resultados para os experimentos dos jarros foram discrepantes em alguns casos, encontrar outro planejador para comparar com o PMTPlan no caso do Problema Temporal dos Jarros. Também pretende-se aprimorar o planejador PMTPlan, visto que este está em fase experimental, acrescentando a ele métricas em relação a quantidades e, principalmente, tempo, dado que este planejador trata apenas métricas de minimização. Além disso, pretende-se criar novos domínios para o novo formalismo, no qual um deles foca no estudo de caso para problema de transporte ferroviário.

REFERÊNCIAS

[ABC+02] Gilles Audemard, Piergiorgio Bertoli, Alessandro Cimatti, Artur Korniłowicz, and Roberto Sebastiani. A sat based approach for solving formulas over boolean and linear mathematical propositions. In *International Conference on Automated Deduction*, pages 195–210. Springer, 2002.

[ACG99] Alessandro Armando, Claudio Castellini, and Enrico Giunchiglia. Sat-based procedures for temporal reasoning. In *European Conference on Planning*, pages 97–108. Springer, 1999.

[Ack54] Wilhelm Ackermann. Solvable cases of the decision problem. *North- Holland Publishing Co., Amsterdam*, pages 102–103, 1954.

[BBC+05] Marco Bozzano, Roberto Bruttomesso, Alessandro Cimatti, Tommi Junttila, Peter Van Rossum, Stephan Schulz, and Roberto Sebastiani. An incremental and layered procedure for the satisfiability of linear arithmetic logic. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 317–333. Springer, 2005.

[BBC+06] Marco Bozzano, Roberto Bruttomesso, Alessandro Cimatti, Tommi Junttila, Silvio Ranise, Peter van Rossum, and Roberto Sebastiani. Efficient theory combination via boolean search. *Information and Computation*, 204(10):1493–1525, 2006.

[BCD+11] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovi'c, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV '11)*, volume 6806 of *Lecture Notes in Computer Science*, pages 171–177. Springer, July 2011. Snowbird, Utah.

[BDS02] Clark W Barrett, David L Dill, and Aaron Stump. Checking satisfiability of first-order formulas by incremental translation to sat. In *International Conference on Computer Aided Verification*, pages 236–249. Springer, 2002.

[BF97] Avrim L Blum and Merrick L Furst. Fast planning through planning graph analysis. *Artificial intelligence*, 90(1-2):281–300, 1997.

[BFT16] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, acessado em 13/01/2018, 2016.

[BFT17] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The SMT-LIB Standard: Version 2.6. Technical report, Department of Computer Science, The University of Iowa, 2017. Available at `www.SMT-LIB.org`.

[BG98] Blai Bonnet and Héctor Geffner. Hsp: Heuristic search planner. *AIPS-98 Planning Competition*, 1998.

[BGV99] Randal E Bryant, S German, and Miroslav N Velev. Exploiting positive equality in a logic of equality with uninterpreted functions. *Computer Aided Verification*, 1999.

[BK00] Fahiem Bacchus and Froduald Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116(1-2):123–191, 2000.

[BL99] Avrim L Blum and John C Langford. Probabilistic planning in the graphplan framework. In *European Conference on Planning*, pages 319–332. Springer, 1999.

[BM85] Robert S Boyer and J Strother Moore. Integrating decision procedures into heuristic theorem provers: A case study of linear arithmetic. In *Machine intelligence*. Citeseer, 1985.

[Bru] Roberto Bruttomesso. Satisfiability modulo theories: Lezione 4 - the lazy approach. `https://pt.slideshare.net/robertobruttomesso/smtlecture4`, acessado em 13/01/2018.

[BSST09] Clark Barrett, Roberto Sebastiani, Sanjit Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 26, pages 825–885. IOS Press, February 2009.

[BT07] Clark Barrett and Cesare Tinelli. Cvc3. In *International Conference on Computer Aided Verification*, pages 298–302. Springer, 2007.

[BT17] Clark Barrett and Cesare Tinelli. Satisfiability modulo theories. In Ed Clarke, Thomas Henzinger, and Helmut Veith, editors, *Handbook of Model Checking*. 2017. In preparation.

[Byl94] Tom Bylander. The computational complexity of propositional strips planning. *Artificial Intelligence*, 69(1-2):165–204, 1994.

[CFLS08] Andrew Coles, Maria Fox, Derek Long, and Amanda Smith. A hybrid relaxed planning graph'lp heuristic for numeric planning domains. In *ICAPS*, pages 52–59, 2008.

[Cok13] David R. Cok. The SMT-LIBv2 Language and Tools: A Tutorial. `smtlib.github.io/jSMTLIB/SMTLIBTutorial.pdf`, acessado em 13/03/2018, 2013.

[Coo71] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

[CRT98] Alessandro Cimatti, Marco Roveri, and Paolo Traverso. Strong planning in non-deterministic domains via model checking. In *AIPS*, volume 98, pages 36–43, 1998.

[CT91] Ken Currie and Austin Tate. O-plan: the open planning architecture. *Artificial intelligence*, 52(1):49–86, 1991.

[DdM06] Bruno Dutertre and Leonardo de Moura. System description: Yices 1.0. *Proc. SMT-COMP*, 6, 2006.

[DEK+09] Christian Dornhege, Patrick Eyerich, Thomas Keller, Sebastian Trüg, Michael Brenner, and Bernhard Nebel. Semantic attachments for domain-independent planning systems. 2009.

[dic19] Dicionário Michaelis. ://michaelis.uol.com.br/, acessado em 15/10/2018, 2019.

[DLL62] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[DMB08] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[DMRS02] Leonardo De Moura, Harald Rueß, and Maria Sorea. Lemmas on demand for satisfiability solvers. *Proc. SAT*, 2:244–251, 2002.

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM (JACM)*, 7(3):201–215, 1960.

[DT94] Brian Drabble and Austin Tate. *The use of optimistic and pessimistic resource profiles to inform search in an activity based planner*. University of Edinburgh, Artificial Intelligence Applications Institute, 1994.

[EMW97] Michael D Ernst, Todd D Millstein, and Daniel S Weld. Automatic sat-compilation of planning problems. In *IJCAI*, volume 97, pages 1169–1176, 1997.

[ES03] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *International conference on theory and applications of satisfiability testing*, pages 502–518. Springer, 2003.

[FJOS03] Cormac Flanagan, Rajeev Joshi, Xinming Ou, and James B Saxe. Theorem proving using lazy proof explication. In *International Conference on Computer Aided Verification*, pages 355–367. Springer, 2003.

[FM92] Robert E Frederking and Nicola Muscettola. Temporal planning for transportation planning and scheduling. In *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, pages 1225–1230. IEEE, 1992.

[FN71] Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

[GBT09] Yeting Ge, Clark Barrett, and Cesare Tinelli. Solving quantified verification conditions using satisfiability modulo theories. *Annals of Mathematics and Artificial Intelligence*, 55(1-2):101–122, 2009.

[Gef00] Héctor Geffner. Functional strips: a more flexible language for planning and problem solving. In *Logic-based artificial intelligence*, pages 187–209. Springer, 2000.

[GH00] P Haslum H Geffner and P Haslum. Admissible heuristics for optimal planning. In *Proceedings of the 5th Internat. Conf. of AI Planning Systems (AIPS 2000)*, pages 140–149, 2000.

[GLFB12] Peter Gregory, Derek Long, Maria Fox, and J Christopher Beck. Planning Modulo Theories : Extending the Planning Paradigm. *International Conference on Automated Planning and Scheduling*, pages 65–73, 2012.

[GNT04] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: theory and practice*. Elsevier, 2004.

[Gre69] Cordell Green. Application of theorem proving to problem solving. In *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, IJCAI'69, pages 219–239, San Francisco, CA, USA, 1969. Morgan Kaufmann Publishers Inc.

[HN01] Jörg Hoffmann and Bernhard Nebel. The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[Hof03] Jörg Hoffmann. The metric-ff planning system: Translating"ignoring delete lists"to numeric state variables. *Journal of Artificial Intelligence Research*, 20:291–341, 2003.

[IC19] Danielle de Fátima Ivanchechen and Marcos Alexandre Castilho. Analysis of Planning Performance Module Theories for a New Domain Using Resources and Time. *ENIAC*, 2019.

[KMS96] Henry Kautz, David McAllester, and Bart Selman. Encoding plans in propositional logic. *KR*, 96:374–384, 1996.

[KNHD97] Jana Koehler, Bernhard Nebel, Jörg Hoffmann, and Yannis Dimopoulos. Extending planning graphs to an adl subset. In *European Conference on Planning*, pages 273–285. Springer, 1997.

[KS+92] Henry A Kautz, Bart Selman, et al. Planning as satisfiability. In *ECAI*, volume 92, pages 359–363, 1992.

[KS98] Henry Kautz and Bart Selman. Blackbox: A new approach to the application of theorem proving to problem solving. In *AIPS98 Workshop on Planning as Combinatorial Search*, volume 58260, pages 58–60, 1998.

[KS99] Henry Kautz and Bart Selman. Unifying sat-based and graph-based planning. In *IJCAI*, volume 99, pages 318–325, 1999.

[KW00] Henry Kautz and Joachim P Walser. Integer optimization models of ai planning problems. *The Knowledge Engineering Review*, 15(1):101–117, 2000.

[Lab03] Philippe Laborie. Algorithms for propagating resource constraints in ai planning and scheduling: Existing approaches and new results. *Artificial Intelligence*, 143(2):151–188, 2003.

[LF99] Derek Long and Maria Fox. Efficient implementation of the plan graph in stan. *Journal of Artificial Intelligence Research*, 10:87–115, 1999.

[LF03] Derek Long and Maria Fox. Exploiting a graphplan framework in temporal planning. 2003.

[LG95] Philippe Labone and Malik Ghallab. Planning with sharable resource constraints. In *Proceedings of the 14th international joint conference on Artificial intelligence*, volume 2, pages 1643–1649. Citeseer, 1995.

[McD97] Drew McDermott. The current state of ai planning research. In *Proc. The Ninth International Conf. of Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pages 25–34, 1997.

[MGH+98] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. Pddl-the planning domain definition language. 1998.

[MH69] John McCarthy and Patrick Hayes. Some philosophical problems from the standpoint of artificial intelligence. 4:463–502, 01 1969.

[MJ04] Filip Marić and Predrag Janičić. argo-lib: A generic platform for decision procedures. In *International Joint Conference on Automated Reasoning*, pages 213–217. Springer, 2004.

[ML06] Michał Moskal and Jakub Lopuszanski. Fast quantifier reasoning with lazy proof explication, 2006.

[MMZ+01] Matthew W Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient sat solver. In *Proceedings of the 38th annual Design Automation Conference*, pages 530–535. ACM, 2001.

[MR08] Frederic Maris and Pierre Régnier. Tlp-gp: New results on temporally-expressive planning benchmarks. In *Tools with Artificial Intelligence, 2008. ICTAI'08. 20th IEEE International Conference on*, volume 1, pages 507–514. IEEE, 2008.

[MS08] Joao Marques-Silva. Practical applications of boolean satisfiability. *Proceedings - 9th International Workshop on Discrete Event Systems, WODES' 08*, pages 74–80, 2008.

[MSS99] João P Marques-Silva and Karem A Sakallah. Grasp: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

[NO79] Greg Nelson and Derek C Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 1(2):245–257, 1979.

[NO80] Greg Nelson and Derek C. Oppen. Fast Decision Procedures Based on Congruence Closure. *Journal of the ACM*, 27(2):356–364, 1980.

[NS+72] Allen Newell, Herbert Alexander Simon, et al. *Human problem solving*, volume 104. Prentice-Hall Englewood Cliffs, NJ, 1972.

[Ped89] Edwin PD Pednault. Adl: Exploring the middle ground between strips and the situation calculus. *Kr*, 89:324–332, 1989.

[Rib11] Bruno Cesar Ribas. Satisfatibilidade não-clausal restrita às variáveis de entrada. Master's thesis, Universidade Federal do Paraná, 2011.

[Ric85] Elaine Rich. Artificial intelligence and the humanities. *Computers and the Humanities*, 19(2):117–122, 1985.

[RN03] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. 2003.

[SCK00] Fabiano Silva, Marcos Alexandre Castilho, and Luis Allan Künzle. Petriplan: a new algorithm for plan generation (preliminary report). In *Advances in Artificial Intelligence*, pages 86–95. Springer, 2000.

[SD05] Ji-Ae Shin and Ernest Davis. Processes and continuous change in a sat-based planner. *Artificial Intelligence*, 166(1-2):194–253, 2005.

[Seb07] Roberto Sebastiani. Lazy Satisfiability Modulo Theories. *Journal on Satisfiability, Boolean Modeling and Computation*, 3:141–224, 2007.

[Sho78] Robert E. Shostak. An algorithm for reasoning about equality. *Communications of the ACM*, 21(7):583–585, 1978.

[Sho79] Robert E. Shostak. A Practical Decision Procedure for Arithmetic with Function Symbols. *Journal of the ACM*, 26(2):351–360, 1979.

[Sho84] Robert E. Shostak. Deciding Combinations of Theories. *Journal of the ACM*, 31(1):1–12, 1984.

[Sil05] Fabiano Silva. Rede de planos: uma proposta para a solução de problemas de planejamento em inteligência artificial usando redes de petri. 2005.

[Str02] Ofer Strichman. Optimizations in decision procedures for propositional linear inequalities. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, 2002.

[TEHN96] Reiko Tsuneto, Kutluhan Erol, James Hendler, and Dana Nau. Commitment strategies in hierarchical task network planning. In *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, pages 536–542, 1996.

[VDBBKV07] Menkes Van Den Briel, J Benton, Subbarao Kambhampati, and Thomas Vossen. An lp-based heuristic for optimal planning. In *International Conference on Principles and Practice of Constraint Programming*, pages 651–665. Springer, 2007.

[WAS98] Daniel S Weld, Corin R Anderson, and David E Smith. Extending graphplan to handle uncertainty & sensing actions. In *Aaai/iaai*, pages 897–904, 1998.

[Wel99] Daniel S. Weld. Recent Advances in AI Planning. *AI Magazine*, 20(2):93, 1999.

[WW99] Steven A Wolfman and Daniel S Weld. The lpsat engine & its application to resource planning. In *IJCAI*, pages 310–317, 1999.

[WW01] Steven A Wolfman and Daniel S Weld. Combining linear programming and satisfiability solving for resource planning. *The Knowledge Engineering Review*, 16(1):85–99, 2001.

[Zha97] Hantao Zhang. Sato: An efficient prepositional prover. *Automated Deduction—CADE-14*, pages 272–275, 1997.

[ZKC01] Zhihong Zeng, Priyank Kalla, and Maciej Ciesielski. Lpsat: a unified approach to rtl satisfiability. In *Design, Automation and Test in Europe, 2001. Conference and Exhibition 2001. Proceedings*, pages 398–402. IEEE, 2001.

APÊNDICE A – MODELAGEM DO PROBLEMA DOS JARROS

Neste Apêndice apresenta-se a descrição completa do domínio do Problema dos Jarros, proposto por Gregory, Derek, Maria Fox e Beck, disponível em seu acervo pessoal.

A.1 MODELAGEM DO DOMÍNIO DO PROBLEMA DOS JARROS

```
(define (domain jugpour)
  (:types
   jug
  )

  (:modules integer set)

  (:functions
     (quantity ?t - jug) - integer
     (space    ?t - jug) - integer
     (capacity ?t - jug) - integer
  )


(:action fill-jug
  :parameters
   (?t - jug)
  :precondition
   (true)
  :effect
   (
    (assign (quantity ?t) (capacity ?t))
    (assign (space ?t) 0)
   )
)


(:action empty-jug
  :parameters
   (?t - jug)
  :precondition
   (true)
  :effect
   (
    (assign (quantity ?t) 0)
    (assign (space ?t) (capacity ?t))
   ))
```

```
(:action transfer1
  :parameters
   (?t1 - jug ?t2 - jug)
  :precondition
   (
    (!= ?t1 ?t2)
    (< (space ?t2) (quantity ?t1))
    (= (space ?t1) (-- (capacity ?t1) (quantity ?t1)) )
    (= (space ?t2) (-- (capacity ?t2) (quantity ?t2)) )
   )
  :effect
   (
    (decrease (quantity ?t1) (space ?t2))
    (increase (space ?t1) (space ?t2))
    (assign (quantity ?t2) (capacity ?t2))
    (assign (space ?t2) 0)
   )
)


(:action transfer3
  :parameters
   (?t1 - jug ?t2 - jug)
  :precondition
   (
    (!= ?t1 ?t2)
    (> (space ?t2) (quantity ?t1))
    (= (space ?t1) (-- (capacity ?t1) (quantity ?t1)) )
    (= (space ?t2) (-- (capacity ?t2) (quantity ?t2)) )
   )
  :effect
   (
    (decrease (space ?t2) (quantity ?t1))
    (increase (quantity ?t2) (quantity ?t1))
    (assign (quantity ?t1) 0)
    (assign (space ?t1) (capacity ?t1))
   )
)

)
```

APÊNDICE B – MODELAGEM DO PROBLEMA TEMPORAL DOS JARROS

Neste Apêndice apresenta-se a descrição completa do domínio do Problema Temporal dos Jarros.

B.1 MODELAGEM DO DOMÍNIO DO PROBLEMA DOS JARROS

```
(define (domain jugpour)
(:types
    jug
    act
)

(:modules integer set temporal )

(:functions
    (quantity  ?t - jug) - integer
    (space     ?t - jug) - integer
    (capacity  ?t - jug) - integer
    (free-act  ?a - act) - boolean
    ;;(time-to-transfer ?t ?t1 - jug) - integer
)


(:action fill-jug
    :parameters(
       ?t - jug
       ?a - act
    )
    :precondition(
        (duration-of (this) (capacity ?t))
        (< (quantity ?t) (capacity ?t))
        (at-start (free-act ?a))
    )
    :effect(
        (at-start (assign (free-act ?a) false))

        (assign (quantity ?t) (capacity ?t))
        (assign (space ?t) 0)

        (at-end (assign (free-act ?a) true))
    )
)
```

```
(:action empty-jug
    :parameters(
        ?t - jug
        ?a - act
    )
    :precondition(
        (duration-of (this) (quantity ?t))
        (> (quantity ?t) 0)
        (at-start(free-act ?a))
    )
    :effect(
        (at-start (assign (free-act ?a) false))
        (assign (quantity ?t) 0)
        (assign (space ?t) (capacity ?t))
        (at-end (assign (free-act ?a) true))
    )
)


(:action transfer1
    :parameters(
        ?t1 - jug
        ?t2 - jug
        ?a  - act
    )
    :precondition(
        (duration-of (this)
            (decrease (capacity ?t2) (quantity ?t2)) )
        (!= ?t1 ?t2)
        (at-start (free-act ?a))
        ;; jug 2
        (<= (space ?t2) (quantity ?t1))
        (= (space ?t1) (-- (capacity ?t1) (quantity ?t1)) )
        (= (space ?t2) (-- (capacity ?t2) (quantity ?t2)) )

    )
    :effect(
        (decrease (quantity ?t1) (space ?t2))
        (increase (space ?t1) (space ?t2))
        (assign (quantity ?t2) (capacity ?t2))
        (assign (space ?t2) 0)
        (at-start (assign (free-act ?a) false))
        (at-end (assign (free-act ?a) true))
    )
)
```

```
(:action transfer3
    :parameters(
        ?t1 - jug
        ?t2 - jug
        ?a  - act
    )
    :precondition(
        (duration-of (this) (quantity ?t1) )
        (!= ?t1 ?t2)
        (at-start(free-act ?a))
        (> (quantity ?t1) 0)
        (>= (space ?t2) (quantity ?t1))
        (= (space ?t1) (-- (capacity ?t1) (quantity ?t1)) )
        (= (space ?t2) (-- (capacity ?t2) (quantity ?t2)) )
    )
    :effect(
(decrease (space ?t2) (quantity ?t1))
        (increase (quantity ?t2) (quantity ?t1))
        (assign   (quantity ?t1) 0)
        (assign   (space ?t1) (capacity ?t1))

(at-start (assign (free-act ?a) false))
        (at-end (assign (free-act ?a) true))
    )
)

)
```