Topologia de espaços de estados em planejamento com variáveis numéricas

Aldebaran Perseke

Dissertação apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do grau
de
Mestre em Ciência da Computação

Área de concentração: Inteligência Artificial
Orientadora: Prof. Dra. Leliane Nunes de Barros

— São Paulo, Agosto/2003 —

Topologia de espaços de estados em planejamento com variáveis numéricas

Este exemplar corresponde à redação final
da dissertação de mestrado devidamente
corrigida e defendida por
Aldebaran Perseke
e aprovada pela comissão julgadora.

São Paulo, Agosto/2003

Banca examinadora:

. Prof. Dra. Leliane Nunes de Barros (orientadora) - IME/USP
. Prof. Dr. Marcelo Finger - IME/USP
. Prof. Dra. Maria Carolina Monard - ICMS/USP

À minha família.

Agradecimentos

À minha esposa Simone de Bittencourt,
que jamais me permitiu sucumbir às barreiras da jornada.
À professora Leliane Nunes de Barros,
que me orientou e me incentivou nesse trabalho.
Aos professores do Departamento de Computação,
que contribuíram para minha formação no mestrado.
À todos os meus familiares, pela força e apoio de sempre.

Resumo

As pesquisas sobre planejamento em Inteligência Artificial, por muito tempo, se preocuparam apenas com os aspectos lógicos dos problemas relativos à geração de cursos de ações ou planos, isto é, a realização de objetivos de satisfação lógica. Por sua vez, os trabalhos na área de escalonamento procuram resolver problemas de atribuição de tarefas, previamente escolhidas, aos recursos que irão executá-las, de forma a otimizar a utilização desses recursos, bem como satisfazer restrições sobre tempo e variáveis do domínio. Entretanto, muitos problemas reais possuem características híbridas, situando-se em uma região intermediária entre o planejamento e o escalonamento. Diante desse desafio, foram propostos na competição internacional de planejamento, *IPC-2002*, novos domínios e problemas envolvendo aspectos tanto de planejamento como de escalonamento. Como mostram os resultados dessa competição, pode-se afirmar que esses novos domínios apresentam dificuldades para os melhores planejadores até então desenvolvidos, entre eles os planejadores de busca heurística. Hoffmann propôs uma teoria para análise do espaço de busca de domínios de planejamento que mostra quais são as características dos domínios que tornam os problemas fáceis ou difíceis para um planejador. Porém, sua análise se limitou aos domínios anteriores à competição de 2002. Este trabalho estende a teoria de Hoffmann para problemas envolvendo a utilização de recursos e faz uma análise teórica dos novos domínios da competição internacional *IPC-2002*. Finalmente, esses resultados são confrontados com dados obtidos empiricamente a partir da execução de um planejador implementado, capaz de tratar problemas envolvendo variáveis numéricas relacionadas a utilização de recursos.

Abstract

Since the beginning, research in Artificial Inteligence Planning has only focused on the logical aspects of the plan generation, i.e., how to achieve logical goals. On the other hand, research from the schedulling community, tipically tries to solve the problem of assigning predefined tasks to a number of resources (machines) in order to optimize their use, while satisfying time and domain variables constraints. Nevertheless, many real problems exhibit aspects from both areas: planning and schedulling. To tackle this challenge, the *IPC-2002* international competition proposed new planning domains and problems involving both aspects. Based on the competition results, one can assert that the new domains offer aditional difficulties even to the best planners developed so far, including the heuristic planners. Hoffmann proposed a theory to analise topological aspects of a domain search space that indicates the difficulty that a planner can find during problem solving in a particular domain. However, this analisis was limited to domains from previous competition, *IPC-2000*. This work extends Hoffmann's theory to deal with problems involving resources allocation which are used to analise the new domains. Finally, it was implemented a planner able to solve such new problems from *IPC-2002* competition and its performance is then compared with the theoretical prediction about the analised domains.

Sumário

1 Introdução **1**

1.1 Motivação 1

1.2 Objetivo 3

1.3 Organização 4

2 Planejamento em Inteligência Artificial **7**

2.1 O problema de planejamento 7

2.2 Descrição formal do problema de planejamento 8

2.3 Planejamento como um problema em um modelo de estados 9

2.4 Planejamento como inferência lógica: o cálculo de situações 10

2.5 Exemplo clássico de domínio de planejamento: o mundo dos blocos . . . 12

2.6 O problema do quadro 14

2.7 Abordagens lógicas para planejamento 15

2.8 Algoritmos para planejamento clássico 17

2.8.1 Planejamento como busca no espaço de estados 18

2.8.2 Planejamento como busca no espaço de planos 20

2.8.3 Planejamento de ordem parcial 21

2.9 A linguagem STRIPS 23

2.10 Outras linguagens 25

2.11 Considerações finais 27

3 O planejador *Graphplan* **29**

3.1 O grafo de planejamento 30
3.2 Relações binárias de exclusão mútua 31
3.3 A busca por uma solução 34
3.4 O algoritmo *Graphplan* 36
3.5 O planejador *Blackbox* 37
3.6 Considerações finais 38

4 Planejamento & Escalonamento **41**
4.1 Planejamento e escalonamento 42
4.2 Recursos 43
4.3 Variáveis numéricas 44
4.4 Abordagens integradas para planejamento e escalonamento 45
4.4.1 O planejador *ZENO* 46
4.4.2 O planejador *IPP* 47
4.4.3 O planejador *LPSAT* 47
4.4.4 O planejador *ILP-Plan* 48
4.5 Domínios com variáveis numéricas 49
4.5.1 O problema do telescópio espacial 49
4.5.2 O problema do robô de exploração 52
4.6 Considerações finais 56

5 O planejador *FF* **57**
5.1 O algoritmo *Hill-Climbing* reforçado 58
5.2 A heurística do *FF* 59
5.3 Transformação de esquemas de operadores 63
5.4 O planejador *Metric-FF* 64
5.4.1 Variáveis numéricas de estado 64
5.4.2 A heurística do *Metric-FF* 68
5.5 O planejador *GMetric-FF* 72

5.6 Considerações finais 73

6 Análise dos domínios de teste **75**

6.1 Topologia de espaços de estados em planejamento 76

6.1.1 Becos sem Saída 77

6.1.2 Platôs 80

6.2 A Teoria de Hoffmann 81

6.2.1 Verificação de becos sem saída 81

6.2.2 Verificação de mínimos locais 85

6.3 Extensão da teoria para domínios numéricos 86

6.4 Análise dos domínios estudados 89

6.4.1 Domínio do mundo dos blocos 90

6.4.2 Domínio do Satélite 91

6.4.3 Domínio do Robô de Exploração 92

6.4.4 Domínio do Satélite com variáveis numéricas 94

6.4.5 Domínio do Robô de Exploração com variáveis numéricas 96

6.4.6 Taxonomia dos domínios analisados 97

6.5 Considerações finais 99

7 Resultados experimentais **101**

7.1 Metodologia empregada nos testes 102

7.2 Análise experimental nos domínios STRIPS 103

7.2.1 Sobre a qualidade do planejador *FF* implementado 103

7.2.2 Geração de ações proposições nos domínios STRIPS 103

7.2.3 Problemas resolvidos nos domínios STRIPS 105

7.3 Análise experimental nos domínios numéricos 109

7.3.1 Geração de ações proposições nos domínios numéricos 109

7.3.2 Problemas resolvidos nos domínios numéricos 110

7.4 Resultados da competição IPC-2002 111

7.5 A heurística gulosa do *GMetric-FF* 114

8 Conclusão **119**

8.1 Principais contribuições 120

8.2 Trabalhos futuros 121

A O planejador *Satplan* **129**

A.1 Algoritmos de satisfatibilidade 129

A.2 O algoritmo *Satplan* 130

A.3 Compilação do problema em uma CNF 132

B Especificação Completa do Graphplan **135**

C Domínios da competição AIPS-2002 **145**

C.1 O domínio do satélite STRIPS 145

C.2 Exemplo de problema para a versão STRIPS 147

C.3 O domínio do satélite com variáveis numéricas 149

C.4 Exemplo de problema para a versão númerica 151

C.5 O domínio do robô de exploração STRIPS 154

C.6 Exemplo de problema para a versão STRIPS 159

C.7 O domínio do robô de exploração com variáveis numéricas 162

C.8 Exemplo de problema para a versão númerica 167

D Demonstrações **171**

D.1 Domínio do mundo dos blocos 171

D.2 Domínio do satélite 176

D.3 Domínio do robô de exploração 180

Lista de Figuras

2.1 Estado inicial e estado objetivo para o problema "Anomalia de Sussman" no mundo dos blocos 12

2.2 Grafo do espaço de estados para o problema no mundo dos blocos com três blocos A, B e C. 18

2.3 Exemplo de um espaço de planos parcialmente ordenados 21

2.4 Vínculo causal ameaçado 22

2.5 Pseudo código do algoritmo *POP* 23

2.6 Representação STRIPS do estado inicial e do estado objetivo para o problema da anomalia de Sussman, além das ações do domínio do mundo dos blocos 26

3.1 Representação STRIPS do estado inicial e do estado objetivo para o problema do jantar surpresa, além das ações do domínio 31

3.2 **Grafo de Planejamento** para o domínio do jantar surpresa 32

3.3 Pseudo código do algoritmo que constrói o grafo de planejamento para um problema 34

3.4 Pseudo código do algoritmo que extrai uma solução de um grafo de planejamento 35

3.5 Pseudo código do algoritmo *Graphplan* 37

5.1 Pseudo código do algoritmo *Hill-Climbing* reforçado 58

5.2 Grafo de planejamento para a versão relaxada do problema da anomalia de Sussman 62

5.3 Desempenho dos planejadores na competição IPC-2000 no domínio do Mundo dos Blocos 64

5.4 Pseudo código do algoritmo para instanciar esquemas de operadores . . 65

5.5 Pseudo código do algoritmo de construção do grafo de planejamento relaxado com variáveis numéricas 70

5.6 Pseudo código do algoritmo de busca por um plano no grafo de planejamento relaxado com variáveis numéricas 74

6.1 Taxonomia dos domínios da competição *IPC-2000* 89

6.2 Taxonomia dos domínios analisados. As regiões em cinza indicam domínios em que os planejadores estudados devem ser capazes de resolver todas as instâncias de problemas. A região marcada com um X, indica que não existem domínios sem mínimos locais e com becos não-reconhecíveis. De acordo com a teoria estudada, todo domínio com becos não-reconhecíveis deve possuir mínimos locais. 98

7.1 Tamanhos dos planos encontrados pelo planejador *FF* implementado e pelo planejador da competição em problemas no domínio do Satélite . . 104

7.2 Tamanhos dos planos encontrados pelo planejador *FF* implementado e pelo planejador da competição em problemas no domínio do do Robô de Exploração 105

7.3 Quantidade de ações totalmente proposicionais dos problemas STRIPS resolvidos pelo planejador *FF* 106

7.4 Tempo médio gasto para encontrar um plano para os problemas resolvidos pelo planejador *FF*. Os tempos foram calculados como a média dos tempos gastos em dez execuções do planejador para cada um dos problemas. 107

7.5 Tamanho do plano encontrado para os problemas resolvidos pelo planejador *FF*. Nos problemas em que não foi possível encontrar uma solução, o tamanho do plano é igual a zero. 108

7.6 Quantidade de ações totalmente proposicionais dos problemas numéricos resolvidos pelo planejador *GMetric-FF* 109

7.7 Tempo médio gasto para encontrar um plano para os problemas resolvidos pelo planejador *GMetric-FF* 110

7.8 Tamanho do plano encontrado para os problemas resolvidos pelo planejador *GMetric-FF* 111

7.9 Resultados da competição *IPC-2002* para o domínio do Satélite com variáveis numéricas 112

7.10 Resultados da competição *IPC-2002* para o domínio do Robô de Exploração com variáveis numéricas 113

7.11 Tamanhos dos planos encontrados pelo planejador *GMetric-FF* implementado executado nos modos qualidade e rapidez em problemas no domínio do Satélite 114

7.12 Tamanhos dos planos encontrados pelo planejador *GMetric-FF* implementado executado nos modos qualidade e rapidez em problemas no domínio do Robô de Exploração 115

8.1 Taxonomia dos domínios analisados. 120

A.1 Arquitetura do planejador *Satplan* 131

A.2 Representação STRIPS do estado inicial e do estado objetivo para o problema do jantar surpresa, além das ações do domínio 132

Capítulo 1

Introdução

1.1 Motivação

Formas de vida inteligentes e sofisticadas, como os seres humanos, possuem a capacidade de deliberar sobre as ações que devem tomar para atingir seus objetivos. Para tal, agentes inteligentes, robóticos ou de software, devem raciocinar sobre seus objetivos e as ações que podem executar para atingi-los, raciocínio esse que é chamado de **planejamento** em Inteligência Artificial.

A abordagem mais simples para resolver problemas de planejamento é provar a existência de uma seqüência de ações que alcancem as condições do estado objetivo [13], o que pode ser feito com o uso de algum sistema automático ou interativo para prova de teoremas. As abordagens algorítmicas baseiam-se na busca pelo espaço de estados de um problema [23]. Inflexível em alguns casos, a busca no espaço de estados foi então suplantada pelos *planejadores de ordem parcial*, que efetuam busca em um espaço de planos parcialmente ordenados.

Recentemente, três abordagens promoveram uma verdadeira revolução na maneira de resolver problemas de planejamento: *Satplan*, *Graphplan* e planejamento por buscas heurísticas [17]. O sucesso de alguns algoritmos, como o *GSAT*, que resolvem problemas de satisfatibilidade proposicional motivou uma abordagem em que o problema de planejamento é formulado como um problema de satisfatibilidade. Essa abordagem deu origem ao planejador *Satplan* [26], que apresentou uma melhora significativa de desem-

penho quando comparado aos planejadores de ordem parcial. O planejador *Graphplan* [3] emprega uma abordagem diferente para realizar uma busca por planos. A idéia básica é o uso de um tipo de análise de conflitos na escolha de ações, por meio do *grafo de planejamento* de forma a eliminar muitas das combinações e seqüências de ações que não são compatíveis entre si [46].

Os algoritmos de busca heurística, que realizam a busca pelo espaço de estados, ganharam nova força com a utilização de uma heurística, confiável e calculada de forma eficiente, baseada na solução de uma tarefa de planejamento relaxada. No planejador *FF* [17] isso foi realizado por meio da construção do *grafo de planejamento*, proposto pelo *Graphplan*, para resolver a tarefa relaxada. Esse planejador emprega um algoritmo que combina busca local e busca sistemática, o que tornou o *FF* um dos mais bem sucedidos planejadores da competição internacional de planejamento, *IPC-2002*.

As pesquisas sobre planejamento em Inteligência Artificial sempre focaram os problemas NP-difíceis nos quais existe uma dependência em cascata nas escolhas das ações, e ocorrem complexas interações entre as ações. Além disso, por muito tempo, a área se preocupou apenas com os aspectos lógicos dos problemas relativos à geração de cursos de ações ou planos, isto é, a realização de objetivos de satisfação lógica. Em contrapartida, as pesquisas sobre escalonamento focaram problemas nos quais a escolha de ações (ou tarefas) é trivial, mas o problema de alocação temporal das tarefas e otimização de uma função objetivo são de grande dificuldade [46]. Porém, muitos problemas práticos do mundo real exibem características tanto de planejamento quanto de escalonamento. Um exemplo de problema bastante relevante que envolve planejamento e escalonamento é apresentado por Smith [47], e consiste na construção das escalas de pequenas observações para o Telescópio Espacial Hubble (HST). Uma versão desse problema foi incluída ao conjunto de domínios a que foram submetidos os sistemas de planejamento que participaram da competição de planejamento e escalonamento do congresso internacional *AIPS-2002*. As competições que ocorreram durante os congressos internacionais *AIPS-1998* e *AIPS-2000* [1] contaram apenas com domínios puramente proposicionais. Mesmo assim, a evolução dos sistemas de planejamento foi surpreendente durante os

[1] Mais informações sobres esses três congressos internacionais podem ser obtidos pelos seguintes endereços eletrônicos: http://www-2.cs.cmu.edu/ aips98/, http://www-aig.jpl.nasa.gov/public/aips00/aips_home.html - para o *AIPS-2000*, e http://www.laas.fr/aips/ - para o *AIPS-2002*.

anos dessas duas primeiras competições. Em função desse avanços, e com a intenção de atrair o maior número possível de pesquisadores para sua competição, o congresso internacional *AIPS-2002* [2] teve como objetivo apresentar problemas de planejamento que envolvessem raciocínio sobre tempo e variáveis numéricas [35]. Dessa forma, os problemas deixaram de ter um caráter estritamente artificial, e começaram a ter desafios muito mais próximos dos encontrados em problemas práticos do mundo real.

Muitos dos sistemas desenvolvidos para resolver os problemas de planejamento são incapazes de representar ou raciocinar sobre aspectos típicos de escalonamento, como recursos, variáveis métricas, tempo, e a otimização de alguma função objetivo. Apesar de diversas propostas de extensões às técnicas de planejamento de forma a permitir que sistemas de planejamento possam tratar dos aspectos de escalonamento [46], como será mostrado neste trabalho, não existe ainda uma proposta de planejador capaz de resolver todos os problemas dos novos domínios propostos na competição *AIPS-2002*.

Em função de todos os avanços recentes, muito há para ser estudado e pesquisado pois, as novidades nos algoritmos e na descrição dos problemas de planejamento engendrou várias questões, muitas delas ainda em aberto. Dentre os estudos mais recentes, Hoffmann [22] propôs uma teoria para análise do espaço de busca de domínios de planejamento que mostra quais são as características dos domínios que tornam os problemas fáceis ou difíceis para um planejador. Este trabalho estende a teoria de Hoffmann para problemas envolvendo a utilização de recursos e faz uma análise teórica e empírica dos novos domínios da competição internacional *IPC-2002*.

1.2 Objetivo

A proposta desse trabalho consiste em:

- Apresentar um estudo sobre a evolução da área de planejamento em Inteligência Artificial visando a introdução do vocabulário necessário para o restante do trabalho;

[2] Para fins de simplificação, as competições serão referidas pelo acrônimo IPC (*International Planning Competition*) seguido do ano em que a mesma ocorreu

- Apresentar um estudo sobre alguns dos sistemas de planejamento que solucionam problemas que envolvam recursos e variáveis numéricas;
- Implementar um sistema de planejamento capaz de tratar problemas envolvendo variáveis numéricas relacionadas a utilização de recursos;
- Apresentar uma teoria criada para classificar domínios de planejamento segundo as propriedades topológicas do seu espaço de busca [22];
- Propor uma extenção da teoria de Hoffmann para problemas envolvendo a utilização de recursos representados como variáveis numéricas e analisar os domínios da competição *AIPS-2002*, que não foram analisados por Hoffmann;
- Apresentar uma análise empírica com o sistema de planejamento implementado sobre instâncias de problemas nos domínios novos estudados e analisados teoricamente.

Dentre as contribuições que este trabalho traz, estão:

- Análise de dois novos domínios de planejamento: o domínio do Satélite e o domínio do Robô de Exploração usando a teoria de Hoffmann em suas versões Strips e numérica;
- Extensão da teoria de Hoffmann para problemas que envolvem variáveis numéricas. Somente com tal extensão foi possível analisar as versões numéricas dos domínios;
- Modificação na estratégia do *Metric-FF*, utilizando a função objetivo para o desempate entre estados de mesma heurística na busca local, aumentando a eficiência deste planejador.

1.3 Organização

Essa dissertação está organizada da seguinte maneira:

Capítulo 2 apresenta o problema de planejamento e as abordagens clássicas para resolvê-lo. O objetivo desse capítulo é fornecer uma visão geral da área de pla-

nejamento em Inteligência Artificial e das maneiras utilizadas para descrever e resolver problemas de planejamento clássico.

Capítulo 3 apresenta o sistema de planejamento *Graphplan*; O objetivo desse capítulo é apresentar a estrutura de dados chamada de grafo de planejamento, construída e utilizada pelo planejador *Graphplan* para o processo de busca por um plano solução.

Capítulo 4 apresenta os aspectos que caracterizam os problemas de planejamento e escalonamento, e quais são as abordagens de extensão de sistemas de planejamento mais indicadas para resolver esses problemas; apresenta os domínios do Satélite e do Robô de Exploração em suas versões totalmente proposicional e numérica, ou seja, que inclui aspectos de escalonamento; O objetivo desse capítulo é familiarizar o leitor com os aspectos típicos de problemas de escalonamento e as formas de representar tais aspectos em problemas tipicamente de planejamento.

Capítulo 5 apresenta o planejador *FF* e o planejador *Metric-FF*; O objetivo desse capítulo é descrever os dois planejadores implementados como parte desse trabalho.

Capítulo 6 apresenta uma série de definições e lemas que serão utilizados para tirar conclusões sobre o grau de dificuldade dos problemas empregados nos testes experimentais; O objetivo desse capítulo é fornecer justificativas teóricas para os comportamentos que se espera encontrar na resolução de problemas com planejadores que fazem busca local.

Capítulo 7 apresenta os resultados experimentais obtidos com a implementação do planejador *FF* aplicado na resolução de problemas totalmente proposicionais, e do planejador *GMetric-FF* aplicado na resolução de problemas numéricos; O objetivo desse capítulo é comprovar empiricamente as características esperadas a partir da análise teórica para os domínio estudados.

Capítulo 8 apresenta as conclusões obtidas com esse trabalho indicando suas principais contribuições, além de apontar novas direções que podem ser seguidas a partir dos resultados alcançados até aqui.

Capítulo 2

Planejamento em Inteligência Artificial

Esse capítulo fornece uma visão geral da área de planejamento em Inteligência Artificial, apresentando as maneiras utilizadas para descrever os problemas de planejamento, assim como os métodos clássicos para resolvê-los.

2.1 O problema de planejamento

Uma característica fundamental de formas de vida inteligentes e sofisticadas, como os seres humanos, é a capacidade de raciocinar sobre as ações apropriadas que devem tomar em seu ambiente para com isso atingir seus objetivos.

Tipicamente, o raciocínio sobre a escolha de ações e da ordem em que elas devem ser executadas é chamado de planejamento. Para exemplificar, considere o problema real de planejamento de uma viagem, saindo de São Paulo e chegando à Paris. Esse objetivo consiste de várias etapas: reservar as passagens aéreas, ir ao aeroporto, trocar reais por francos, fazer a reserva do hotel, encontrar o hotel ao chegar à Paris, etc. Atingir esse objetivo requer a execução de um conjunto complexo de ações na ordem correta, e uma pessoa prudente deveria gastar algum tempo raciocinando sobre essas ações e sua ordem apropriada com antecedência [6]. Esse tipo de comportamento inteligente é muito mais sofisticado do que o comportamento reativo, em que as ações tomadas são simplesmente reações imediatas aos estímulos recebidos do ambiente [40]. Com

a finalidade de reproduzir as características desse tipo de comportamento inteligente, a área de planejamento em Inteligência Artificial dedica-se ao estudo de formas de representação dessa classe de problemas, bem como a elaboração de algoritmos para solucioná-los.

Fundamentalmente, planejar é uma tarefa de síntese, que envolve a formulação de um curso de ações para permitir a um agente [1] atingir alguns objetivos desejados. Muitos problemas de planejamento são combinatórios, e portanto o número dos possíveis cursos de ação, ou o tempo necessário para avaliá-los é exponencial na descrição do problema, sendo que vários problemas pertencem a classe NP-difícil.

O desafio de construir agentes inteligentes levou os pesquisadores de Inteligência Artificial (IA) a investigar algoritmos de planejamento cada vez mais eficientes a fim de solucionar essa classe de problemas. Os recentes avanços das pesquisas nessa área aliados a recursos computacionais cada vez mais poderosos tem permitido a solução de problemas de domínios cada vez mais complexos, como por exemplo, o planejamento de manufatura [36], o planejamento de escalas de observações espaciais [47], o planejamento aplicado à problemas de logística [1], etc.

2.2 Descrição formal do problema de planejamento

O problema de planejamento pode ser formalmente descrito por uma 3-upla $< \Delta, \Sigma, \Gamma >$ [2], onde:

1. Δ representa a descrição das possíveis ações que podem ser escolhidas pelo agente, denominada teoria do domínio;
2. Σ representa a descrição do estado inicial do mundo;
3. Γ representa a descrição do objetivo do agente.

A 3-upla $< \Delta, \Sigma, \Gamma >$ é o argumento de entrada para um algoritmo de planejamento, cuja resposta é uma seqüência Π de ações que, quando executada em qualquer mundo que satisfaça a descrição Σ do estado inicial, irá atingir o estado objetivo Γ.

[1] Em Inteligência Artificial utiliza-se o termo **agente** para designar uma entidade que percebe e age em um determinado ambiente, podendo ser um sistema robótico ou até mesmo um *software*.

Essa formulação para o problema de planejamento é bastante abstrata, pois na verdade ela especifica uma classe de problemas de planejamento parametrizados pelas linguagens utilizadas para representar o mundo, os objetivos e as ações. Outro ponto importante que deve ser enfatizado é que a área de planejamento em IA trata pura e simplesmente de como representar o mundo por meio de um modelo de ações, e como seqüenciar ações para atingir os objetivo de um agente. Assim, não é uma preocupação da área de planejamento em IA o estudo do como esse agente percebe o mundo e como ele executa suas ações.

2.3 Planejamento como um problema em um modelo de estados

Espaços de estados compõem o modelo fundamental de ações para problemas envolvendo ações determinísticas e informação completa. Um estado é simplesmente um conjunto de propriedades ou predicados que descrevem um instante particular do modelo. Um espaço de estados consiste de um número finito de estados $\mathcal{S}$, um conjunto finito de ações $\mathcal{A}$ que podem ser executadas, e uma função de transição de estados f que descreve como uma ação aplicada a um estado leva a um estado resultante. Um espaço de estados estendido com um estado inicial s_0 e um conjunto S_G de estados objetivo é chamado de modelo de estados [4].

Definição 1 *Um modelo de estados é descrito pela 5-upla* $< \mathcal{S}, s_0, S_G, \mathcal{A}, f >$ *onde,* $\mathcal{S}$ *é um conjunto finito e não vazio de estados* s, $s_0 \in \mathcal{S}$ *é um estado inicial,* $S_G \subseteq \mathcal{S}$ *é um sub-conjunto não vazio de estados objetivo,* $\mathcal{A}$ *é um conjunto de ações, sendo* $\mathcal{A}(s) \subseteq \mathcal{A}$ *o subconjunto de ações aplicáveis no estado* $s \in \mathcal{S}$, *e* $f(a, s)$ *é uma função de transição de estados para todos os estados* $s \in \mathcal{S}$ *e ações* $a \in \mathcal{A}(s)$.

Um modelo de estados pode ser representado por um grafo orientado, sendo que os vértices desse grafo correspondem aos estados e as arestas correspondem à aplicação de uma ação em um estado, e que de acordo com a função de transição de estados, conduz a um estado resultante. Nesse modelo, solucionar uma tarefa de planejamento consiste na busca por uma seqüência de ações $< a_1, a_2, \ldots, a_n >$, denominada plano solução, que define no grafo do modelo de estados um caminho orientado que vai do

estado inicial s_0 a um estado objetivo $s \in S_G$.

A complexidade da síntese de planos depende de uma variedade de propriedades do ambiente e do agente, a saber:

1. mudanças no ambiente dependem apenas de ações executadas pelo agente, ou podem acontecer fora do controle deste;
2. o ambiente é totalmente observável, ou parcialmente desconhecido;
3. as ações do agente tem efeitos determinísticos no estado do ambiente, ou podem ter efeitos estocásticos.

As simplificações mais comuns nessas propriedades consideram um ambiente estático, onde as mudanças ocorrem somente como resposta a execução de alguma ação pelo agente, o ambiente é totalmente observável, e as ações do agente têm efeitos determinísticos. A síntese de planos sob essas condições é conhecida como **problema de planejamento clássico** [23].

2.4 Planejamento como inferência lógica: o cálculo de situações

Agentes que planejam possuem uma representação interna do mundo em que estão inseridos e das ações que podem executar. Um dos formalismos mais antigos e bem conhecidos para representação de ações e seus efeitos em `IA` é o cálculo de situações [43]. O cálculo de situações é uma maneira de representação de um modelo de estados por meio do uso de sentenças da lógica. A linguagem do cálculo de situações é a lógica de predicados de primeira ordem, e apresenta uma ontologia particular que inclui situações, fluentes, e ações. Situações podem ser imaginadas como fotografias de um instante do mundo, descritas por meio de um conjunto de fluentes. Fluentes descrevem propriedades do mundo, que podem assumir valores distintos em diferentes situações, ou seja, propriedades variáveis no decorrer do tempo. Ações alteram os valores dos fluentes quando são executadas. Dentro do cálculo de situações, o predicado $Holds(\beta, s)$ indica

que um fluente β é verdadeiro na situação s. A função $Result(\alpha, s)$ [2] denota a nova situação obtida quando a ação α é executada na situação s. A conjunção de todos os predicados $Holds$ que descrevem o estado inicial s_0 de um problema será denotada por Σ.

Para descrever os efeitos de uma ação utiliza-se um conjunto de fórmulas chamadas de axiomas de efeito. Esses axiomas devem levar em consideração as pré-condições para a aplicação da ação, ou seja, os fluentes que devem ser verdadeiros e/ou que não devem ser verdadeiros na situação em que a ação será executada. Um axioma de efeito assume uma das seguintes formas:

$$\forall s[Holds(\beta, Result(\alpha, s)) \leftarrow \Theta]$$

$$\forall s[\neg Holds(\beta, Result(\alpha, s)) \leftarrow \Theta]$$

O primeiro axioma indica que o fluente β é verdadeiro na situação resultante da execução da ação α no estado s, e o segundo axioma indica que o fluente β não é verdadeiro na situação resultante da execução da ação α no estado s. Nos dois axiomas, Θ denota a pré-condição para aplicação da ação α, descrita como uma conjunção de predicados $Holds$:

$$Holds(\beta_1, s) \wedge \neg Holds(\beta_2, s) \wedge \ldots \wedge \neg Holds(\beta_{n-1}, s) \wedge Holds(\beta_n, s)$$

Qualquer termo presente na conjunção Θ pode assumir a forma $Holds(\beta_i, s)$ ou $\neg Holds(\beta_i, s)$. O conjunto de todas as sentenças que descrevem os axiomas de efeito de um domínio será denotada por Δ.

Definição 2 *Um plano é uma seqüência ordenada* $(\alpha_1, \alpha_2, \ldots, \alpha_k)$ *de ações, denotado por* Π.

O estado s_g resultante da aplicação ordenada da seqüência de ações de um plano Π em um estado s é dado pelo termo:

$$s_g = Result(\alpha_k, \ldots (Result(\alpha_2, Result(\alpha_1, s))) \ldots)$$

[2] Os símbolos predicativos do cálculo de situações foram mantidos em inglês para preservar a fidelidade dessa ontologia amplamente conhecida.

Definição 3 *Um plano* Π *resolve um problema de planejamento* $< \Delta, \Sigma, \Gamma >$ *se o estado* s_g *resultante da aplicação ordenada da seqüência de ações de* Π *no estado inicial* Σ *satisfaz todas as propriedades do estado objetivo* Δ *[48].*

Se um plano Π resolve um problema de planejamento, diz-se que esse plano é completo e correto.

2.5 Exemplo clássico de domínio de planejamento: o mundo dos blocos

Um domínio muito estudado em planejamento é o Mundo dos Blocos. Esse domínio consiste de um conjunto de blocos rotulados dispostos sobre uma mesa ou empilhados, sendo que apenas um bloco pode ser colocado diretamente no topo de outro bloco. Um braço robótico pode pegar um único bloco e movê-lo para a mesa ou para o topo de outro bloco. Os problemas são formulados em termos de uma disposição inicial do conjunto de blocos e da disposição que se deseja obter a partir de sucessivas movimentações de blocos feitas pelo robô. A despeito de sua aparente simplicidade, problemas nesse domínio capturam muitas das dificuldades encontradas durante a busca por uma solução, tendo sido provado que encontrar um plano ótimo é NP-difícil [15]. A Figura 2.1 apresenta um problema no Mundo dos Blocos conhecido como Anomalia de Sussman.

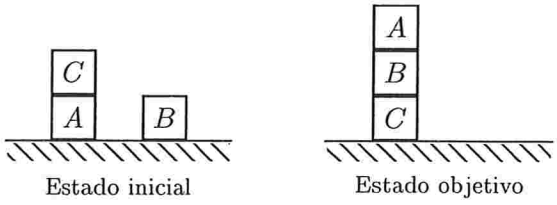

Figura 2.1: Estado inicial e estado objetivo para o problema "Anomalia de Sussman" no mundo dos blocos

Para descrever esse problema utilizando o cálculo de situações faz-se uso de cinco fluentes [10], a saber:

1. $Sobre(x, y)$: denota que o bloco x está sobre o bloco y;

2. $SobreMesa(x)$: denota que o bloco x está sobre a mesa;

3. $Limpo(x)$: denota que não há qualquer bloco sobre o bloco x;

4. $BracoLivre$: denota que o braço robótico não está segurando um bloco;

5. $Segurando(x)$: denota que o braço robótico está segurando o bloco x.

Dessa forma, a situação que representa o estado inicial para o problema da anomalia de Sussman, denotada por S_0 é descrita por:

$Holds(SobreMesa(A), S_0)$
$Holds(SobreMesa(B), S_0)$
$Holds(Sobre(C, A), S_0)$
$Holds(Limpo(B), S_0)$
$Holds(Limpo(C), S_0)$
$Holds(BracoLivre, S_0)$

Faz-se notar que essa não é uma descrição completa da situação, pois somente a informação positiva foi incluída. Essas fórmulas, que são interpretadas como fórmulas da lógica clássica, não permitem tirar qualquer conclusão sobre o que não foi descrito, como por exemplo $\neg Holds(Limpo(A))$. Portanto, as sentenças negativas devem estar presentes para que o exemplo esteja completo, o que foi omitido aqui apenas por conveniência.
O domínio do mundo dos blocos possui quatro ações, descritas abaixo:

1. $Pegar(x)$: ação para que o braço robótico pegue o bloco x que está sobre a mesa;

2. $Soltar(x)$: ação para que o braço robótico solte sobre a mesa o bloco x que ele está segurando;

3. $Desempilhar(x, y)$: ação para que o braço robótico pegue o bloco x que está sobre o bloco y;

4. $Empilhar(x, y)$: ação para que o braço robótico empilhe o bloco x, que está segurando, sobre o bloco y.

Essa versão do domínio do mundo dos blocos é a mesma que foi adotada para os problemas da competição *IPC-2000*. Para a ação empilhar temos os seguintes axiomas de efeito:

$$\forall s[\neg Holds(Segurando(x), Result(Empilhar(x,y), s))]$$
$$\leftarrow Holds(Segurando(x), s) \wedge Holds(Limpo(y), s)$$
$$\forall s[\neg Holds(Limpo(y), Result(Empilhar(x,y), s))]$$
$$\leftarrow Holds(Segurando(x), s) \wedge Holds(Limpo(y), s)$$
$$\forall s[Holds(Limpo(x), Result(Empilhar(x,y), s))]$$
$$\leftarrow Holds(Segurando(x), s) \wedge Holds(Limpo(y), s)$$
$$\forall s[Holds(BracoLivre, Result(Empilhar(x,y), s))]$$
$$\leftarrow Holds(Segurando(x), s) \wedge Holds(Limpo(y), s)$$
$$\forall s[Holds(Sobre(x,y), Result(Empilhar(x,y), s))]$$
$$\leftarrow Holds(Segurando(x), s) \wedge Holds(Limpo(y), s)$$

2.6 O problema do quadro

Partindo da descrição do estado inicial Σ e dos axiomas de efeito Δ, pode-se inferir as proposições que são conseqüências lógicas da execução de uma ação. Pode-se, por exemplo, provar que $Holds(Segurando(C), S_1)$ é verdadeira na situação S_1 resultante da execução da ação $Pegar(C)$ no estado inicial S_0, ou seja:

$$S_1 = Result(Pegar(C), S_0)$$
$$\Sigma' \wedge \Delta \models Holds(Segurando(C), S_1)$$

onde Σ' denota a conjunção de todos os predicados $Holds$ que descrevem o estado S_1. Apesar de muitas das conclusões estarem presentes nas conseqüências lógicas de $\Sigma' \wedge \Delta$, algumas conclusões importantes não estão. No exemplo acima, a execução da ação $Pegar(C)$ não altera o fato do bloco A estar sobre a mesa, e no entanto não é possível inferir esse fato de $\Sigma' \wedge \Delta$, ou seja:

$$\Sigma' \wedge \Delta \nvDash Holds(SobreMesa(A), S_1)$$

Isso ocorre porquê os axiomas de efeito descrevem apenas os fluentes que mudam como resultado da execução de uma ação. Falta, portanto, descrever os fluentes que não mudam, por meio de axiomas que garantam a persistência dos fluentes que não são afetados por cada ação, chamados de axiomas de quadro. Um axioma de quadro deve garantir que se um fluente β era verdadeiro antes da execução de uma ação α, e se β não é afetado por essa ação, então o fluente β deve ser mantido no estado resultante. Por exemplo, um axioma de quadro para a ação $Pegar$ é:

$$\forall s[Holds(SobreMesa(u), Result(Pegar(x), \phi)) \leftarrow Holds(SobreMesa(u), \phi)$$

A conjunção de axiomas de efeito Δ acrescida dos axiomas de quadro será denotada por Δ'. Para um problema que tenha n fluentes e m ações, o número total de axiomas de quadro necessários para garantir a correção da descrição é da ordem de $O(n \times m)$. O problema do quadro (*Frame Problem*) [44] surge com a necessidade de manter-se uma grande quantidade de axiomas de quadro. O problema do quadro também aparece quando há a necessidade de adicionar novas ações ao domínio, o que implica na manutenção do conjunto de axiomas de quadro por meio da adição de novos axiomas, tantos quanto forem os fluentes existentes no domínio. Assim, a preocupação com o problema do quadro é justamente em maneiras de minimizar o número de axiomas de quadro necessários para descrever os fluentes que não são afetados por cada ação do domínio.

2.7 Abordagens lógicas para planejamento

Dada uma formulação de ações e dos estados do mundo, a abordagem mais simples para resolver problemas de planejamento é provar a existência de uma seqüência de ações que alcançam as condições do estado objetivo [13]. Esse processo de resolução ou prova, pode se dar pelo uso de algum sistema automático ou interativo para prova de teoremas.

Provadores de teoremas utilizam processos de resolução, ou algum procedimento de inferência, para provar um conjunto de sentenças lógicas [40].

Para o caso de um problema de planejamento, sendo Γ um objetivo e Σ o estado inicial, o teorema a ser provado assume a seguinte forma:

$$\forall S_0 Holds(\Sigma, S_0) \rightarrow \exists S_k Holds(\Gamma, S_k) \wedge Reachable(S_k, S_0)$$

Ou seja, deve ser provado que existe um estado S_k em que o objetivo Γ é verdadeiro, e que S_k pode ser atingido pela execução de sucessivas ações a partir do estado S_0 [13].

Green [14] foi o primeiro a implementar um sistema de planejamento desse tipo, exprimindo os problemas por meio do cálculo de situações em adição a uma implementação baseada na resolução de um provador teoremas. A grande vantagem do sistema criado por Green, chamado de *QA3*, é a forte correspondência entre a especificação e a implementação. Os mesmos axiomas utilizados na descrição formal do problema de planejamento constituem a base da representação utilizada pelo planejador. Do ponto de vista teórico, o sistema *QA3* foi muito admirado apesar de ter se mostrado muito ineficiente, principalmente em função do problema do quadro. Essa ineficiência da abordagem lógica criou uma forte crença de que um provador automático de teoremas não pode ser o fundamento de um sistema de planejamento para fins práticos.

Em um artigo recente, Shanahan resgata a utilização de abordagens lógicas, usando o cálculo de eventos circunscritivo como formalismo para raciocinar sobre ações e seus efeitos, em substituição ao cálculo de situações. Além disso, ao invés de utilizar processos dedutivos, Shanahan apresenta a programação lógica abdutiva como técnica de prova de teoremas [45]. Barros e Lago mostraram, por meio de comparações empíricas entre implementações de sistemas de planejamento clássico e um planejador abdutivo, que é possível utilizar o formalismo lógico para obter um planejador eficiente [7].

Outra abordagem lógica foi empregada por Kautz & Selman na criação do planejador *Satplan* [26]. O algoritmo *GSat* [42] (sigla para *Greedy local search procedure*) encontra, caso exista, uma atribuição de valores verdade para as variáveis de fórmulas descritas como uma `CNF` (*Conjunctive Normal Form*). Uma `CNF` assume a seguinte

forma:

$$(a_1 \vee a_2 \vee \cdots \vee \neg a_i) \wedge (\cdots) \wedge \cdots \wedge (\neg n_1 \vee n_2 \vee \cdots \vee n_j)$$

Motivados pelo sucesso do algoritmo *GSat*, Kautz & Selman formularam o problema de planejamento como um problema de satisfatibilidade (SAT), engendrando então o planejador *Satplan*, que obteve resultados surpreendentes. O problema da satisfatibilidade é o complementar da dedução, e consiste em encontrar um modelo, ou uma interpretação, para um conjunto de axiomas.

A abordagem de planejamento como SAT também beneficia-se do fato de que diversos pesquisadores de outras áreas contribuem para o aprimoramento dos algoritmos de satisfatibilidade, que podem então ser incorporados aos planejadores [3]. Apenas para citar um outro exemplo, o algoritmo *WalkSat* [41], que é um aprimoramento do algoritmo *GSat*, é ainda mais eficiente que este último, resolvendo fórmulas com vários milhares de variáveis proposicionais.

Enquanto a dedução é um problema complexo (comprovadamente NP-difícil), a área de IA sempre supôs que o problema SAT fosse ainda pior. Realmente, para o caso da lógica de primeira ordem a prova de teoremas é no mínimo semi-decidível, enquanto pode não existir um procedimento completo para encontrar modelos para fórmulas arbitrárias. No caso proposicional, entretanto, parece não haver uma base teórica que afirme que um problema é mais difícil que o outro [26].

Mais detalhes sobre o algoritmo *GSat* e sobre o planejador *Satplan* estão descritos no apêndice A.

2.8 Algoritmos para planejamento clássico

Diversos algoritmos para sintetizar planos tem sido desenvolvidos para resolver os problemas de planejamento clássico. A ineficiência dos provadores de teoremas de primeira ordem disponíveis na época (1970), aliada à dificuldade de resolver o problema do quadro, levaram ao surgimento das abordagens algorítmicas baseadas na busca pelo grafo de estados [23]. Como a busca no espaço de estados mostrou-se inflexível em alguns ca-

[3] A biblioteca de algoritmos de satisfatibilidade Satlib (http://www.satlib.org) é uma ótima fonte de artigos e implementações de algoritmos prontos para o uso.

sos, surgiram os planejadores que efetuam buscas em um espaço de planos parciais. Um algoritmo mais recente, chamado *Graphplan* [3], emprega uma abordagem diferente na busca por planos. A idéia básica é o emprego de um tipo de análise de alcançabilidade para eliminar muitas das combinações e seqüências de ações que não são compatíveis entre si [46]. Vários dos melhores planejadores apresentados na competição *IPC-1998* são baseados na técnica do *Graphplan*. O Capítulo 3 será inteiramente dedicado à apresentação desse planejador e sua mais importante extensão, o planejador *Blackbox*.

2.8.1 Planejamento como busca no espaço de estados

A maneira mais simples de se fazer planejamento é realizando uma busca de força bruta no grafo do espaço de estados [31] sendo que, resolver um problema dessa forma é encontrar um caminho nesse grafo que leva do estado inicial a um estado objetivo [50]. Utilizando a definição 1 pode-se construir o grafo do espaço de estados para um problema. A Figura 2.2 apresenta o grafo do espaço de estados para um problema do mundo dos blocos com 3 blocos. Geralmente as arestas são orientadas, mas no domínio do mundo dos blocos da forma como foi apresentado aqui, todas as ações são inversíveis, razão pela qual duas arestas orientadas foram substituídas por uma única aresta simples de forma a simplificar a figura [50]. Da mesma maneira, foram omitidos nessa figura os estados intermediários em que o braço robótico está segurando um bloco.

A busca pode ser feita de forma progressiva, quando parte do estado inicial em direção ao estado objetivo, ou regressiva, quando parte do estado objetivo em direção ao estado inicial. A complexidade dessa busca depende de dois parâmetros do problema: o fator de ramificação e a profundidade da solução. O fator de ramificação b de um problema pode ser considerado como o número médio de novos estados que podem ser gerados a partir de um dado estado pela aplicação de um operador, e a profundidade d de um problema é o comprimento do caminho solução mais curto entre o estado inicial e o estado objetivo.

A busca em largura gasta tempo $O(b^d)$ para encontrar uma solução de comprimento d, e ocupa espaço $O(b^d)$ pois todos os estados em um determinado nível são mantidos armazenados para que o próximo nível seja construído. A busca em profundidade utiliza espaço linear pois armazena apenas um único caminho, mas necessita de uma

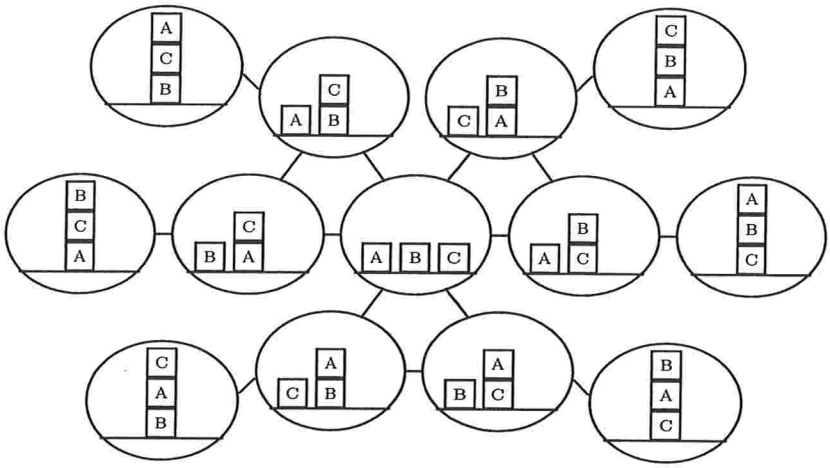

Figura 2.2: Grafo do espaço de estados para o problema no mundo dos blocos com três blocos A, B e C.

profundidade de corte arbitrária para que a busca termine. Se essa profundidade de corte for inferior a profundidade d da solução, o que é raramente conhecido de antemão, não há garantia de encontrar uma solução caso ela exista. Se a profundidade de corte for superior a d a primeira solução encontrada pode não ser ótima. Esses problemas são resolvidos por um algoritmo chamado busca em profundidade iterativa (*Iterative deepening search*). Esse algoritmo efetua uma série de buscas em profundidade, iniciando com uma profundidade de corte unitária, que é incrementada de um (ou mais) a cada iteração até que uma solução seja encontrada. Esse algoritmo encontra uma solução de comprimento d em tempo $O(b^d)$ ocupando espaço $O(d)$. A vantagem desse algoritmo é encontrar uma solução ótima com um custo assintoticamente ótimo em termos de tempo e espaço [31].

A desvantagem dos algoritmos de busca não informada é a ineficiência, o que é resolvido com o uso de algum conhecimento sobre o problema para guiar o processo de busca na escolha correta dos vértices a serem expandidos. Esses métodos de busca que utilizam conhecimento são chamados de busca informada. A fonte de conhecimento mais estudada no contexto de problemas de busca é a avaliação de uma função heurística, que é uma forma computacionalmente eficiente de estimativa do custo para atingir-se um objetivo a partir de um determinado estado. Existe um grande número de algoritmos bem conhecidos que utilizam funções de avaliação heurística para reduzir

a busca, incluindo *hill-climbing*, busca *best-first*, e A^* [31]. O sistema de planejamento *HSP - Heuristic Search Planner*[4] teve um ótimo desempenho na competição *IPC-1998*, enquanto o sistema de planejamento *FF - Fast-Forward* brilhou nas competições *IPC-2000* e *IPC-2002*. O sistema de planejamento *FF* será apresentado com detalhes no capítulo 5.

2.8.2 Planejamento como busca no espaço de planos

Alguns planejadores como o *Prodigy* [10] e o *UCPOP* [38] passaram a realizar buscas não no espaço de estados, mas no espaço de planos [50]. O espaço de planos também pode ser descrito por meio de um grafo, onde os vértices representam planos parciais, ou seja, planos não necessariamente completos, e as arestas representam operações de refinamento do plano, como a adição de um passo ou a imposição de uma restrição.

Um plano parcial é representado por um conjunto de passos, um conjunto de restrições de ordenação que descrevem a ordem em que os passos são executados, e um conjunto de restrições auxiliares que contém informação que auxilia a construção do plano [6]. A busca pelo espaço de planos é um processo mais geral, pois a busca no espaço de estados pode ser vista como um caso especial da busca no espaço de planos, em que o plano parcial, nesse caso o caminho que ainda não leva até o estado objetivo, é estendido pela adição de um elemento primitivo de plano ao final do plano corrente [13]. O processo de busca por esse grafo de planos parciais inicia-se em um vértice que representa um plano nulo, e por meio de sucessivos refinamentos termina quando é obtido um plano completo que solucione o problema. A solução por esse processo de busca não é um caminho entre dois vértices do grafo, mas apenas o vértice que representa um plano completo que soluciona o problema.

De acordo com a maneira como representam os vértices do espaço de planos, planejadores desse tipo podem ser de dois tipos: totalmente ordenados e parcialmente ordenados. Planejadores totalmente ordenados, conforme seu nome, representam planos como uma seqüência totalmente ordenada de ações. Nesse caso, cada ação adicionada ao plano é feita de modo a estar totalmente ordenada em relação as demais ações do plano. Por sua vez, os planejadores parcialmente ordenados evitam o comprometimento prematuro com qualquer restrição, seja de ordem ou de instanciação das variáreis de

um operador. Isso é chamado de comprometimento mínimo, ou seja, são armazenadas na representação do plano apenas as restrições essenciais que resolvem conflitos entre duas ou mais ações do plano.

A Figura 2.3 apresenta um exemplo de um espaço de planos parcialmente ordenados. Os vértices desse espaço representam planos, e as arestas representam operações de refinamento do plano, como a adição de um novo passo ao plano adjacente, ou a imposição de uma restrição, como a ordenação relativa de dois passos.

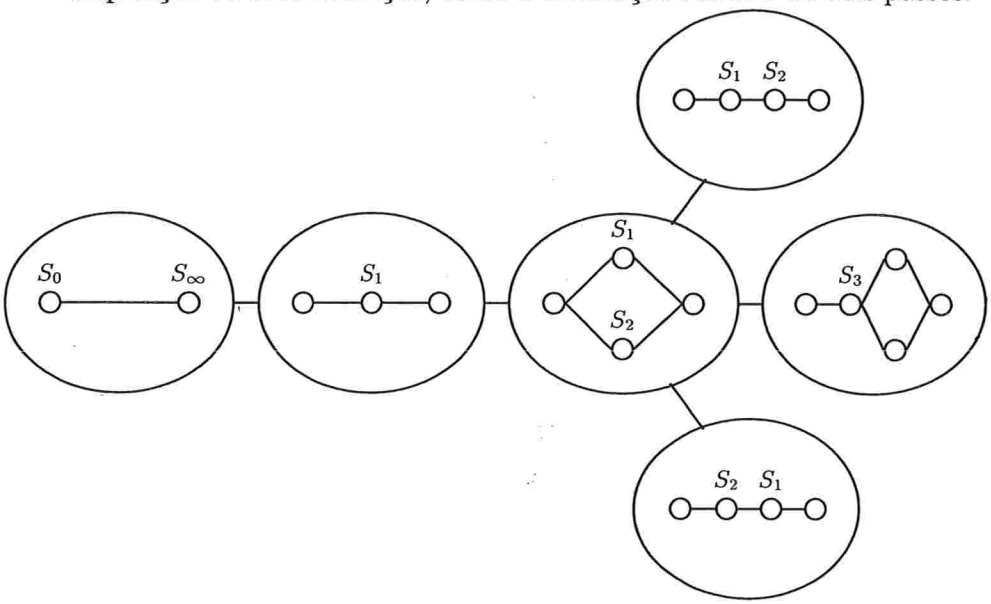

Figura 2.3: Exemplo de um espaço de planos parcialmente ordenados

2.8.3 Planejamento de ordem parcial

POP, acrônimo para Planejador de Ordem Parcial, é um planejador clássico que faz buscas regressivas no espaço de planos parcialmente ordenado. Um plano é representado pela 3-upla $< \mathcal{A}, \mathcal{O}, \mathcal{L} >$, sendo $\mathcal{A}$ um conjunto de ações, $\mathcal{O}$ um conjunto de restrições de ordem sobre as ações de $\mathcal{A}$, e $\mathcal{L}$ é um conjunto de vínculos causais (*causal links*). Um vínculo causal é uma restrição auxiliar composta de três campos: uma ação produtora $\mathcal{A}_p \in \mathcal{A}$, uma ação consumidora $\mathcal{A}_c \in \mathcal{A}$, e uma proposição c, que é um efeito da ação produtora e uma pré-condição da ação consumidora. Um vínculo causal, representado por $\mathcal{A}_p \xrightarrow{c} \mathcal{A}_c$, é adicionado a $\mathcal{L}$ sempre que a ação $\mathcal{A}_p$ é adicionada a $\mathcal{A}$. Um vínculo

causal, que indica o propósito da adição de uma ação ao plano, tem a finalidade de auxiliar na detecção de conflitos entre duas ações do plano.

Um vínculo causal $\mathcal{A}_i \xrightarrow{c} \mathcal{A}_j$ é ameaçado se algum passo $\mathcal{A}_k$ pode ser ordenado entre os passos $\mathcal{A}_i$ e $\mathcal{A}_j$, e $\mathcal{A}_k$ remove a proposição c. A Figura 2.4 apresenta um grafo do espaço de planos onde o vínculo causal $\mathcal{S}_1 \xrightarrow{c} \mathcal{S}_g$ é ameaçado pelo passo $\mathcal{S}_3$. Um plano que contém ameaças é um plano inconsistente, ou seja, se o plano for executado não resolverá o problema de planejamento. Portanto, o planejador deve detectar ameaças e removê-las. Uma maneira de remover ameaças é pela adição de restrições de ordem entre as ações envolvidas, ou seja, a ação $\mathcal{A}_k$ deve ser ordenada antes ou depois do intervalo definido pelo vínculo causal $\mathcal{A}_p \xrightarrow{c} \mathcal{A}_c$. A adição de uma restrição de ordem do tipo $\mathcal{A}_k \prec \mathcal{A}_i$ é chamada de uma demoção, e a restrição $\mathcal{A}_j \prec \mathcal{A}_k$ é uma promoção (*Demotion* e *Promotion*, respectivamente).

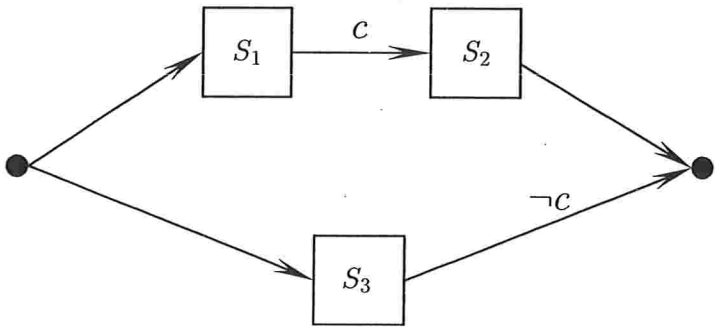

Figura 2.4: Vínculo causal ameaçado

O planejador *POP* inicia com um plano nulo composto de duas ações fictícias, $\mathcal{A} = \{\mathcal{A}_0, \mathcal{A}_\infty\}$, sendo que o efeito da ação $\mathcal{A}_0$ é adicionar o estado inicial do problema e a pré-condição da ação $\mathcal{A}_\infty$ é o estado objetivo. O plano nulo também possui a restrição de ordem $\mathcal{O} = \{\mathcal{A}_0 \prec \mathcal{A}_\infty\}$ e o conjunto de vínculos causais vazio $\mathcal{L} = \emptyset$. A Figura 2.5 apresentada a descrição do algoritmo *POP*.

É importante salientar que o planejador *POP* apresentado é uma simplificação que considera domínios descritos por ações totalmente instanciadas, ou seja, livre de variáveis. Essa é uma prática que simplifica em muito o algoritmo do planejador. Em um planejador que utilize domínios descritos com esquemas de ações, ou seja, ações parametrizadas por variáveis, o plano parcialmente ordenado deve conter um

---

Algoritmo POP $(< \mathcal{A}, \mathcal{O}, \mathcal{L} >, \Gamma, \Delta)$
Entradas:
Um plano parcialmente ordenado $< \mathcal{A}, \mathcal{O}, \mathcal{L} >$
O conjunto de objetivos não satisfeitos Γ
Uma teoria do domínio Δ
Saída:
Falha ou um plano parcialmente ordenado $< \mathcal{A}, \mathcal{O}, \mathcal{L} >$

---

1. Se $\Gamma = \emptyset$, encerrar com $< \mathcal{A}, \mathcal{O}, \mathcal{L} >$
2. Escolher uma proposição $c \in \Gamma$, sendo $\mathcal{A}_c \in \mathcal{A}$ o passo em que c é pré-condição
3. Seja $\mathcal{A}_p$ uma ação que adiciona a proposição c, podendo ser uma ação escolhida em Δ, ou uma ação existente em $\mathcal{A}$ que seja consistentemente ordenada antes de $\mathcal{A}_c$. Se tal ação não existir, retroceder ou encerrar com falha.
4. Seja $\mathcal{L}' = \mathcal{L} \cup \{\mathcal{A}_p \xrightarrow{c} \mathcal{A}_c\}$, $\mathcal{O}' = \mathcal{O} \cup \{\mathcal{A}_p \prec \mathcal{A}_c\}$.
5. Se $\mathcal{A}_p$ é uma nova ação escolhida de Δ, então seja $\mathcal{A}' = \mathcal{A} \cup \{\mathcal{A}_p\}$, e $\mathcal{O}' = \mathcal{O}' \cup \{\mathcal{A}_0 \prec \mathcal{A}_p \prec \mathcal{A}_\infty\}$. Senão, seja $\mathcal{A}' = \mathcal{A}$.
6. Seja $\Gamma' = \Gamma - \{c\}$
7. Se $\mathcal{A}_p$ é uma nova ação escolhida de Δ, então seja $\Gamma' = \Gamma' \cup$ pré-condições de $\mathcal{A}_p$.
8. Para cada ação $\mathcal{A}_k$ que ameaça o vínculo causal $\mathcal{A}_p \xrightarrow{c} \mathcal{A}_c \in \mathcal{L}'$ escolher uma das possíveis restrições de ordem:
   - **Demoção**: Seja $\mathcal{O}' = \mathcal{O}' \cup \{\mathcal{A}_k \prec \mathcal{A}_p\}$
   - **Promoção**: Seja $\mathcal{O}' = \mathcal{O}' \cup \{\mathcal{A}_c \prec \mathcal{A}_k\}$

   Se nenhuma das duas restrições for consistente, encerrar com falha.
9. Recursivamente invocar POP $(< \mathcal{A}', \mathcal{O}', \mathcal{L}' >, \Gamma', \Delta)$.

Figura 2.5: Pseudo código do algoritmo *POP*

conjunto adicional $\mathcal{B}$ de instanciações das variáveis Um planejador desse tipo deve ter um algoritmo capaz de resolver restrições nas unificações das variáveis, por meio da adição ou remoção de uma possível unificação de uma variável do conjunto $\mathcal{B}$.

2.9 A linguagem STRIPS

A maioria dos algoritmos para planejamento clássico descreve estados e operadores em uma linguagem restrita, derivada da lógica de primeira ordem, conhecida como STRIPS [4]. A linguagem STRIPS [9] é uma alternativa ao uso do cálculo de situações para representação de domínios e problemas.

Na linguagem STRIPS, um estado s_t de um modelo, correspondente à uma situação, é descrito pela conjunção de n literais l_i

$$s_t : l_1 \wedge l_2 \wedge \cdots \wedge l_n$$

Essa sintaxe assume que todas as variáveis de um estado são universalmente quantificadas. Diferentemente do cálculo de situações que exige uma representação completa das situações, no STRIPS a descrição de um estado não precisa ser completa pois há o pressuposto de que o agente está em um ambiente inacessível, ou suposição do mundo fechado. Isso significa o mesmo que:

$$\forall l_i \notin s_t \rightarrow \neg l_i$$

Na linguagem STRIPS, o objetivo do agente é uma descrição parcial de um estado do mundo.

Operadores são elementos básicos que representam mapeamentos para as reais ações que um agente pode executar. Os operadores são agrupados em famílias chamadas de esquemas. Um operador Op de um domínio, identificado pelo nome n_j e por uma lista de m argumentos v_k, é descrito por uma 4-upla:

$$Op : (n_j(v_1, v_2, \cdots, v_m), p, Ep, En)$$

onde p é a pré-condição do operador, Ep é a lista de efeitos que o operador adiciona ao

[4] A sigla STRIPS é a abreviação para "STanford Research Institute Problem Solver", um famoso e influente planejador construído nos anos 70 para controlar um robô móvel, afetivamente conhecido como "Shakey".

estado em que é aplicado, e *En* é a lista de efeitos que o operador remove do estado em que é aplicado. Todas as variáveis de um esquema de operadores são universalmente quantificadas. A pré-condição p segue as mesmas regras que a descrição do objetivo do agente, e as listas *Ep* e *En* são conjuntos não ordenados de literais.

Apenas operadores totalmente instanciados podem ser executados pelo agente, ou seja, cujas variáveis foram devidamente instanciadas com alguma constante. Um operador *Op* é aplicável em um estado s_t somente se suas variáveis v_k puderem ser instanciadas de modo que p unifica com uma conjunção de um subconjunto de literais de s_t.

A Figura 2.6 apresenta um exemplo da representação STRIPS do estado inicial e do objetivo para o problema da anomalia de Sussman, bem como as ações para o domínio do mundo dos blocos. As descrições do estado inicial e do estado objetivo correspondem exatamente as configurações dos blocos apresentadas na Figura 2.1. A descrição da ação *Pegar*, por exemplo, indica que para que ela possa ser executada, o braço robótico deve estar livre, ou seja, não pode estar segurando outro bloco. Também é necessário que o bloco a ser pego esteja sobre a mesa, e que não haja outro bloco sobre ele. Como resultados da execução dessa ação, o braço estará segurando o bloco que foi pego, de forma que o braço robótico não estará livre para pegar outro bloco. Da mesma maneira, o bloco que foi pego não estará mais sobre a mesa.

2.10 Outras linguagens

A linguagem STRIPS é amplamente utilizada até hoje, porém, outras linguagens foram criadas para ampliar a expressividade na descrição de domínios e problemas de planejamento. A linguagem ADL [37] (*Action Description Language*) é essencialmente uma reformulação do cálculo de situações como um esquema de operadores, semelhantes aos operadores da linguagem STRIPS com suas listas de adição e remoção. Mesmo sendo mais expressiva que a linguagem STRIPS, ADL ainda é uma linguagem menos expressiva que a lógica de primeira ordem [38]. A linguagem ADL possibilita uma expressividade maior na descrição de domínios por meio das seguintes características:

Estado Inicial :

SobreMesa(A) ∧ SobreMesa(B) ∧ Sobre(C, A) ∧ Limpo(B) ∧ Limpo(C) ∧ BracoLivre

Objetivo :

SobreMesa(C) ∧ Sobre(B, C) ∧ Sobre(A, B)

Teoria do Domínio :

Operador$_1$: Pegar(x)
Pré-condição: Limpo(x) ∧ SobreMesa(x) ∧ BracoLivre
Lista adição: { Segurando(x) }
Lista remoção: { SobreMesa(x), Limpo(x), BracoLivre }

Operador$_2$: Soltar(x)
Pré-condição: Segurando(x)
Lista adição: { SobreMesa(x), Limpo(x), BracoLivre }
Lista remoção: { Segurando(x) }

Operador$_3$: Desempilhar(x, y)
Pré-condição: Sobre(x, y) ∧ Limpo(x) ∧ BracoLivre
Lista adição: { Segurando(x), Limpo(y) }
Lista remoção: { Limpo(x), Sobre(x, y), BracoLivre }

Operador$_4$: Empilhar(x, y)
Pré-condição: Segurando(x) ∧ Limpo(y)
Lista adição: { Limpo(x), Sobre(x, y), BracoLivre }
Lista remoção: { Segurando(x), Limpo(y) }

Figura 2.6: Representação STRIPS do estado inicial e do estado objetivo para o problema da anomalia de Sussman, além das ações do domínio do mundo dos blocos

- **pré-condições disjuntivas**: diferentemente da linguagem STRIPS, que só permite pré-condições expressas como uma conjunção de literais positivos, qualquer uma das parcelas da disjunção pode definir a aplicabilidade de um operador;

- **efeitos condicionais**: são utilizados para descrever ações cujos efeitos são dependentes do contexto em que a ação é executada. Por exemplo, efeitos condicionais podem ser utilizados para descrever um operador que modifica as proposições que indicam a localização de um conjunto objetos de objetos colocados sobre um caminhão, após o caminhão tê-los levado para outro local. Nesse exemplo, o efeito de modificar a localização afetará condicionalmente todos os objetos que estavam sobre o caminhão.

- **quantificação universal**: na linguagem STRIPS, existe quantificação universal implícita apenas sobre as pré-condições dos operadores. A linguagem ADL estende o conceito para os efeitos dos operadores. O exemplo dado para os efeitos condicionais só é possível com o uso da quantificação universal sobre os efeitos, ou seja, o efeito de modificar a localização vale para todos os objetos que estão colocados sobre o caminhão.

A linguagem PDDL [34] foi proposta para a competição *IPC-1998*, com o intuito de universalizar as descrições dos domínios e problemas propostos. Devido a sua pretensão de padronizar a descrição de domínios, e tendo em vista que nem todos os planejadores são capazes de tratar determinados aspectos de uma descrição, a linguagem PDDL foi fatorada em um sub-conjunto de características, chamadas de requerimentos. Inicialmente, os requerimentos eram os seguintes: ações descritas no estilo da linguagem STRIPS, ações que necessitam das extensões da linguagem ADL, axiomas de domínio, e ações hierárquicas, compostas de um conjunto de ações primitivas. Dessa forma, os planejadores são agrupados pelos tipos de requerimentos que são capazes de tratar sendo que, durante as competições, todos os planejadores de um mesmo grupo recebem como entrada a mesma descrição de domínio e problema. Para a competição *IPC-2002*, cujo foco foi voltado para os planejadores capazes de lidar com problemas envolvendo variáveis numéricas e restrições temporais [35], a linguagem PDDL foi estendida. Assim, a linguagem PDDL 2.1, que neste trabalho será chamada simplesmente de PDDL, possui três níveis: o primeiro nível corresponde ao nível proposicional e ADL, o segundo nível incrementa o primeiro nível com variáveis numéricas e a habilidade de testar e alterar seus valores instantaneamente, e o terceiro nível contém ações durativas, cujo tempo para serem executadas não é instantâneo e cujos efeitos acontecem algum tempo após a execução das mesmas.

2.11 Considerações finais

Neste capítulo foi apresentado o problema de planejamento, sua descrição como um problema de busca em um grafo do modelo de estados, e sua descrição como um problema de inferência lógica utilizando o cálculo de situações. Também foram apresentadas algumas abordagens algorítmicas para resolver problemas de planejamento, o planejamento

como uma busca no espaço de planos e o planejamento como uma busca heurística. Finalmente, as linguagens comumente utilizadas para descrever domínios e problemas foram mostradas. Todos esses conceitos fundamentais da área de planejamento em IA serão de grande importância para a compreensão dos demais capítulos.

No próximo capítulo é apresentada com detalhes a estrutura chamada de grafo de planejamento, construída pelo planejador *Graphplan* para o processo de busca por um plano solução, e utilizada por diversos sistemas de planejamento desenvolvidos como uma extensão do mesmo. O sistema *Graphplan* em sí não é necessário para este trabalho, mas o grafo de planejamento é fundamental para os planejadores *FF* e *Metric-FF*, que foram implementados como parte deste trabalho e serão apresentados no capítulo 5.

Capítulo 3

O planejador *Graphplan*

Nesse capítulo é apresentada a estrutura de dados chamada de grafo de planejamento, construída e utilizada pelo planejador *Graphplan* para o processo de busca por um plano solução. A estrutura simples do grafo de planejamento permite facilmente identificar conflitos entre pares de proposições em um possível estado, e entre escolhas de ações. A importância do grafo de planejamento para esse trabalho se deve: (i) às extensões do planejador *Graphplan*; (ii) o seu uso no cálculo da função heurística do planejador implementado nesse trabalho, que será apresentado no capítulo 5.

O planejador *Graphplan*, criado por Blum & Furst [3], é visto por alguns pesquisadores como um dos mais interessantes desenvolvimentos recentes na área de planejamento em IA pelos seguintes motivos:

- *Graphplan* é um algoritmo simples e elegante que engendra um planejador extremamente eficiente;
- A representação utilizada pelo *Graphplan*, chamada de grafo de planejamento, forma a base para uma das mais bem sucedidas codificações de problemas de planejamento em problemas de satisfatibilidade proposicional;
- A partir do grafo de planejamento é possível obter informações sobre conflitos entre ações de forma direta. Por esse motivo, muitos sistemas de planejamento foram criados com base no planejador *Graphplan*, dentre os quais estão os mais bem sucedidos planejadores das competições de planejamento;

- Algumas propostas estenderam o conceito do grafo de planejamento para com isso resolver problemas mais complexos, como por exemplo, problemas que envolvem recursos, variáveis numéricas, e tempo.

O planejador *Graphplan* alterna entre duas fases: a construção de um grafo de planejamento e a extração de uma solução desse grafo.

3.1 O grafo de planejamento

A construção do grafo de planejamento foi originalmente proposta para domínios cuja descrição é baseada na linguagem STRIPS [9]. O grafo de planejamento é um grafo orientado de níveis, ou camadas. Esse grafo tem alternadamente níveis de proposições e níveis de ações. Os níveis pares do grafo representam proposições por meio de um vértice, sendo o nível zero do grafo formado pelas proposições do estado inicial do problema de planejamento. Os níveis ímpares do grafo representam, também por meio de vértices, todas as ações aplicáveis, ou seja, aquelas cujas pré-condições são satisfeitas pelas proposições do nível anterior. Um tipo especial de ação chamada $no - op$ é utilizada para a persistência das proposições pelos níveis do grafo, resolvendo de forma algorítmica o problema do quadro [44]. Os vértices contendo as ações em um nível i são conectados por arestas de pré-condição às proposições do nível $i - 1$, e por arestas de adição e remoção às proposições do nível $i + 1$. As arestas sempre terão a direção de um nível menor para um nível maior, indicando a ordem de execução das ações em um plano solução.

Para ilustrar o processo de construção do grafo de planejamento, será utilizado o domínio do jantar surpresa proposto por Weld [51]. No problema apresentado na Figura 3.1 juntamente com esse domínio, um marido deve preparar um jantar surpresa para sua esposa, além de embrulhar um presente e se livrar do lixo da cozinha. A esposa está dormindo, e para não estragar a surpresa, ela não deve ser acordada com qualquer forma de barulho até que as três tarefas tenham sido realizadas. Esse domínio é utilizado aqui pelo fato de ser o domínio mais simples e que possui todos os tipos de relação de exclusão mútua, que serão apresentadas na seção 3.2.

A Figura 3.2 apresenta o grafo de planejamento para o problema do jantar surpresa.

Estado Inicial :

Lixo $\wedge$ MaosLimpas $\wedge$ Silencio

Objetivo :

Jantar $\wedge$ Presente $\wedge$ $\neg$ Lixo

Teoria do Domínio :

Operador$_1$: Cozinhar
Pré-condição: MaosLimpas
Lista adição: { Jantar }

Operador$_2$: Embrulhar
Pré-condição: Silencio
Lista adição: { Presente }

Operador$_3$: CarregarLixo
Pré-condição: Lixo
Lista remoção: { Lixo, MaosLimpas }

Operador$_4$: TriturarLixo
Pré-condição: Lixo
Lista remoção: { Lixo, Silencio }

Figura 3.1: Representação STRIPS do estado inicial e do estado objetivo para o problema do jantar surpresa, além das ações do domínio

Note que as arestas não estão orientadas, assim como não foram representados os vértices de ações $no - op$. Isso foi feito apenas para simplificar a figura.

3.2 Relações binárias de exclusão mútua

Como veremos adiante, o plano solução para o problema de planejamento será extraído do próprio grafo de planejamento. Entretanto, na forma como o grafo de planejamento é construído, um plano solução pode ter como escolhas todas as ações ou proposições de cada um dos níveis. Qualquer representação consistente de um estado jamais pode conter duas proposição p e $\neg p$, pois p e $\neg p$ são mutuamente exclusivas. O mesmo princípio é aplicável à ações. Dessa forma, durante a construção do grafo de planejamento, após a adição de cada par de níveis de ações e proposições, o *Graphplan* efetua uma análise de relações de exclusão mútua entre todas as combinações de dois vértices de um mesmo nível do grafo.

Definição 4 *Duas proposições p e q, ambas de um mesmo nível i do grafo de planejamento, são mutuamente exclusivas se não existir um plano válido que as contenha*

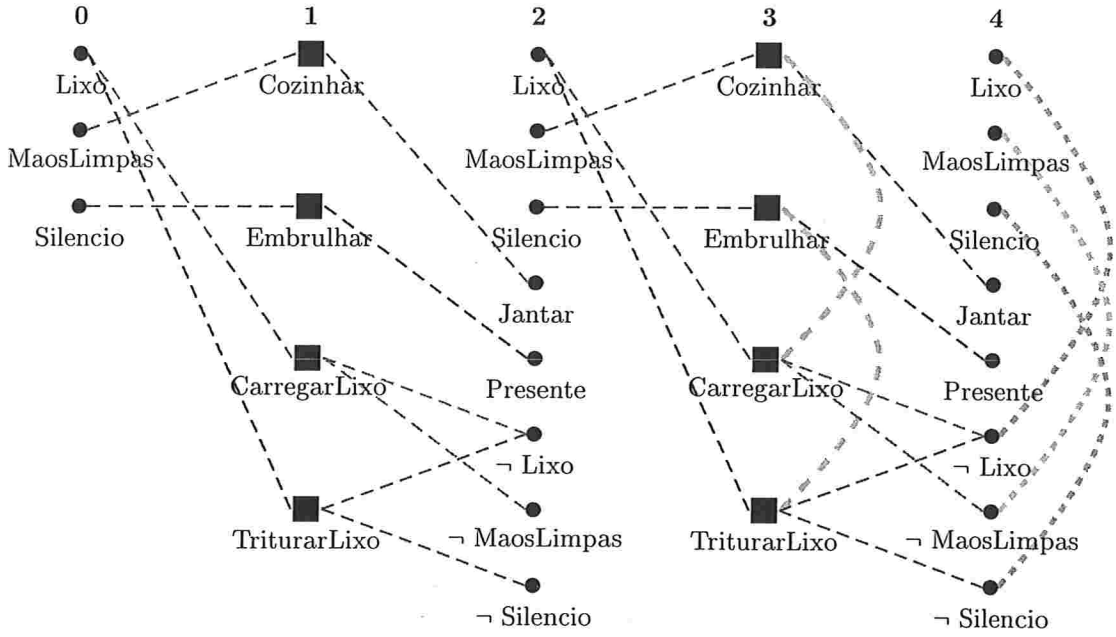

Figura 3.2: **Grafo de Planejamento** para o domínio do jantar surpresa

verdadeiras no nível i.

Da definição 4 temos duas situações para exclusão mútua de proposições: se uma proposição é a negação da outra, ou se todas as maneiras de adicionar uma proposição p são mutuamente exclusivas às maneiras de adicionar a proposição q. Esse último conceito é apresentado na definição 5, lembrando-se que as ações $no - op$ também podem ser utilizadas para adicionar proposições.

Definição 5 *Qualquer proposição é válida em um nível i se e somente se é um efeito de adição ou remoção de alguma ação existente no nível* $i - 1$.

Definição 6 *Duas ações* A_1 *e* A_2*, ambas de um mesmo nível i do grafo de planejamento, são mutuamente exclusivas se não existir um plano válido que as contenha no nível i.*

Deve-se notar que da definição 6 não há implicação que impeça que ambas as ações pertençam ao plano solução, mas apenas que essas ações não podem pertencer ao mesmo nível em que são exclusivas. As relações de exclusão mútua, também chamadas de *mutex*, de acordo com Weld [51], podem ser de quatro tipos [1]:

efeitos inconsistentes acontece quando uma ação adiciona uma proposição que é removida por outra ação;

interferência ocorre quando uma ação remove uma proposição que faz parte da pré-condição de outra ação;

necessidades conflitantes ocorre sempre que uma proposição $p_{A,1}$, da pré-condição de uma ação A_1, e uma proposição $p_{A,2}$, da pré-condição de uma ação A_2, são marcadas como mutuamente exclusivas no nível de proposições anterior àquele que contém essas ações;

suporte inconsistente acontece quando duas proposições p e q não puderem ser verdadeiras simultaneamente, devido a existência de exclusão mútua entre todas as combinações de ações que adicionam essas proposições.

Na Figura 3.2 são apresentas as relações de exclusão mútua entre as ações e as proposições dos níveis 3 e 4 por meio de arcos entre os vértices do mesmo nível. É importante salientar que esses arcos não são arestas do grafo, mas apenas indicações das relações de exclusão.

Durante o processo de busca por um plano solução, os arcos que indicam as proposições e ações exclusivas auxiliam o *Graphplan* a evitar caminhos de busca que apresentam conflitos. Isso portanto reduz o número de nós visitados durante essa busca, garantindo dessa forma a eficiência do algoritmo. Essa é a razão da preocupação com esse tipo de raciocínio sobre interações entre as ações.

Na Figura 3.3 é apresentado o algoritmo que contrói o grafo de planejamento para um problema.

[1] Neste trabalho Weld estendeu o trabalho inicial de Blum & Furst, e caracterizou melhor as relações de exclusão mútua

---

Algoritmo GrafoPlanejamento (Δ, Σ, Γ)
Entradas:
Uma descrição de um domínio Δ
Uma conjunção de proposições do estado inicial Σ
Uma conjunção de proposições do estado objetivo Γ
Saída:
Falha ou um grafo de planejamento $G(V, A)$

---

1. **G** $\longleftarrow$ *criaNivelZeroGrafo* (Σ)
2. Criar um conjunto de arestas de exclusao mútua A_e vazio
3. **i** $\longleftarrow$ 0
4. Se *objetivoSatisfeito* (**G**, **i**, A_e , Γ), então
5. Encerrar retornando o grafo **G**
6. Fim Se
7. **G** $\longleftarrow$ *expandeGrafo* (**G**, **i** , Δ)
8. Se o nível **i** + **1** do grafo não possui vértices, então
9. Encerrar retornando **falha**
10. Fim Se
11. $A_e \longleftarrow$ *relacoesExclusaoAcoes* (G, i, A_e)
12. $A_e \longleftarrow$ *relacoesExclusaoProposicoes* (G, i, A_e)
13. **i** $\longleftarrow$ **i** + 2
14. Voltar para 4

Figura 3.3: Pseudo código do algoritmo que constrói o grafo de planejamento para um problema

3.3 A busca por uma solução

A partir de um grafo de planejamento cujo último nível contenha todas as proposições do estado objetivo, um plano solução pode ser encontrado por meio de uma busca de encadeamento reverso por esse grafo. Na Figura 3.4 é apresentado esse algoritmo de busca.

Algoritmo buscaSolucaoGrafo $(\Gamma, G(V, A), E_m)$
Entradas:
Uma conjunção de proposições do estado objetivo Γ
Um grafo de planejamento $G(V, A)$ com k níveis
Um conjunto de arestas de exclusão mútua entre ações E_m
Saída:
Falha ou um plano solução Π

---

1. Criar um plano parcialmente ordenado Π vazio
2. $P_k \longleftarrow \Gamma$
3. **i** $\longleftarrow$ k
4. Se $P_i = \emptyset$, encerrar com Π
5. $P_{i-2} \longleftarrow \emptyset$
6. Para cada proposição p do conjunto P_i, fazer
7. Escolher uma ação a_{i-1} do nível $i-1$ do grafo $G(V, A)$ que adiciona p e que não seja exclusiva a outras ações escolhidas nesse mesmo nível, utilizando o conjunto E_m para essa verificação
8. Se não for possível encontrar a_{i-1}, retroceder nas escolhas anteriores
9. Se não for possível retroceder, encerrar retornando **falha**
10. Se $i > 2$, adicionar as pré-condições da ação a_{i-1} ao conjunto P_{i-2}
11. adicionar a ação a_{i-1} ao plano Π
12. Fim Para
13. $i \longleftarrow$ i - 2
14. Voltar para 4

Figura 3.4: Pseudo código do algoritmo que extrai uma solução de um grafo de planejamento

O algoritmo de busca por uma solução inicia com um plano parcialmente ordenado vazio e com um conjunto de objetivos do último nível k do grafo composto pelos objetivos do problema de planejamento. O processo de busca inicia no nível $i = k$ do grafo e a condição de parada é não ter proposições objetivo para esse nível. Tal condição ocorre quando não há proposições no objetivo Γ do problema, ou quando um plano Π

para o problema é encontrado. Para cada nível i da busca, são escolhidas ações não exclusivas que adicionam cada um dos objetivos desse nível, encerrando com falha caso não seja possível encontrar tais ações. Para garantir a minimalidade do plano solução, a escolha das ações deve, sempre que possível, ser iniciada por ações do tipo $no - op$. Para cada ação escolhida, suas pré-condições são adicionadas ao conjunto de objetivos do nível $i - 2$, exceto para o nível $i = 2$ pois, as ações do nível $i = 1$ são as aplicáveis no estado inicial e portanto, suas pré-condições já estão inicialmente satisfeitas. Essa exceção irá satisfazer a condição de parada na próxima iteração. Cada ação escolhida, bem como seu nível, é adicionada ao plano solução Π, sendo que o nível de cada ação define a ordenação parcial do plano solução. A busca é iterativamente repetida para o nível de proposições anterior até que alguma condição de parada seja alcançada.

3.4 O algoritmo *Graphplan*

Conforme dito no início desse capítulo, o planejador *Graphplan* alterna entre a construção do grafo de planejamento e a extração de uma solução desse grafo. A partir de um grafo de planejamento cujo último nível contenha todas as proposições do estado objetivo, um plano solução pode ser encontrado por meio de uma busca de encadeamento reverso por esse grafo. Com base nessas duas fases, já descritas nas seções anteriores, na Figura 3.5 é apresentado o algoritmo *Graphplan*.

Dessa maneira, em cada iteração do algoritmo descobre-se um plano válido ou prova-se que não pode existir um plano solução que tenha a quantidade de níveis presente no grafo de planejamento. Blum & Furst [3] provaram que o algoritmo é correto e completo.

No apêndice B é apresentada a especificação completa do algoritmo utilizado para a construção do grafo de planejamento. Tal especificação foi utilizada pelo autor durante a implementação dos sistemas *Blackbox* e *FF*, a serem apresentados a seguir.

Algoritmo Graphplan (Δ, Σ, Γ)
Entradas:
Uma descrição de um domínio Δ
Uma conjunção de proposições do estado inicial Σ
Uma conjunção de proposições do estado objetivo Γ
Saída:
Falha ou um plano solução Π

---

1. Seja $G(V, A)$ um grafo de planejamento inicialmente nulo
2. Seja E_m um conjunto de arestas de exclusão mútua inicialmente vazio
3. Criar o nível 0 do grafo de planejamento contendo um vértice para cada uma das proposições do estado inicial do problema
4. Iterativamente montar os níveis i de ações e $i+1$ de proposições, até que todas as proposições do estado objetivo estejam presentes no último nível sem que haja exclusão mútua entre elas
5. A partir do grafo de planejamento, realizar uma busca por um plano solução, encerrando se encontrar um plano válido, expandindo o nível seguinte no caso contrário

Figura 3.5: Pseudo código do algoritmo *Graphplan*

3.5 O planejador *Blackbox*

As estratégias utilizadas pelos planejadores *Satplan* e *Graphplan* compartilham de várias características comuns. Se analisados independentemente, nenhum dos dois é substancialmente superior ao outro no que diz respeito ao desempenho. Os dois algoritmos apresentam as seguintes semelhanças: ambos possuem duas fases bem distintas sendo que, primeiro criam uma estrutura proposicional (o *Graphplan* constrói um grafo de planejamento, enquanto o *Satplan* codifica o problema como uma CNF), para só então efetuarem uma busca por essa estrutura.

Estudando os detalhes do grafo de planejamento, determinou-se que ele emprega (de maneira limitada) uma propagação binária negativa. Isso significa a seguinte regra:

dado: $\{\neg p \vee \neg q\}, \{p \vee r\}$
infere-se: $\{\neg q \vee r\}$

A sentença $\{\neg p \vee \neg q\}$ significa o mesmo que dizer que as proposições p e q são mutuamente exclusivas. Foi então apresentado em um artigo de Kautz, McAllester & Selman que o grafo de planejamento obtido pelo planejador *Graphplan* poderia ser facilmente compilado em uma CNF [25]. Essa CNF apresenta exatamente as mesmas características descritas na maneira de efetuar a busca pelo grafo de planejamento por meio da resolução de um problema CSP dinâmico. Para tal, basta criar uma variável proposicional para cada um dos possíveis valores que uma variável do problema CSP pode assumir.

A partir dessas observações Kautz & Selman criaram um novo sistema de planejamento que combina as melhores características dos planejadores *Graphplan* e *Satplan*. Esse planejador, chamado de *Blackbox* [24], trabalha em três fases:

1. Um problema de planejamento especificado na linguagem STRIPS é convertido em um grafo de planejamento
2. O grafo de planejamento é transformado em uma CNF
3. A CNF é resolvida por algum algoritmo de satisfatibilidade proposicional, como o *GSat* ou *WalkSat*

Blackbox foi um dos planejadores de melhor desempenho na competição *IPC-1998* [27].

3.6 Considerações finais

Nesse capítulo foi apresentada a estrutura do grafo de planejamento, e a maneira como essa estrutura é utilizada para inferir a existência de conflitos na escolha de ações. Também foi apresentado o planejador *Blackbox*, uma das mais importantes extensões do planejador *Graphplan*. O uso do grafo de planejamento será utilizado também no capítulo 5 para a descrição dos planejadores *FF* e *Metric-FF*.

No próximo capítulo são apresentados os aspectos típicos de problemas de escalonamento, que podem ser incorporados nas descrições de problemas de planejamento.

Dois problemas da competição *IPC-2002*, que envolvem aspectos de escalonamento, e que serão analisados no capítulo 6, também serão abordados.

Capítulo 4

Planejamento & Escalonamento

Esse capítulo apresenta três aspectos típicos de problemas de escalonamento, recursos, variáveis numéricas e tempo, bem como as formas de representar tais aspectos em problemas tipicamente de planejamento. Também são descritos os problemas de planejamento que envolvem variáveis numéricas, relevantes para esse trabalho.

Em problemas reais de planejamento existe um conjunto de restrições, que podem ser, por exemplo, temporais ou de utilização de recursos limitados. Essas restrições devem ser satisfeitas para que um plano válido seja obtido por um sistema de planejamento. Isso requer a resolução de um problema de escalonamento, que deve garantir a disponibilidade de recursos e a satisfação de todas as restrições do problema [12]. O que torna esses problemas particularmente difíceis é o fato de serem problemas de otimização, que envolvem tempo contínuo, recursos, variáveis numéricas, e uma complexa mistura de escolha de ações e decisões de ordenação. Quando considerados separadamente, os problemas de escolha de ações e ordenação são, em geral, resolvidos por sistemas de planejamento, enquanto os problemas de satisfação de restrições temporais e do uso de recursos são resolvidos por sistemas de escalonamento. Smith [46] apresenta um vasto levantamento das técnicas de planejamento em IA e de escalonamento, de forma a salientar as diferenças e similaridades entre essas duas tarefas. A necessidade de resolver problemas que envolvem planejamento e escalonamento vem motivando o surgimento de diversas arquiteturas de extensão dos sistemas de planejamento que os permita tratar aspectos típicos dos problemas de escalonamento.

4.1 Planejamento e escalonamento

As pesquisas sobre planejamento em IA sempre focaram os problemas onde existe uma dependência em cascata nas escolhas das ações, e ocorrem complexas interações entre as ações. Em um problema de planejamento, usualmente o número de tarefas ou ações necessárias para atingir o objetivo é desconhecido. Em contrapartida, as pesquisas em escalonamento focam problemas grandes nos quais existe uma pequena escolha de ações, mas cujo problema de ordenação e atribuição de recursos das mesmas é de grande dificuldade [46].

O conceito comum de escalonamento na comunidade de IA é de que se trata de um caso especial de planejamento, nos quais todas as ações já estão previamente escolhidas, reduzindo o problema à determinação de uma ordenação apropriada [46]. Dean & Kambhampati [6] definem planejamento como a escolha de **o que** fazer, e escalonamento como a escolha de **quando** e **como** fazer. Infelizmente, essa é uma trivialização do problema de escalonamento [46].

O que distingue os tratamentos dados aos problemas de escalonamento pela comunidade de IA dos tratamentos da comunidade de Pesquisa Operacional (PO), é que IA tende a focar em representações e técnicas genéricas que cobrem uma gama de diferentes tipos de problemas de escalonamento. Em contraste a isso, os trabalhos de PO, normalmente, focam o desenvolvimento de técnicas otimizadas para classes específicas de problemas [46], como o *Flow Shop* e *Job Shop*.

Muitas das abordagens para problemas de planejamento clássico não permitem a representação, ou o raciocínio sobre recursos, variáveis numéricas ou tempo contínuo. Essas abordagens também ignoram a otimização de algum parâmetro. No entanto, várias abordagens foram propostas para estender as técnicas de planejamento clássico de forma a tratar: recursos e variáveis numéricas, critérios de otimização ou restrições de tempo. A avaliação clássica para um plano é baseada puramente no comprimento do plano, ou seja, o número de ações existentes no mesmo. Com a introdução de restrições sobre recursos e tempo, a utilidade de um plano é influenciada pelo custo e consumo de recursos e tempo, o que demanda meios mais complexos para a avaliação da qualidade de um plano [33]. Um exemplo de sistema que utiliza essas abordagens para resolver um problema real de grande importância é o *HSTS* [47] , responsável pela geração de

escalonamentos de pequenas observações para o telescópio espacial Hubble.

Esse trabalho restringe-se aos sistemas de planejamento que tratam apenas de problemas que envolvem restrições sobre recursos e sua otimização.

4.2 Recursos

Na comunidade de *IA*, o termo *recurso* tem sido utilizado para se referir a um número diferente de aspectos, desde equipamentos compartilháveis, como um grupo de máquinas idênticas em uma fábrica, até quantidades contínuas que são consumíveis e/ou renováveis (produzíveis), como combustível. Não existe uma definição única para o termo recurso, que normalmente refere-se a algum objeto ou quantidade, cuja utilização restringe de alguma forma a estrutura de um plano [33]. De acordo com Maria Fox [33], há dois tipos de recursos: os que restringem o número de ações que podem ser executadas em paralelo (por exemplo, máquinas ou processadores), e os que restringem o número de ações que podem ser livremente executadas durante todo o ciclo de vida de um plano (por exemplo, combustível ou capacidade de memória). O segundo tipo de recursos, que podem ser consumíveis ou produzíveis, são denominados por Smith [46] como **variáveis numéricas**, nomenclatura adotada nesse trabalho.

Recursos tem recebido relativamente pouca atenção da comunidade de *IA*. Em escalonamento, recursos são normalmente classificados como de capacidade unitária ou de múltipla capacidade. Recursos de capacidade unitária não apresentam dificuldade para a maioria dos formalismos de planejamento. Se duas ações necessitam de um mesmo recurso em um mesmo instante, há um conflito entre a escolha simultânea dessas duas ações nesse instante (por exemplo, a existência de um único braço robótico). Nesse caso, não deve ser permitida a sobreposição no uso do recurso. Recursos de múltipla capacidade apresentam muito mais dificuldades, por duas razões: é mais difícil identificar grupos de ações com conflito de recursos, e o número de maneiras possíveis de resolver os conflitos cresce exponencialmente com o número de ações envolvidas. Esses são alguns problemas que surgem quando se trata de recursos de múltipla capacidade em problemas de planejamento. Como resultado, as técnicas utilizadas em planejamento para resolver esse tipo de restrição de recurso carregam uma semelhança grande com as técnicas que foram desenvolvidas em escalonamento.

Os planejadores mais conhecidos capazes de resolver conflitos na utilização de recursos são o *O-Plan* e o *IxTeT* [32]. Existem também algumas propostas que estendem os sistemas baseados no *Graphplan* e no *Satplan* [46].

4.3 Variáveis numéricas

Como dito na seção anterior, nesse trabalho, recursos que podem ser consumíveis ou produzíveis, são denominados **variáveis numéricas**. Várias técnicas de planejamento discutidas anteriormente foram estendidas para permitir valores reais ou variáveis numéricas. A distância de um agente até seu objetivo e a quantidade de combustível que um robô móvel tem disponível são exemplos de variáveis numéricas. Uma proposta de representação é aumentar as pré-condições e efeitos dos operadores STRIPS com restrições de igualdade e desigualdade envolvendo expressões aritméticas e funções. Para vários propósitos, essa representação é suficiente. Entretanto, se duas ações podem afetar a mesma variável numérica simultaneamente, torna-se necessário descrever como a quantidade varia durante um curso de ação. A presença de variáveis numéricas e restrições sobre estas pode causar grandes dificuldades no processo de planejamento. Uma das maneiras de lidar com essas dificuldades é ignorar as restrições métricas até que os valores das variáveis sejam precisamente conhecidos. Nesse momento, a restrição é verificada, e se não for satisfeita, ocorre um retrocesso no processo de busca [46] (*backtracking*).

Alguns planejadores foram construídos para tentar checar a consistência de restrições em conjuntos métricos mesmo antes que todas as variáveis sejam conhecidas. Isso pode envolver raciocínio matemático de dificuldade que varie desde o raciocínio sobre funções complexas até análises limitadas a sub-conjuntos de restrições métricas consistindo de igualdades e desigualdades.

Alguns planejadores tem utilizado técnicas de Programação Linear (PL) para manipular esse tipo de restrições. Um esforço mais recente para lidar com variáveis numéricas é apresentado pelo planejador *LPSAT* [52]. O planejador *LPSAT* usa técnicas de PL em conjunto com as técnicas de satisfatibilidade proposicional (SAT).

Outra abordagem para manipular variáveis numéricas é representar o problema de

planejamento como uma mistura de Programação Linear e Inteira (PLI). Essa proposta é apresentada por Kautz & Walser [28] em um sistema chamado *ILP-Plan*, que estende o planejamento SAT pela representação de problemas de planejamento como PLI. Apesar de ainda não ser competitiva com os métodos de planejamento SAT, essa é uma abordagem muito nova e que recebeu, até o momento, pouca atenção da comunidade de planejamento [46].

Um problema que surge com o tratamento de variáveis numéricas é a possibilidade de que exista um número infinito de ações proposicionais (*i.e.*, infinitas instâncias de variáveis numéricas). Por exemplo, considerando a ação de reabastecimento, onde a quantidade é uma função contínua da duração. Isso corresponde a um número infinito das possíveis ações totalmente instanciadas, o que simplesmente não pode ser manipulado pela estrutura dos planejadores SAT.

4.4 Abordagens integradas para planejamento e escalonamento

Considerando as diferentes propostas de extensão de sistemas de planejamento apresentadas, existem fundamentalmente três arquiteturas para a resolução de problemas de planejamento e escalonamento:

Planejamento e escalonamento estratificados A abordagem usualmente adotada pela comunidade de escalonamento é separar planejamento de escalonamento. A escolha das ações é feita antes, para então o problema de escalonamento resultante ser resolvido.

Planejamento e escalonamento intercalados Nesta abordagem é possível intercalar a escolha de ações com as decisões de ordenação. Tipicamente, a escolha de ações é feita, sendo que os conflitos (entre ações, ou no uso de recursos e variáveis numéricas) são resolvidos em outros passos que impõem restrições às ações escolhidas.

Planejamento e escalonamento homogêneos Essa abordagem, inspirada no planejamento como um problema de satisfatibilidade, representa os aspectos de pla-

nejamento e de escalonamento do problema em um único formalismo, por exemplo, inequações lineares para programação inteira. Desta maneira, é possível resolver todo o problema por um único algoritmo de propósito geral.

Por muitos anos, existiu um grande abismo entre as técnicas de planejamento e as técnicas de escalonamento. Como resultado, muito do trabalho em planejamento clássico não se aplicava à problemas reais de escalonamento, como por exemplo, o problema da espaçonave. Entretanto, existem alguns sinais encorajadores de que essa situação pode estar mudando. Muitas das extensões descritas na seção anterior prometem construir uma ponte que una definitivamente essas duas comunidades.

4.4.1 O planejador *ZENO*

O planejador *ZENO* [39] estabeleceu um marco na tentativa de unir o raciocínio sobre variáveis numéricas ao problema de planejamento. A linguagem utilizada para especificar os domínios e problemas no planejador *ZENO* é uma extensão do ADL. As restrições sobre variáveis numéricas são especificadas como pré-condições de ações enquanto equações sobre variáveis numéricas são descritas como efeitos de ações que podem ser de três tipos diferentes: equações de consumo, produção e atribuição de variáveis numéricas. Objetivos podem ser de dois tipos: objetivos lógicos (objetivos segundo a linguagem ADL para planejamento clássico) e objetivos sobre restrições métricas (conjunto de inequações). Essa linguagem tem sido adotada pela maioria dos trabalhos mais recentes de integração de planejamento e escalonamento.

O sistema é formado pela integração de um conjunto de algoritmos para a resolução de sub-problemas específicos. Durante o planejamento, o sistema pode adicionar novos sub-objetivos, que podem ser dos dois tipos descritos acima. Assim como no *POP*, o algoritmo executa uma busca regressiva para satisfazer os objetivos lógicos enquanto que os objetivos sobre restrições métricas são resolvidos por algoritmos de PO. Equações lineares são resolvidas por eliminação de Gauss, inequações lineares são resolvidas pelo algoritmo *Simplex*. Eventuais equações não-lineares são postergadas até que se tornem lineares. Ações não são consideradas instantâneas, portanto o tempo de execução deve ser especificado por meio de intervalos, sendo que o sistema possui um terceiro processo para resolver os conflitos temporais e definir a ordenação dos passos. O processo de

planejamento termina quando não existirem objetivos a serem satisfeitos e quando o conjunto de restrições estiver livre de conflitos.

O planejador *ZENO* é considerado um sistema ultrapassado porque é baseado em um planejador de ordem parcial, que é pouco eficiente quando comparado aos planejadores baseados nas abordagens mais recentes, como o *Satplan*, *Graphplan*, e busca local. O planejador *ZENO* é um sistema de arquitetura intercalada, pois seu algoritmo alterna entre a escolha de ações e a satisfação das restrições métricas e temporais.

4.4.2 O planejador *IPP*

O planejador *IPP* [29] surgiu inicialmente como como uma extensão do planejador *Graphplan*. Essencialmente, *IPP* utiliza um processo de construção de um grafo de planejamento para domínios descritos na linguagem ADL, de maneira muito semelhante a empregada pelo *Graphplan*. Mais tarde, o planejador *IPP* foi estendido para permitir o raciocínio sobre variáveis numéricas [30]. Inspirado no planejador *ZENO*, *IPP* utiliza a mesma linguagem para a descrição dos domínios e problemas, que também possui característica comuns às da linguagem ADL. Com uma proposta algorítmica totalmente aplicada ao problema de planejamento com restrições de variáveis numéricas, *IPP* trabalha em duas etapas: a criação de um grafo de planejamento juntamente com um mapa da utilização de recursos por intervalo de tempo (RTM, *Resource Time Map*), e um processo de busca por uma solução no grafo e no mapa RTM. Assim como o *Graphplan*, *IPP* utiliza uma forma de raciocínio sobre as relações de exclusão mútua existentes no grafo de planejamento e no RTM. Essas relações são levadas em conta pelo processo de busca de um plano solução através do grafo de planejamento. O planejador *IPP* é um sistema de arquitetura intercalada, pois seu algoritmo utiliza duas estruturas distintas para raciocinar sobre a construção de um plano: o grafo de planejamento para a escolha de ações e o RTM para a verificação da satisfação das restrições métricas.

4.4.3 O planejador *LPSAT*

O planejador *LPSAT* [52] também utiliza uma arquitetura semelhante a apresentada por *ZENO*. *LPSAT* aproveita a codificação do problema de planejamento como um problema SAT, e se beneficia do desempenho dos algoritmos de satisfatibilidade pro-

posicional para resolver a escolha das ações. A escolha das ação dispara a geração de restrições sobre as variáveis numéricas, que são resolvidas por um algoritmo de PL. Como também alterna entre a escolha de ações e a satisfação das restrições métricas, o *LPSAT* pode ser considerado um sistema de arquitetura intercalada. Qualquer insucesso na tentativa de satisfazer as restrições sobre as variáveis numéricas faz com que uma nova satisfação para as sentenças proposicionais seja tentada. No aspecto do raciocínio temporal o *LPSAT* é menos representativo que o *ZENO*. Assim como no planejador *Satplan*, a noção de tempo para o planejador *LPSAT* assume apenas valores inteiros positivos presentes nos fluentes que indicam a execução de uma ação, diferentemente do uso da representação de intervalos de tempo para os passos do plano adotado pelo *ZENO*. Por utilizar algoritmos de propósito geral eficientes para resolver os problemas SAT e o problema PL, o planejador *LPSAT* tem um desempenho muito melhor que os dos planejadores *ZENO* e *IPP*.

4.4.4 O planejador *ILP-Plan*

O planejador *ILP-Plan* (*Integer Linear Programming*) utiliza as idéias propostas por Vossen, Ball, Lotem & Nau [48] para traduzir todo o problema de planejamento para um problema de PI, e resolvê-lo com um algoritmo apropriado para PI. Vossen *et.al.* [48] apresentam uma maneira de codificar apenas o problema de planejamento, representado pela linguagem STRIPS, como um conjunto de inequações lineares inteiras. Desenvolvido por Kautz & Walser [28], *ILP-Plan* estende a idéia inicial de Vossen et.al. ao propor uma maneira de adicionar o raciocínio sobre variáveis numéricas. A codificação das sentenças da lógica também é um pouco diferente, pois *ILP-Plan* utiliza o algoritmo *WSat* [49] para resolver o problema de PI, que é uma generalização do algoritmo *WalkSat*. Em função do uso do *WSat*, *ILP-Plan* aproveita as idéias do *Satplan* para codificação das sentenças proposicionais, que já apresentavam uma resolução bastante eficiente por parte do *WalkSat*. Essa foi uma tentativa de aproveitar o melhor das duas abordagens, *Satplan* e PI. Apesar de ainda não ser competitiva com os métodos de planejamento SAT, essa é uma abordagem muito nova, e que recebeu pouca atenção da comunidade de planejamento até o momento [46]. O planejador *ILP-Plan* é um sistema de arquitetura homogênea, pois transforma todo o problema em um único modelo de PI.

4.5 Domínios com variáveis numéricas

Os planejadores descritos na seção anterior foram propostos no final da década 90, indicando assim o interesse crescente da comunidade de planejamento em resolver problemas que envolvam aspectos de escalonamento, isto é, problemas mais próximos do mundo real.

Em função dos avanços alcançados pelos planejadores que participaram das competições *IPC-1998* e *IPC-2000* e, com a intenção de atrair o maior número possível de pesquisadores, a competição *IPC-2002* teve como objetivo apresentar problemas de planejamento que envolvessem raciocínio sobre tempo e variáveis numéricas [35]. Dessa forma, os problemas deixaram de ter um caráter estritamente artificial e começaram a ter desafios muito mais próximos dos encontrados em problemas práticos do mundo real. Para descrever esses novos problemas, a linguagem PDDL foi estendida de forma a possibilitar três níveis de problemas: o primeiro nível corresponde ao nível proposicional e ADL, o segundo nível incrementa o primeiro nível com variáveis numéricas e a habilidade de testar e alterar seus valores instantaneamente, e o terceiro nível incrementa o nível anterior com ações durativas, cujo tempo para serem executadas não é instantâneo e que tem efeitos que acontecem em um determinado instante de tempo após sua execução.

Dois problemas dessa competição foram selecionados para esse trabalho, e estão descritos nas próximas sub-seções.

4.5.1 O problema do telescópio espacial

O problema do telescópio espacial, batizado na competição de **Satellite** (http://www.dur.ac.uk/d.p.long foi inspirado em um problema real descrito por Smith [46]. O problema consiste de um conjunto de satélites de observação, cada qual com um conjunto de instrumentos que suportam alguns modos de observação, como fotográfico, raios-X, infravermelho, etc. Os objetivos do problema são descritos como uma quantidade de observações a serem realizadas de fenômenos espaciais, como planetas, galaxias, supernovas, etc. Além do fenômeno espacial, uma observação de um objetivo deve especificar o modo em que tal observação será realizada, como observação fotográfica do planeta Júpiter, observação

de raios-X da supernova K6, etc.

Para realizar uma observação objetivo, um satélite deve primeiramente possuir um instrumento que suporte o modo na qual a observação deve ser realizada. Antes de efetuar a observação, o instrumento deverá ser ligado e calibrado, o que exige que o satélite seja direcionado para um alvo de calibragem desse instrumento. Só após esse procedimento o satélite poderá ser direcionado para o objetivo de modo a obter a observação do mesmo.

Problemas para a versão STRIPS podem conter mais de um satélite capaz de realizar uma mesma observação, sendo que um planejador que escolher o satélite cujo instrumento já está calibrado pode encontrar planos melhores. Para descrever problemas nesse domínio, são utilizados oito fluentes a saber:

1. $on_board(i, s)$: denota que o satélite s possui o instrumento i;
2. $supports(i, m)$: denota que o instrumento i suporta o modo m de operação;
3. $pointing(s, d)$: denota que o satélite s está apontado para a direção d;
4. $power_avail(s)$: denota que o suprimento de energia do satélite s está disponível, ou seja, não está sendo utilizado por nenhum instrumento;
5. $power_on(i)$: denota que o instrumento i está ligado e utilizando o suprimento de energia de seu satélite;
6. $calibrated(i)$: denota que o instrumento i está calibrado;
7. $have_image(d, m)$: denota que uma imagem da direção d foi obtida utilizando o modo m;
8. $calibration_target(i, d)$: denota que a direção d é um alvo para a calibragem do instrumento i.

O domínio do satélite possui cinco ações, descritas abaixo:

1. $turn_to(s, d_new, d_prev)$: ação para que o satélite s, que está apontado para a direção d_prev se movimente de modo a apontar para a direção d_new;

2. $switch_on(i, s)$: ação para que o instrumento i do satélite s seja ligado, utilizando o suprimento de energia do satélite;

3. $switch_off(i, s)$: ação para que o instrumento i do satélite s seja desligado, disponibilizando o suprimento de energia do satélite;

4. $calibrate(s, i, d)$: ação para que o instrumento i do satélite s, que está apontado para o alvo de calibragem d, seja calibrado;

5. $take_image(s, d, i, m)$: ação para que o modo m do instrumento i do satélite s obtenha uma imagem da direção d, para a qual o satélite está apontado.

Problemas para a versão numérica estendem os problemas da versão STRIPS com a inclusão de seis variáveis numéricas, a saber:

1. $data-capacity(s)$: indica a capacidade de dados do satélite s;

2. $data(d, m)$: indica a capacidade de dados necessária para a obtenção de uma imagem no modo m do alvo d;

3. $slew-time(a, b)$: indica a quantidade de combustível necessária para a movimentação de um satélite, que está apontado para o alvo a, passar a apontar para o alvo b;

4. $data-stored$: indica a capacidade total de dados armazenados por todos os satélites existentes em um problema;

5. $fuel(s)$: indica a quantidade de combustível disponível no satélite s;

6. $fuel-used$: indica a quantidade total de combustível utilizada por todos os satélites existentes em um problema, durante as ações de movimentação.

As variáveis numéricas $data-capacity(s)$, $data(d, m)$, $slew-time(a, b)$, e $fuel(s)$ são parametrizadas pelo conjunto de satélites, modos, e alvos existentes em um problema. As três primeiras dessas quatro variáveis são na verdade constantes pois, nenhuma ação do domínio altera os seus valores. Já as variáveis $data-stored$ e $fuel-used$ não afetam a aplicabilidade de nenhuma ação do domínio e, destinam-se apenas para uso no critério de otimização.

A escolha do satélite que realizará cada observação e a ordem em que elas serão executadas tem um forte impacto na quantidade de combustível utilizada pelos satélites, e na quantidade de observações que poderão ser concluídas. Esse problema é considerado complexo por três razões:

1. só pode ser utilizado um instrumento por vez;
2. cada instrumento precisa ser calibrado antes do uso, a menos que sucessivas observações sejam executadas com o mesmo instrumento;
3. os giros utilizam combustível, cuja quantidade é limitada.

O objetivo final é maximizar o retorno científico da missão, ou pelo menos, utilizar o combustível de maneira inteligente. Esse problema é muito semelhante ao apresentado em outro trabalho [47], que descreve com detalhes a solução desenvolvida para o problema do telescópio espacial Hubble.

No apêndice C são apresentadas as descrições PDDL completas dos domínios do satélite para as versões STRIPS e numérica, bem como um exemplo de problema para cada uma dessas versões.

4.5.2 O problema do robô de exploração

O problema do robô de exploração, que na competição recebeu o nome de **Rovers** (http://www.dur.ac.uk/d.p.long/IPC/domains.html), consiste de um ou mais robôs equipados com instrumentos para análise de solo, análise de rocha, e obtenção de imagens. Os objetivos dos problemas nesse domínio são descritos como uma série de análises e imagens a serem obtidas de localidades distintas. Para que possa realizar um objetivo, um robô deve primeiramente possuir um instrumento adequado, para então movimentar-se até a localidade onde a análise ou imagem será obtida. No caso da obtenção de uma imagem, a câmera do robô deve ser devidamente calibrada antes de ser utilizada, o que exige que o robô esteja situado em um local com visibilidade para um alvo de calibragem desse instrumento. Após obter uma análise ou uma imagem, um robô deve deslocar-se para uma localidade próxima a uma estação-base a fim de enviar a essa estação os dados obtidos, cumprindo então com o objetivo.

Problemas para a versão STRIPS podem conter mais de um robô capaz de obter uma mesma análise ou imagem, sendo que um planejador que escolher o robô cujo instrumento já está calibrado pode encontrar planos melhores. Para descrever problemas nesse domínio, são utilizados vinte e três fluentes, a saber:

1. $at(r, x)$: denota que o robô r está na localidade x;
2. $at_lander(l, x)$: denota que a estação-base l está na localidade x;
3. $can_traverse(r, x, y)$: denota que o robô r pode se deslocar da localidade x para a y;
4. $equipped_for_soil_analysis(r)$: denota que o robô r está equipado para análise de solo;
5. $equipped_for_rock_analysis(r)$: denota que o robô r está equipado para análise de rocha;
6. $equipped_for_imaging(r)$: denota que o robô r está equipado para obtenção de imagens;
7. $empty(s)$: denota que o compartimento de armazenagem s está vazio;
8. $have_rock_analysis(r, x)$: denota que o robô r tem uma análise de rocha da localidade x;
9. $have_soil_analysis(r, x)$: denota que o robô r tem uma análise de solo da localidade x;
10. $full(s)$: denota que o compartimento de armazenagem s está lotado;
11. $calibrated(c, r)$: denota que a câmera c do robô r está calibrada;
12. $supports(c, m)$: denota que a câmera c suporta o modo m de operação;
13. $visible(x, y)$: denota que a localidade y é visível a partir da localidade x, para fins de deslocamento de um robô de um localidade à outra;
14. $have_image(r, x, m)$: denota que o robô r tem uma imagem no modo m da localidade x;

15. *communicated_soil_data*(x): denota que os dados da análise de solo da localidade x foram enviados à estação-base;

16. *communicated_rock_data*(x): denota que os dados da análise de rocha da localidade x foram enviados à estação-base;

17. *communicated_image_data*(x, m): denota que os dados da imagem no modo m da localidade x foram enviados à estação-base;

18. *at_soil_sample*(x): denota que a localidade x possui uma amostra de solo;

19. *at_rock_sample*(x): denota que a localidade x possui uma amostra de rocha;

20. *visible_from*(x, y): denota que a localidade y é visível a partir da localidade x, para fins de calibragem e obtenção de imagens;

21. *store_of*(s, r): denota que o robô r tem um compartimento de armazenagem s;

22. *calibration_target*(c, x): denota que a localidade x é um alvo para a calibragem da câmera c;

23. *on_board*(c, r): denota que o robô r está equipado com a câmera c.

O domínio do robô de exploração possui nove ações, descritas abaixo:

1. *navigate*(r, x, y): ação para que o robô r, que está situado na localidade x se movimente para a localidade y;

2. *sample_soil*(r, s, x): ação para que o robô r armazene em seu compartimento s uma análise de solo da localidade x;

3. *sample_rock*(r, s, x): ação para que o robô r armazene em seu compartimento s uma análise de rocha da localidade x;

4. *drop*(r, s): ação para que o robô r retire de seu compartimento s a análise que estiver armazenando;

5. *calibrate*(r, c, x, y): ação para que o robô r calibre sua câmera c por meio do alvo de calibragem x, visível a partir da localidade y onde este se encontra;

6. *take_image*(r, x, y, c, m): ação para que o robô r obtenha uma imagem do alvo y, utilizando a câmera c no modo m, visível a partir da localidade x onde este se encontra;

7. *communicate_soil_data*(r, l, x, y, z): ação para que o robô r, situado na localidade y, envie à estação-base l, situada na localidade z, os dados de uma amostra de solo da localidade x;

8. *communicate_rock_data*(r, l, x, y, z): ação para que o robô r, situado na localidade y, envie à estação-base l, situada na localidade z, os dados de uma amostra de rocha da localidade x;

9. *communicate_image_data*(r, l, x, m, y, z): ação para que o robô r, situado na localidade y, envie à estação-base l, situada na localidade z, os dados de uma imagem obtida no modo m da localidade x;

Problemas para a versão numérica estendem os problemas da versão STRIPS com a inclusão de duas variáveis numéricas, a saber:

1. *energy*(r): indica a quantidade de energia do robô r;

2. *recharges*: indica a quantidade total de recargas de energia efetuadas pelos robôs.

A variável numérica *energy*(r) é parametrizada pelos robôs existentes em um problema. Problemas para a versão numérica também incluem uma ação adicional, *recharge*(r, x), que recarrega o robô r a partir de energia solar obtida na localidade x. O fluente adicional *in_sun*(x) denota que a localidade x está sob insolação, o que permite que a energia de um robô seja recarregada nessa localidade.

No apêndice C são apresentadas as descrições PDDL completas dos domínios do robô de exploração para as versões STRIPS e numérica, bem como um exemplo de problema para cada uma dessas versões.

4.6 Considerações finais

Nesse capítulo foram apresentados os aspectos típicos de problemas de escalonamento, ou seja, recursos, variáveis numéricas e tempo. Foram discutidas as três arquiteturas possíveis para a construção de sistemas que resolvam problemas de planejamento e escalonamento: estratificada, intercalada e homogênea. Foi feito, ainda, um resumo dos planejadores conhecidos até a competição *IPC-2002*, que tratam de aspectos de escalonamento para algumas classes de problemas. Também foram apresentados dois domínios de interesse para esse trabalho, o domínio do satélite e o domínio do robô de exploração, introduzidos na competição *IPC-2002* os quais envolvem recursos e variáveis numéricas.

No próximo capítulo é apresentado o planejador *FF* e sua extensão, o planejador *Metric-FF*, capaz de resolver problemas de planejamento que envolvem recursos e variáveis numéricas.

Capítulo 5

O planejador *FF*

Nesse capítulo são apresentados os algoritmos utilizados pelos sistemas de planejamento *FF*, que resolve apenas problemas do tipo STRIPS, e *Metric-FF*, que resolve problemas que envolvem recursos e variáveis numéricas. Esses dois planejadores foram implementados como parte desse trabalho.

Dentre os cinco planejadores totalmente automáticos premiados na competição *IPC-2000*, quatro eram baseados na idéia de uma busca heurística. Dentre esses planejadores de busca heurística, o planejador *FF* é o que teve o melhor desempenho, o que repetiu-se na competição *IPC-2002*. O planejador *Metric-FF*, uma extensão do planejador *FF* para resolver problemas que envolvem recursos e variáveis numéricas, também teve bom desempenho entre os planejadores que resolvem problemas com variáveis numéricas.

O planejador *FF* [18], acrônimo para *Fast Forward*, é um planejador que avalia uma função heurística específica para o problema de planejamento e, dessa forma, orienta um algoritmo de busca *Hill-Climbing* reforçado. A heurística empregada pelo *FF* é um aprimoramento da heurística do planejador *HSP* [4], que resgatou a idéia de buscas heurísticas aplicadas à sistemas de planejamento. O planejador *Metric-FF* [21] estende a função heurística do *FF*, permitindo com isso a resolução de problemas que envolvem recursos e variáveis numéricas.

5.1 O algoritmo *Hill-Climbing* reforçado

O planejador *FF* e sua extensão *Metric-FF*, resolvem um problema de planejamento através de uma busca *Hill-Climbing* reforçada pelo espaço de estados. O algoritmo *Hill-Climbing* reforçado efetua uma busca local pelo espaço de estados, orientada por uma função heurística $h(s)$. A Figura 5.1 descreve esse algoritmo.

Algoritmo Hill-Climbing-reforçado (Δ, Σ, Γ)
Entradas:
uma teoria do domínio Δ
um estado inicial Σ
um conjunto de objetivos Γ
Saída:
falha ou um plano solução para o problema de planejamento

---

1. Inicializa o plano solução como um plano vazio;
2. $S \longleftarrow \Sigma$
3. Enquanto $h(S) \neq 0$, fazer
4. efetuar uma busca em largura por um estado S' com $h(S') < h(S)$
5. Se não encontrou um estado S', encerrar com falha
6. adicionar as ações no caminho até S' ao plano solução
7. $S \longleftarrow S'$
8. Fim fazer
9. Encerrar com o plano solução

Figura 5.1: Pseudo código do algoritmo *Hill-Climbing* reforçado

Alguns controles adicionais são necessários para que o algoritmo *Hill-Climbing* reforçado tenha um bom desempenho. A busca em largura realizada pelo algoritmo *Hill-Climbing* reforçado para encontrar um estado mais próximo do estado objetivo $(S' \mid h(S') < h(S))$, deve ser limitada por uma profundidade de corte. Essa profundidade de corte permite que o algoritmo encerre normalmente, com falha como resultado, caso o problema não tenha uma solução. Sem a profundidade de corte, a

busca em largura pode não ter fim. Infelizmente, o uso de uma profundidade de corte implica na não completude do algoritmo, dependendo do problema de planejamento a que esse é submetido.

A manutenção de uma lista dos estados já visitados durante a busca em largura também é fundamental para o desempenho do algoritmo *Hill-Climbing* reforçado, pois dessa forma, ignora-se qualquer ramo da árvore de busca que passe por algum estado que já tenha sido previamente considerado. Isso é especialmente útil em espaços de busca que apresentam circuitos.

O processo de busca em largura pode adotar dois tipos de controle de parada: encerrar no momento em que é encontrado um estado de menor heurística ou, encerrar apenas depois de expandir todos os ramos de uma determinada profundidade da árvore de busca. O primeiro tipo de controle garante maior velocidade ao processo de busca, enquanto o segundo garante que o estado escolhido pelo processo de busca é o de melhor heurística, garantindo também o plano solução de menor tamanho.

5.2 A heurística do *FF*

A heurística empregada pelo planejador *FF* é um aprimoramento da heurística do planejador *HSP* [4], que avalia o tamanho, em quantidade de ações, de uma solução para uma versão relaxada do problema de planejamento. Dada uma tarefa de planejamento, descrita por uma 3-upla $< \Delta, \Sigma, \Gamma >$, a versão relaxada dessa mesma tarefa de planejamento é descrita pela 3-upla $< \Delta', \Sigma, \Gamma >$. A Seção 2.9 descreve um operador STRIPS de um domínio Δ como a 4-upla:

$$Op : (n_j(v_1, v_2, \cdots, v_m), p, Ep, En)$$

Os operadores do domínio de uma tarefa relaxada ignoram os efeitos de remoção existentes nos operadores da tarefa de planejamento original, de forma que, Δ' é definido como:

$$\Delta' = \{(n_j(v_1, v_2, \cdots, v_m), p, Ep, \emptyset) \quad | \quad (n_j(v_1, v_2, \cdots, v_m), p, Ep, En) \in \Delta\}$$

Ao ignorar as listas de remoção dos operadores do domínio, elimina-se a existência de conflitos no planejamento de ações para a tarefa relaxada.

A heurística do planejador *HSP* é obtida por meio da computação de uma aproximação grosseira do número de ações de um plano para uma tarefa de planejamento relaxada.

Definição 7 *Uma seqüência de ações é dito um plano relaxado de uma tarefa de planejamento P se e somente se ela resolve a tarefa relaxada P' obtida a partir de P*

A função heurística do planejador *HSP* assume que cada átomo do objetivo é satisfeito independentemente dos demais. Por esse motivo, essa heurística ignora possíveis interações positivas entre duas ações o que, para alguns domínios, faz com que o resultado da computação seja muito maior do que a verdadeira distância entre o estado avaliado e o objetivo.

Para corrigir os desvios na heurística do planejador *HSP*, a heurística do planejador *FF* é obtida a partir de um grafo de planejamento [3] para a tarefa relaxada. A vantagem no uso do grafo de planejamento sobre o processo empregado pelo planejador *HSP* é que, devido à estrutura do grafo, a busca por uma solução para a tarefa relaxada também resolve as interações positivas ignoradas pelo *HSP*. Isso se deve ao fato de que as ações que adicionam mais de um átomo do estado objetivo são considerados apenas uma vez, melhorando consideravelmente a qualidade na computação da heurística. Funções heurísticas admissíveis são por natureza otimistas, pois nunca superestimam o custo de uma solução e, por conseqüência, são mais informativas que heurísticas não-admissíveis. De fato, a heurística do planejador *HSP* é não-admissível, enquanto a heurística do planejador *FF* é admissível, o que justifica a superioridade desse planejador.

O uso do grafo de planejamento em tarefas relaxadas tem algumas particularidades importantes. O conjunto de ações não tem lista de remoção, portanto o grafo de planejamento gerado não possui proposições negadas. Como conseqüência, esse grafo também não contém qualquer relação de exclusão mútua entre pares de proposições e entre pares de ações, pois essas relações só ocorrem quando existem conflitos entre proposições e suas respectivas negações. Assim sendo, qualquer busca por uma solução

pelo grafo da tarefa relaxada, como visto na Seção 3.3, não tem pontos de escolha, uma vez que os retrocessos na busca estão associados à existência de relações de exclusão entre duas escolhas possíveis. Isso significa que a busca por uma solução para a tarefa de planejamento relaxada tem complexidade polinomial, conforme foi provado por Hoffmann [17]. É importante lembrar que o processo de construção do grafo de planejamento também tem complexidade polinomial, conforme provado por Blum & Furst [3]. Esse é o motivo pelo qual a heurística do planejador *FF* é tão eficiente, o que ficará patente no Capítulo 7 com a análise do desempenho de uma implementação desse planejador.

Na Figura 5.2 é apresentado o grafo de planejamento para a versão relaxada do problema da anomalia de Sussman, apresentado na Figura 2.1, partindo do estado inicial desse problema. Com o intuito de simplificar a figura, as arestas não estão orientadas, assim como não foram representados os vértices de ações $no - op$. Além disso, foi utilizada uma descrição de domínio que não possui o braço robótico, e que por essa razão, tem apenas duas ações: $mover(x, y, z)$ e $mover_da_mesa(x, y)$. Com apenas dois níveis de ações, o grafo relaxado para esse problema tem presentes no nível 4 todas as proposições do estado objetivo, conforme pode ser verificado na figura. A busca por uma solução para a versão relaxada do problema teria como escolhas as três ações circuladas na figura.

Utilizando o grafo de planejamento, considerando como estado inicial um estado s da árvore de busca do algoritmo *Hill-Climbing*, obtém-se o plano $\langle o_0, \ldots, o_{m-1} \rangle$ para a sub-tarefa relaxada $< \Delta', s, \Gamma >$, onde o_i é o conjunto de ações escolhidas em paralelo no nível i do grafo de planejamento, e m é o primeiro nível de proposições contendo todos os átomos do objetivo. O resultado da função heurística do planejador *FF* é obtido pela computação do comprimento da linearização dessa solução para a tarefa relaxada, de acordo com a seguinte equação:

$$h(S) = \sum_{i=0,\cdots,m-1} |o_i| \tag{5.1}$$

Caso não seja possível a extração de uma solução desse grafo, o valor da função heurística para o estado S é considerado infinito, o que indica que é impossível atingir o objetivo partindo de S. Como exemplo, a heurística do estado inicial do problema

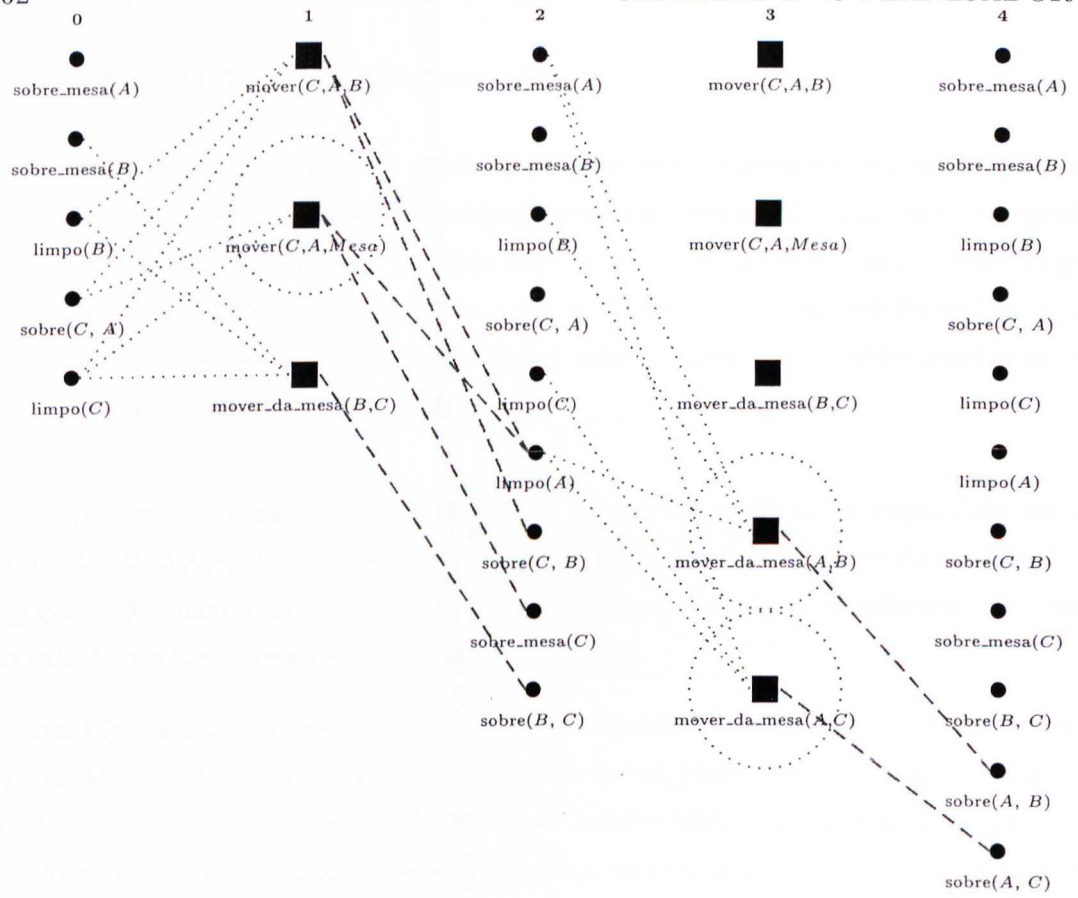

Figura 5.2: Grafo de planejamento para a versão relaxada do problema da anomalia de Sussman

apresentado na Figura 5.2 é igual a 3, já que, conforme citado, o processo de busca por uma solução para a tarefa relaxada encontra um plano composto de três ações. Esse é exatamente o tamanho de um plano solução para esse problema, o que significa que, para esse caso, a heurística do planejador *FF* calcula exatamente a quantidade de ações necessárias para solucionar o problema.

Duas otimizações importantes são empregadas pelo planejador *FF* durante o processo de busca da solução para a tarefa de planejamento relaxada, com o objetivo de minimizar o comprimento dessa solução:

1. Sempre que existir uma ação do tipo $no-op$ que adicione uma proposição p, essa ação é escolhida primeiro, antes da escolha de uma ação real, o que garante a

minimalidade da solução encontrada;

2. Durante a construção do grafo de planejamento é armazenado, junto com cada vértice de ação, o número de proposições da pré-condição dessa ação. Ao escolher uma ação para adicionar um sub-objetivo p, o critério de escolha quando há mais de uma opção é pela ação que tiver o menor número de pré-condições, o que minimiza o número de proposições a serem satisfeitas no nível imediatamente anterior do grafo de planejamento.

Para aumentar ainda mais a eficiência no uso do grafo de planejamento para a avaliação da função heurística, o planejador *FF* implementa uma versão otimizada do algoritmo de construção do grafo de planejamento, por meio da reutilização do mesmo no cálculo da heurística em diferentes estados [17].

Na Figura 5.2 são apresentados os tempos gastos pelos planejadores que participaram da competição *IPC-2000* para encontrar um plano solução para problemas no domínio do Mundo dos Blocos. O eixo x do gráfico indica a quantidade de blocos no problema e o eixo y indica o tempo gasto para encontrar uma solução. Apesar de não ter obtido sucesso em todos os problemas, motivo da sua curva de desempenho não ser contínua, o planejador *FF* foi sempre o mais eficiente, e dentre os planejadores que fazem busca local, foi o que resolveu o maior número de problemas.

5.3 Transformação de esquemas de operadores

O planejador *FF* exige que todas as ações do domínio estejam totalmente instanciadas, ou seja, livres de variáveis. Isso porquê existe um custo computacional alto associado ao processo de unificação dos esquemas de operadores, o que poderia comprometer o desempenho do planejador se a unificação para todas os esquemas de ação ocorresse para cada estado a ser expandido durante o processo de busca em largura. Para otimizar o desempenho do planejador *FF* como um todo, antes que o algoritmo *Hill-Climbing* comece o processo de busca por um plano solução, deve ocorrer uma transformação dos operadores descritos por esquemas em ações totalmente instanciadas, ou seja, ações proposicionais. Na Figura 5.4 é descrito esse algoritmo.

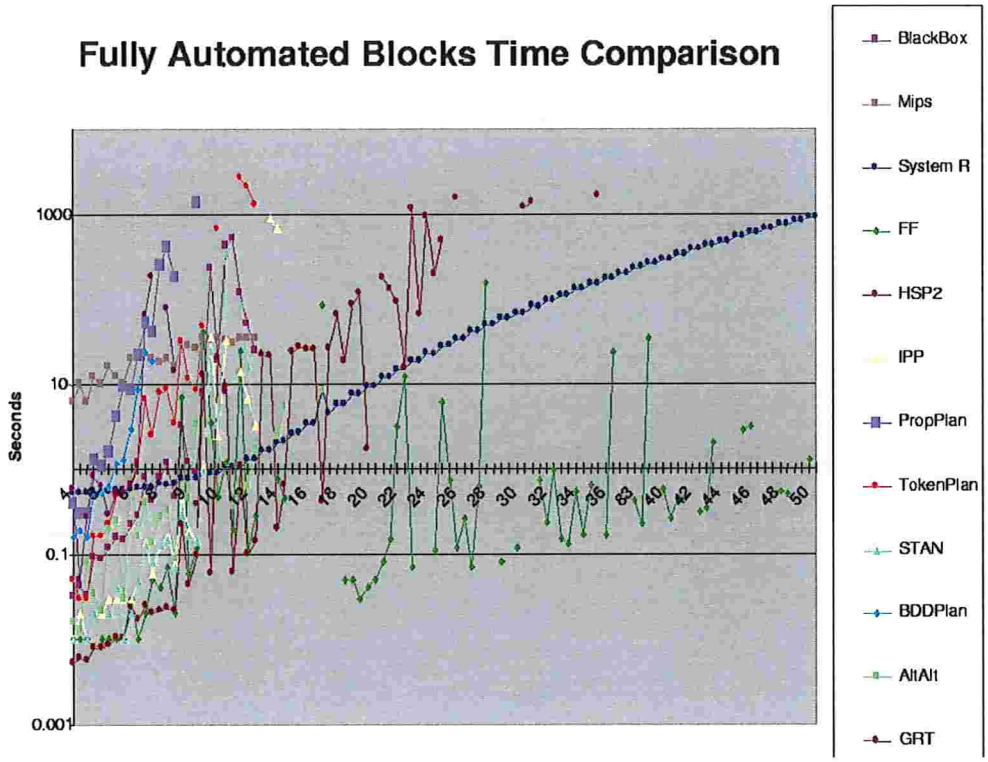

Figura 5.3: Desempenho dos planejadores na competição IPC-2000 no domínio do Mundo dos Blocos

5.4 O planejador *Metric-FF*

O planejador *Metric-FF* [21] estende a idéia da heurística do planejador *FF* para tarefas de planejamento que envolvem variáveis numéricas.

5.4.1 Variáveis numéricas de estado

Tarefas de planejamento que envolvem variáveis numéricas necessitam de uma nova linguagem para a descrição de domínios e problemas. Uma maneira simples de se obter

---

Algoritmo InstanciadorDeOperadores (Δ, Σ)
Entradas:
uma teoria do domínio Δ
um estado inicial Σ

Saída:
um conjunto O de ações proposicionais

---

1. Inicializa O um conjunto vazio de ações proposicionais;
2. Inicializa P um conjunto vazio de proposições;
3. Para cada predicado $p \in \Sigma$, adicionar uma proposição correspondente a p em P;
4. Para cada ação schema $\delta \in \Delta$, fazer
5. Para cada combinação de objetos para os parâmetros de δ, fazer
6. Se todas as proposições correspondentes às pré-condições de δ pertencem à P, fazer
7. Adicionar uma ação proposicional correspondente a δ em O;
8. Para cada predicado $p \in Ep(\delta)$, adicionar uma proposição correspondente a p em P;
9. Para cada predicado $p \in En(\delta)$, adicionar uma proposição correspondente a p em P;
10. Fim se
11. Fim para
12. Fim para
13. Retorna ao passo 4 se alguma nova ação foi adicionada a O;

Figura 5.4: Pseudo código do algoritmo para instanciar esquemas de operadores

uma nova linguagem com as características desejadas é estendendo a linguagem STRIPS. Para isso, é necessário aprimorar a linguagem STRIPS, que é puramente proposicional, com pré-condições numéricas, objetivos sobre variáveis numéricas e efeitos de ações sobre variáveis numéricas [16]. Conforme já descrito na Subseção 2.10, o segundo nível da linguagem PDDL possibilita descrever domínios e problemas segundo essa extensão. O planejador *Metric-FF* [21] utiliza operadores descritos pela linguagem PDDL2.1, extensão de PDDL com variáveis numéricas. Além disso, o planejador *Metric-FF* utiliza uma linguagem para representação interna de estados, ações e planos com variáveis

numéricas descritas a seguir.

Um estado do mundo, que segundo a Definição 1 era descrito por uma conjunção de proposições, passa a ser descrito por:

$$\mathcal{S} = \{(\alpha, \beta) | \alpha : V_P \rightarrow \{\bot, \top\}, \beta : V_N \rightarrow \mathbb{Q}\}$$

onde os conjuntos finitos e disjuntos V_P e V_N são chamados de conjunto de variáveis proposicionais de estado e conjunto de variáveis numéricas de estado, respectivamente. Dessa forma, um estado é descrito por um vetor α de atribuições de valores verdade às variáveis proposicionais de estado V_P, e um vetor β de atribuições de números racionais às variáveis numéricas de estado V_N.

Uma **condição proposicional** sobre uma variável proposicional $v \in V_P$ é descrita como $v = \top$. Uma condição proposicional $v = \top$ é satisfeita por um estado $s = (\alpha, \beta)$ se e somente se, $\alpha(v) = \top$. Um **efeito proposicional** sobre uma variável proposicional $v \in V_P$ com um valor verdade $t \in \{\bot, \top\}$ é descrito como $v \leftarrow t$. Dado o efeito proposicional $v \leftarrow t$, diz-se que v é a **variável afetada**.

Para um conjunto $\mathcal{C}$ de funções racionais (em uma ou mais variáveis), uma **condição numérica** sobre $\mathcal{C}$ é dada por uma função $f \in \mathcal{C}$ de aridade n, sendo n o número de variáveis numéricas do conjunto V_N, e um **operador relacional** $op{-}rel \in \{<, \leq, =, \neq, \geq, >\}$. A condição numérica é escrita como $f(v_1, \cdots, v_n)$ $op{-}rel$ 0, e é satisfeita por um estado $s = (\alpha, \beta)$ se e somente se, $f(\beta(v_1), \cdots, \beta(v_n))$ $op{-}rel$ 0. Para um conjunto $\mathcal{E}$ de funções racionais (em uma ou mais variáveis), um **efeito numérico** sobre $\mathcal{E}$ é dado por uma função $f \in \mathcal{E}$ de aridade n, sendo n o número de variáveis numéricas do conjunto V_N. O efeito numérico é escrito como $v_i \leftarrow f(v_1, \cdots, v_n)$, e diz-se que $v_i \in V_N$ é a **variável numérica afetada**.

Um **operador** sobre os conjuntos de funções racionais $\mathcal{C}$ e $\mathcal{E}$ é o par (Pre, Eff), onde Pre é um conjunto finito de condições proposicionais e condições numéricas sobre $\mathcal{C}$, e Eff é um conjunto finito de efeitos proposicionais e efeitos numéricos sobre $\mathcal{E}$, tais que efeitos diferentes tem variáveis afetadas diferentes.

Definição 8 *Um formalismo numérico de planejamento é uma tripla $\mathcal{F} = (\mathcal{G}, \mathcal{P}, \mathcal{E})$, onde $\mathcal{G}, \mathcal{P}, \mathcal{E} \subseteq \bigcup_{n \geq 1}(\mathbb{Q}^n \rightarrow \mathbb{Q})$ são conjuntos de funções racionais em uma ou mais variáveis.*

A Definição 8 serve para especificar os conjuntos de funções racionais utilizadas na descrição de uma tarefa de planejamento com variáveis numéricas. O conjunto de funções $\mathcal{G}$ é chamado de funções das condições numéricas do objetivo, o conjunto $\mathcal{P}$ é chamado de funções das condições numéricas das pré-condições dos operadores, e o conjunto $\mathcal{E}$ é chamado de funções dos efeitos numéricas dos operadores.

Definição 9 *O problema de planejamento com variáveis numéricas sobre um formalismo numérico $\mathcal{F} = (\mathcal{G}, \mathcal{P}, \mathcal{E})$ é formalmente descrito pela 5-upla $< V_P, V_N, \Delta, \Sigma, \Gamma >$ [16], onde:*

1. *V_P é um conjunto de variáveis proposicionais;*
2. *V_N é um conjunto de variáveis numéricas;*
3. *Δ é um conjunto finito de operadores sobre $\mathcal{P}$ e $\mathcal{E}$;*
4. *Σ é uma descrição do estado inicial do mundo;*
5. *Γ é uma descrição do objetivo do agente sobre $\mathcal{G}$.*

A Definição 9 estende a descrição apresentada na Seção 2.2 com a inclusão do conjunto de variáveis numéricas, condições numéricas e efeitos numéricos. Para o formalismo numérico $\mathcal{F} = (\emptyset, \emptyset, \emptyset)$, o problema de planejamento segundo a Definição 9 reduz-se ao planejamento STRIPS, ou puramente proposicional.

Helmert demonstrou que apenas algumas combinações de formalismos numéricos de planejamento permitem descrever problemas de planejamento que são decidíveis [16]. As versões numéricas dos domínios do Satélite e do Robô de Exploração, descritos na Seção 4.5 utilizam o formalismo numérico $\mathcal{F} = (\emptyset, \{x \mapsto x - c \mid c \in \mathbb{Q}\}, \{x \mapsto x + c \mid c \in \mathbb{Q}\})$. Nesse formalismo numérico, não existem objetivos expressos por meio de condições numéricas pois o conjunto $\mathcal{G}$ é vazio, as condições numéricas de aplicação dos operadores assumem a forma $v_i\ op{-}rel\ c$, e os efeitos numéricos dos operadores assumem a forma $v_i \leftarrow v_i + c$. Segundo Helmert, esse é um formalismo numérico decidível. Esse trabalho irá restringir-se a esse formalismo numérico, chamado de formalismo numérico restrito.

Definição 10 *Seja $T = (V_P, V_N, \Delta, \Sigma, \Gamma)$ uma tarefa de planejamento sobre um formalismo de planejamento $(\mathcal{G}, \mathcal{P}, \mathcal{E})$, e seja $\mathcal{S}$ o conjunto de estados de T. O grafo de transição de estados de T é definido como o grafo orientado com $\mathcal{S}$ como o conjunto de vértices, e o conjunto de arestas $((\alpha, \beta), (\alpha', \beta'))$ se e somente se, existe um operador $\delta \in \Delta$ tal que:*

(α, β) satisfaz todas as condições de $Pre(\delta)$.

Para todos os efeitos proposicionais $v \leftarrow t \in Eff(\delta)$,
$\alpha'(v) = t$.

Para todas as variáveis $v \in V_P$ não afetadas por qualquer efeito de $Eff(\delta)$,
$\alpha'(v) = \alpha(v)$.

Para todos os efeitos numéricos $v_i \leftarrow f(v_1, \cdots, v_n) \in Eff(\delta)$,
$\beta'(v_i) = f(\beta(v_1), \cdots, \beta(v_n))$.

Para todas as variáveis $v \in V_N$ não afetadas por qualquer efeito de $Eff(\delta)$,
$\beta'(v) = \beta(v)$.

A Definição 10 especifica a maneira como um operador δ aplicado em um estado (α, β) leva a um estado (α', β'). Nota-se que, essa definição formaliza a suposição do mundo fechado tanto para as variáveis proposicionais de estado, quanto para as variáveis numéricas de estado. Isso é feito pela persistência dos valores proposicionais e numéricos de todas as variáveis de estado não afetadas pelos efeitos da aplicação do operador.

Com essa linguagem, o planejador *Metric-FF* é capaz de analisar a aplicabilidade de ações com restrições sobre variáveis numéricas, bem como atualizar um estado do mundo levando em consideração os efeitos sobre as variáveis numéricas da ação.

5.4.2 A heurística do *Metric-FF*

A fim de resolver problemas de planejamento com variáveis numéricas de estado, o algoritmo de busca e a função heurística do planejador *Metric-FF* utilizam o formalismo numérico restrito, descrito na seção anterior. A função heurística do planejador *FF*

é monotônica no número de proposições, pois como ignora as listas de remoção, o número de proposições é uma função crescente. A função heurística para o planejador *Metric-FF* estende o conceito de monotonicidade para os efeitos numéricos, ou seja, as variáveis numéricas podem apenas ser incrementadas. Isso é obtido se os efeitos que decrementam as variáveis numéricas forem ignorados. Assim, a tarefa relaxada para o cálculo da heurística do planejador *Metric-FF* ignora os efeitos das listas de remoção e os efeitos que decrementam o valor das variáveis numéricas.

A idéia do planejador *Metric-FF* é resolver a tarefa relaxada para cada estado de busca s, e utilizar o comprimento da solução relaxada como estimativa heurística para um processo de busca local. A extensão da heurística do planejador *FF* é que, em adição aos níveis de proposições e ações, são propagados valores max_t^i para cada variável i através dos níveis t do grafo, que denotam o valor máximo que a variável v_i pode ter após aplicar t níveis de ações relaxadas. Os valores de max_0 são simplesmente os valores das variáveis no estado s.

Um estado objetivo é alcançado em um nível t do grafo de planejamento P quando, os objetivos proposicionais estão contidos em P_t e quando todas as restrições numéricas são satisfeitas de acordo com os valores de max_t^i. O grafo pode atingir um ponto fixo quando, em relação ao nível anterior, o último nível não tem proposições adicionais e os valores max de todas as variáveis numéricas permanece inalterado. Se um ponto fixo é atingido antes que o objetivo do problema seja alcançado, não há um plano que solucione o problema de planejamento relaxado a partir do estado s. O processo de construção desse novo grafo de planejamento relaxado tem complexidade polinomial, pois é muito similar ao empregado pelo planejador *FF*. Na Figura 5.5 é descrito esse algoritmo.

O algoritmo de construção do grafo para a tarefa relaxada (Figura 5.5) permite objetivos descritos como condições numéricas. Isso se deve ao fato de que, os resultados sobre a decidibilidade dos formalismos numéricos ter sido feito posteriormente à proposta do planejador *Metric-FF*. Na descrição desse algoritmo, foi utilizada a notação $P(x)$ para indicar as condições proposicionais do objetivo ou pré-condição x, bem com para indicar os efeitos proposicionais do conjunto de efeitos de um operador. Da mesma forma, $N(x)$ indicar as condições e efeitos numéricos. O passo 7, que verifica se o grafo atingiu um ponto fixo, utiliza $maxneed^i$ para indicar o maior valor de constante

---

Algoritmo constroi-grafo-numerico (Δ, Σ, Γ)
Entradas:
Uma descrição de um domínio Δ
Uma descrição do estado inicial s
Uma descrição do estado objetivo Γ
Saída:
falha ou um grafo de planejamento para o problema relaxado

---

1. $P_0 \longleftarrow \alpha_s, \forall v^i | max_0^i \longleftarrow \beta_s(v^i)$
2. $t \longleftarrow 0$
3. Enquanto $P(\Gamma) \not\subseteq P_t$ ou $(v^i, \geq [>], c) \in N(\Gamma), max_t^i \not\geq [\not>]c$, fazer
4. $\quad A_t \longleftarrow \{\delta \in \Delta | P(Pre(\delta)) \subseteq P_t, \forall (v^i, \geq [>], c) \in N(Pre(\delta)) | max_t^i \geq [>]c\}$
5. $\quad P_{t+1} \longleftarrow P_t \cup \bigcup_{\delta \in A_t} P(Eff(\delta)^+)$
6. $\quad \forall v^i max_{t+1}^i \longleftarrow max_t^i + \sum_{\delta \in A_t, (v^i, +=, c) \in N(Eff(\delta))} c$
7. $\quad$ Se $P_{t+1} = P_t$ e $\forall v^i | max_{t+1}^i = max_t^i$ ou $max_{t+1}^i > maxneed^i$, encerrar com falha
8. $\quad t \longleftarrow t + 1$
9. Fim fazer
10. $UltimoNivel \longleftarrow t$
11. Encerrar com o grafo de planejamento P

Figura 5.5: Pseudo código do algoritmo de construção do grafo de planejamento relaxado com variáveis numéricas

c existente em todas as condições numéricas do tipo $v_i \geq [>]c$.

Assim como o processo de construção do grafo, a busca por uma solução relaxada para a tarefa relaxada com variáveis numéricas é uma extensão do algoritmo utilizado pelo planejador *FF*. Na Figura 5.6 é descrito esse algoritmo.

Na descrição do algoritmo de busca por um plano para a tarefa relaxada com variáveis numéricas (Figura 5.6), foi utilizada a notação $nivel(x)$ para indicar o nível do grafo de planejamento de uma proposição ou ação x. Os passo 2 a 5 desse algoritmo selecionam os objetivos do problema no primeiro nível em que eles aparecem no grafo

de planejamento. Tais objetivos são armazenados em um conjunto de objetivos proposicionais e um conjunto de objetivos numéricos por nível do grafo, $P(\Gamma_t)$ e $N(\Gamma_t)$ respectivamente. De acordo com o passo 6, as escolhas das ações que resolvem os conjuntos de objetivos são feitas a partir do último nível do grafo até o primeiro nível. Os passos 7 a 13 selecionam ações para os objetivos proposicionais, adicionando as pré-condições das ações escolhidas como objetivos do nível imediatamente anterior. De forma semelhante, os passos 14 a 20 selecionam ações para os objetivos numéricos. O passo 19 serve para garantir o valor mínimo que uma variável numérica deve ter em um nível anterior ao nível de escolha de ações, para garantir a satisfação das condições numéricas das ações escolhidas.

Hoffmann [21] apresentou como o planejador *Metric-FF* estende a idéia da heurística para a tarefa relaxada com variáveis numéricas para as classes de formalismos numéricos mais expressivos. Tal extensão permite ao planejador *Metric-FF* utilizar toda a expressividade numérica possível de ser descrita com operadores da linguagem `PDDL2.1`. Tal extensão não é necessária para os problemas estudados nesse trabalho.

Hoffmann implementou duas versões do planejador *Metric-FF*, uma chamada de versão *Speed* e outra chamada de versão *Quality*. A versão *Speed* (velocidade), utiliza o algoritmo de busca *Hill-Climbing* reforçado guiado pela função heurística apresentada nesta sub-seção. Tal versão é destinada a encontrar planos no **menor tempo** possível. A versão *Quality* (qualidade), utiliza o algoritmo de busca WA^*, e é destinada a encontrar o **melhor** plano possível.

O planejador *Metric-FF* é um sistema de arquitetura homogênea, pois durante a avaliação da função heurística, a satisfação das condições numéricas dos problemas é feita simultaneamente com as condições proposicionais durante o processo de busca por um plano para a tarefa relaxada (Figura 5.6). Além do mais, qualquer que seja o algoritmo de busca local utilizado, *Hill-Climbing* reforçado ou WA^*, a escolha das ações para atingir o objetivo leva em consideração apenas o valor da função heurística. Ou seja, não há a necessidade de satisfazer condições numéricas depois de feita uma escolha de ações baseada no valor da função heurística.

5.5 O planejador *GMetric-FF*

Para esse trabalho, foi feita uma implementação do planejador *Metric-FF* que contou com algumas modificações propostas pelo autor. Uma dessa melhorias, como é descrito mais abaixo, é a utilização de uma heurística gulosa para a tratar o critério de otimização dos problemas. Em função dessa heurística gulosa adicional, o planejador implementado foi batizado de *GMetric-FF*.

O planejador *GMetric-FF* utiliza apenas o algoritmo de busca *Hill-Climbing* reforçado guiado pela função heurística. Ao ser executado, deve ser escolhido um dos dois modos de operação: rapidez ou qualidade. Quando executado no modo rapidez, o planejador *GMetric-FF* encerra a busca em largura do algoritmo *Hill-Climbing* reforçado assim que encontrar um estado de menor heurística. Tal forma de controle do planejador garante a rapidez na busca por um plano solução. No entanto, essa pode ser uma decisão precipitada, pois se continuar expandindo todos os nós da busca em largura para a profundidade em que a mesma está ocorrendo, pode ser encontrado um estado cujo valor da função heurística seja ainda menor. Essa é a forma de controle do planejador quando executado no modo qualidade. Nesse caso, podem ser encontrados planos solução menores, pois sendo a escolha local, não há garantia de que globalmente o plano será ótimo.

A outra modificações implementada pelo autor no planejador *GMetric-FF* é a utilização de uma heurística gulosa para tratar o critério de otimização dos problemas. Essa heurística só é utilizada quando o planejador é executado no modo qualidade. A heurística gulosa é utilizada quando a busca em largura encontra mais de um estado de menor heurística, cujo valor desta é igual para todos esses estados. Nessa situação, a escolha do caminho pela árvore da busca em largura é feita levando em consideração o estado que, localmente, está melhorando o critério de otimização. Por exemplo, se o critério de otimização é minimizar o combustível utilizado, e três estados da busca em largura tem o mesmo valor de heurística, menor que a heurística do estado inicial, o estado escolhido é aquele que tem o menor valor para a variável numérica de combustível utilizado. Da mesma maneira que o modo qualidade não garante que a solução encontrada é ótima, essa escolha gulosa não garante que o plano encontrado é ótimo em relação ao critério de otimização. Esse é o comportamento normal de qualquer

algoritmo guloso.

Como será mostrado no Capítulo 7, as modificações propostas pelo autor aqui, e implementas no planejador *GMetric-FF* como parte desse trabalho, fizeram com que este tivesse um desempenho comparável ao dos melhores planejadores participantes da competição *IPC-2002*.

5.6 Considerações finais

Nesse capítulo foram apresentados os algoritmos dos planejadores *FF* e *Metric-FF*, sendo esse último capaz de resolver problemas que envolvem recursos e variáveis numéricas. Também foram apresentados alguns detalhes adicionados ao planejador *GMetric-FF*, uma extensão do planejador *Metric-FF* implementado como parte desse trabalho.

No próximo capítulo é apresentada a teoria de Hoffmann, desenvolvida para análises teóricas de domínios de planejamento descritos na linguagem STRIPS. Também é apresentada uma extensão dessa teoria para analisar domínios que envolvem variáveis numéricas. A teoria de Hoffmann é então utilizada para análise dos domínios do Mundo dos Blocos, do Satélite e do Robô de Exploração, e a extensão da teoria de Hoffmann é utilizada para análise dos domínios do Satélite e do Robô de Exploração nas suas versões numéricas.

Algoritmo busca-plano-numerico-relaxado (P, Γ)
Entradas:
Um grafo de planejamento P
Uma descrição do estado objetivo Γ
Saída:
falha ou um plano para o problema relaxado

---

1. Criar um plano parcialmente ordenado Π vazio
2. Para $t = 1$ até $UltimoNivel$, fazer
3. $\quad P(\Gamma_t) \longleftarrow \{\gamma \in P(\Gamma) | nivel(\gamma) = t\}$
4. $\quad N(\Gamma_t) \longleftarrow \{(v^i, \geq [>], c) \in N(\Gamma) | nivel(v^i, \geq [>], c) = t\}$
5. Fim fazer
6. Para $t = UltimoNivel$ até 1, fazer
7. $\quad$ Para todo $\gamma \in P(\Gamma_t)$, fazer
8. $\quad\quad$ Selecionar a, $nivel(a) = t - 1, \gamma \in P(Eff(a)^+)$, adicionando a ao plano Π
9. $\quad\quad$ Para todo $p \in P(Pre(a)), (v^i, \geq [>], c) \in N(Pre(a))$, fazer
10. $\quad\quad\quad P(\Gamma_{nivel(p)}) \longleftarrow P(\Gamma_{nivel(p)}) \cup \{p\}$
11. $\quad\quad\quad N(\Gamma_{nivel(v^i, \geq [>], c)}) \longleftarrow N(\Gamma_{nivel(v^i, \geq [>], c)}) \cup \{(v^i, \geq [>], c)\}$
12. $\quad\quad$ Fim fazer
13. $\quad$ Fim fazer
14. $\quad$ Para todo $(v^i, \geq [>], c) \in N(\Gamma_t)$, fazer
15. $\quad\quad$ Enquanto $max^i_{t-1} \not\geq [\not>]c$, fazer
16. $\quad\quad\quad$ Selecionar a, $nivel(a) = t - 1, (v^i, + =, c') \in N(Eff(a))$, adicionando a ao plano Π
17. $\quad\quad\quad c \longleftarrow c - c'$
18. $\quad\quad$ Fim fazer
19. $\quad\quad N(\Gamma_{t-1}) \longleftarrow N(\Gamma_{t-1}) \cup \{(v^i, \geq [>], c)\}$
20. $\quad$ Fim fazer
21. Fim fazer
22. Encerrar com Π

Figura 5.6: Pseudo código do algoritmo de busca por um plano no grafo de planejamento relaxado com variáveis numéricas

Capítulo 6

Análise dos domínios de teste

Neste capítulo é mostrada uma análise teórica dos domínios do Satélite e do Robô de Exploração (Capítulo 4) propostos na competição de planejamento *IPC-2002*, que introduziu novos desafios com problemas envolvendo variáveis numéricas. Este trabalho é inspirado em uma análise anterior à competição [20] e que não considera variáveis numéricas. Portanto, este capítulo traz contribuições nessa direção.

Conforme citado anteriormente, a maioria dos sistemas de planejamento premiados nas competições internacionais de planejamento são baseados em busca heurística. As heurísticas de boa parte desses sistemas, assim como no caso do planejador *FF* (Capítulo 5) são computadas pelo tamanho, em número de passos, de uma solução para uma tarefa relaxada. No entanto, a despeito de ter sido provado que a computação de uma solução relaxada ótima pertence à classe NP-difícil [5], os sistemas de planejamento de busca heurística apresentaram um bom desempenho na resolução dos problemas propostos nas competições de planejamento. Isso levantou a suspeita de que o sucesso desses sistemas se deve ao fato dos domínios e problemas, utilizados para testes e para avaliação de desempenho nas competições, apresentarem uma topologia simples do espaço de busca. Hoffmann [22] formulou um conjunto de definições e lemas, que chamaremos aqui de teoria de Hoffmann sobre a topologia de espaços de busca local para planejamento, que permitem classificar domínios e problemas. Os resultados da aplicação da teoria de Hoffmann podem ser usados para explicar o comportamento de métodos de planejamento baseados em busca local em domínios específicos.

Na competição de planejamento *IPC-2002*, novos desafios foram introduzidos com

a adição de domínios de teste envolvendo tanto ações com variáveis numéricas como ações durativas [35]. Uma vez que a teoria de Hoffmann, utilizada nesse trabalho, é anterior a essa competição, esta considera em suas definições e lemas apenas ações do tipo STRIPS. Neste capítulo será mostrado:

- a teoria de Hoffmann sobre a topologia de espaços de estados para algoritmos de busca local em planejamento e sua análise do domínio do mundo dos blocos (Seções 6.1 e 6.4.1);
- como a teoria de Hoffmann pode ser utilizada para analisar as versões STRIPS dos domínios do Satélite e dos Robô de Exploração (Seções 6.1, 6.4.2, e 6.4.3) usando a heurística do *FF*, análise essa não realizada no trabalho de Hoffmann;
- a definição de ação *Numérica de Efeito Irrelevante*, que estende a definição de ação de *Efeito de Remoção Irrelevante* da teoria de Hoffmann (Seção 6.3);
- análise das versões numéricas dos domínios do Satélite e dos Robô de Exploração (Seção 6.4) com a heurística do *Metric-FF*, aplicando a extensão da teoria de Hoffmann.

Este capítulo fornece resultados importantes para explicar o comportamento dos métodos de planejamento baseados em busca local implementados nesse trabalho, o *FF* e *Metric-FF* (Capítulo 5), como será mostrado no próximo capítulo.

6.1 Topologia de espaços de estados em planejamento

Estudos desenvolvidos pela comunidade SAT, como em [11], apontam que o comportamento da busca local depende essencialmente da topologia do espaço de busca, como é de se esperar. A teoria de Hoffmann é baseada nesses estudos, porém, trata especificamente de espaços de estados para problemas de planejamento em IA, levando essencialmente em conta a estrutura das ações ou operadores de um domínio. Os fenômenos topológicos que são relevantes para busca local em planejamento envolvem: (i) a existência de *becos sem saída* em espaço de estados; (ii) a caracterização de espaços de busca como *bidirecional*, *inofensivo*, *reconhecido* e *não-reconhecido*; e (iii) a carac-

terização de regiões do espaço de estados em platôs como *becos sem saída*, *mínimos locais*, *planícies*, *selas*, *contornos* e *mínimos globais*.

6.1.1 Becos sem Saída

Na Seção 2.3 foi apresentada uma definição formal de um modelo de estados para problemas de planejamento em IA, em que o problema de planejamento é descrito pela 5-upla $< \mathcal{S}, s_0, S_G, \mathcal{A}, f >$. Para o propósito deste capítulo, os problemas de planejamento representados por um modelo de estados serão descritos pela 4-upla $< \mathcal{S}, s_0, S_G, \mathcal{E} >$ onde, $\mathcal{S}$ é o conjunto não vazio de estados de um espaço, $s_0 \in \mathcal{S}$ é o estado inicial, $S_G \subseteq \mathcal{S}$ é um subconjunto não vazio de estados objetivo, e $\mathcal{E} \subseteq \mathcal{S} \times \mathcal{S}$ é o conjunto das transições de estados nesse espaço [1].

Definição 11 *Dada uma tarefa de planejamento $< \mathcal{S}, s_0, S_G, \mathcal{E} >$, a distância de um estado $s \in \mathcal{S}$ ao objetivo, denotada por $gd(s)$, é dada por:*

$$gd(s) = min\{dist(s, s') \mid s' \in S_G\} \tag{6.1}$$

A distância $dist(s, s')$ entre os estados s e s', é dada pelo comprimento do caminho mais curto no grafo orientado, que liga o estado s ao estado s'. Caso não exista tal caminho, convenciona-se que $dist(s, s') = \infty$.

Em certos domínios, algumas tarefas de planejamento possuem estados a partir dos quais não existe um caminho que conduz a um estado objetivo. Tais estados são chamados de *becos sem saída*.

Definição 12 *Para qualquer estado $s \in \mathcal{S}$, se $gd(s) = \infty$ então, s é um* ***beco sem saída****.*

[1] O conjunto $\mathcal{E}$ é uma maneira alternativa de descrever a função de transição de estados $f(a, s)$. Dados dois estados $s, s' \in \mathcal{S}$, se $(s, s') \in \mathcal{E}$ então, existe uma ação $a \in \mathcal{A}(s)$ aplicável no estado s que tem s' como estado resultante.

Assim, o espaço de busca a ser considerado por um sistema de planejamento pode ser reduzido ao sub-conjunto de estados $\{s \in \mathcal{S} \mid gd(s) < \infty\}$ sem afetar a completude da busca.

O objetivo de uma função heurística, denotada por $h(s)$, é computar uma boa aproximação para o comprimento $gd(s)$ da forma mais eficiente possível.

Definição 13 *Dada uma tarefa de planejamento* $< \mathcal{S}, s_0, S_G, \mathcal{E} >$, *uma heurística é uma função* $h : \mathcal{S} \rightarrow \mathbb{N}_0 \cup \{\infty\}$ *tal que,* $h(s) = 0 \Longleftrightarrow gd(s) = 0$

De acordo com a Definição 13, é imprescindível que uma função heurística seja capaz de reconhecer todos os estados objetivos. Algumas propriedades topológicas dos espaços de busca podem ser observadas quando utiliza-se o comprimento de uma solução relaxada ótima como função heurística, como é o caso da heurística empregada pelo sistema *FF* [18]. Esse tipo de função heurística é denotado por $h^+(s)$ (o sinal positivo da notação indica que, para a tarefa relaxada a heurística só avalia a lista de adição das ações).

Definição 14 *Uma função heurística qualquer é dita* ***preservadora de completude*** *sempre que valer a seguinte relação:* $\forall s \in \mathcal{S} \quad h^+(s) = \infty \Longrightarrow gd(s) = \infty$

Segundo a Definição 14, uma função preservadora de completude é aquela que, apesar de não garantir reconhecer todos os becos sem saída, sempre que ao avaliar a heurística de um estado como infinita, garantidamente o estado é um beco sem saída. Assim, em uma busca que emprega uma função heurística preservadora de completude, o espaço de busca pode ser reduzido ao sub-conjunto de estados $\{s \in \mathcal{S} \mid h^+(s) < \infty\}$, sem perda da completude do processo de busca. A heurística do FF é preservadora de completude, pois se a tarefa relaxada não pode ser resolvida então a tarefa de planejamento original também não tem solução [20]. A redução do espaço de busca pelo reconhecimento de becos sem saída é uma das características que confere um bom desempenho aos planejadores baseados em busca heurística.

Definição 15 *Dada uma tarefa de planejamento* $< \mathcal{S}, s_0, S_G, \mathcal{E} >$ *com uma heurística*

preservadora de completude, os espaços de estados podem ser classificados segundo um dos quatro tipos abaixo:

1. ***Bidirecional:***

$$\forall (s, s') \in \mathcal{E} \Longleftrightarrow (s', s) \in \mathcal{E} \tag{6.2}$$

2. ***Inofensivo:***

$$\exists (s, s') \in \mathcal{E} \mid (s', s) \notin E \quad e \quad \forall s \in \mathcal{S} \; gd(s) < \infty \tag{6.3}$$

3. ***Reconhecido:***

$$\exists s \in \mathcal{S} \mid gd(s) = \infty \quad e \quad \forall s \in \mathcal{S} \quad gd(s) = \infty \Longrightarrow h^{+}(s) = \infty \tag{6.4}$$

4. ***Não-Reconhecido:***

$$\exists s \in \mathcal{S} \mid gd(s) = \infty \quad e \quad h^{+}(s) < \infty \tag{6.5}$$

Nos espaços de estados nos quais o efeito da aplicação de qualquer ação pode ser revertido pela aplicação de uma ação inversa correspondente, não há becos sem saída, e o espaq é do tipo bidirecional. Quando em um espaço de estado existe pelo menos uma ação que não tem uma inversa correspondente e, assim mesmo o espaço de estados não possui becos sem saída, o espaço é do tipo inofensivo. Espaços que possuem becos sem saída, sendo todos reconhecidos por meio da avaliação da função heurística $h^{+}(s)$ são do tipo reconhecido. Caso a função heurística $h^{+}(s)$ não seja capaz de reconhecer algum beco sem saída de um espaço de estados, ele é do tipo não-Reconhecido.

Planejadores que fazem busca global não são afetados por espaços de estados que contém becos sem saída pois, estes algoritmos podem retroceder no processo de busca sempre que atingirem um beco sem saída. Os planejadores de ordem parcial também não encontram dificuldades nesse tipo de espaços de estados pois, os algoritmos desse tipo de planejadores raciocinam sobre planos e não sobre estados. Em planejadores que fazem busca local, o único caso crítico para a busca é aquele em que o algoritmo visita um beco sem saída não reconhecido. Ao atingir um estado desse tipo, o mesmo não é excluído do processo de busca, como é feito quando a função heurística identifica

que o estado é um beco sem saída. Como os algoritmos de busca local não retrocedem no processo de busca, quando seguem por um caminho que passa por um beco sem saída esses planejadores não são capazes de encontrar uma solução para a tarefa de planejamento. Ou seja, se não for possível reconhecer todos os becos sem saída, um planejador baseado em busca local não é completo.

6.1.2 Platôs

Um platô de nível l é um sub-conjunto de estados para os quais o valor da função heurística é constante e igual a l.

Definição 16 *Seja $\mathcal{P} \in \mathcal{S}$ um sub-conjunto maximal de estados do espaço $\mathcal{S}$, em que o sub-grafo induzido em $(\mathcal{S}, \mathcal{E})$ é fortemente conectado* [2]*, onde $\forall s \in \mathcal{P} \quad h^+(s) = l$, $\mathcal{P}$ é chamado de **platô** de nível l.*

Platôs são, de acordo com a avaliação da função heurística, regiões equivalentes do espaço de estados, sendo que qualquer estado de um espaço pertence a exatamente um único platô.

Definição 17 *Um estado $s \in \mathcal{P}$ é uma **saída do platô** $\mathcal{P}$ se, existir um estado $s' \notin \mathcal{P}$, tal que $(s, s') \in \mathcal{E}$ e $h^+(s') \leq h^+(s)$. O estado s é uma **saída aprimoradora** se, para pelo menos um estado s', $h^+(s') < h^+(s)$.*

Qualquer espaço de busca pode ser particionado em platôs de diferentes níveis [11]. De acordo com a quantidade de saídas de cada platô, os mesmos podem ser agrupados em seis diferentes classes:

1. **beco sem saída**: é um platô de nível $h^+ = \infty$;
2. **mínimo local**: é um platô de nível $0 < h^+ < \infty$ que não tem saídas;

[2] Dado um sub-conjunto S de vértices de um grafo G, o sub-grafo induzido por S é o grafo com o conjunto de vértices S e o conjunto de arestas consistindo das mesmas arestas de G existentes entre dois vértices quaisquer de S. Um grafo direcionado D é dito fortemente conectado se para cada vértice $v \in V(D)$ existir um caminho direcionado de v para qualquer outro vértice de D.

3. **planície**: é um platô de nível $0 < h^+ < \infty$ que possui pelo menos uma saída, mas nenhuma saída aprimoradora;

4. **sela**: é um platô de nível $0 < h^+ < \infty$ que possui pelo menos uma saída aprimoradora, e pelo menos um estado que não é uma saída aprimoradora;

5. **contorno**: é um platô de nível $0 < h^+ < \infty$ que consiste exclusivamente de saídas aprimoradora;

6. **mínimo global**: é um platô P de nível 0.

6.2 A Teoria de Hoffmann

Hoffmann [22] formulou sua teoria como um conjunto de definições e lemas que permitem classificar domínios quanto a topologia do seu espaço de busca, essencialmente por meio da análise dos operadores do domínio de planejamento. Na Subseção 6.2.1 são apresentados os critérios necessários para caracterizar tarefas de planejamento quanto à existência ou não de **becos sem saída**. Na Subseção 6.2.2 são apresentados os critérios necessários para caracterizar tarefas de planejamento quanto à existência ou não de **mínimos locais**.

6.2.1 Verificação de becos sem saída

A fim de identificar a existência ou não de becos sem saída no espaço de estados de uma tarefa de planejamento, é necessário identificar certas propriedades dos operadores do domínio dessa tarefa. Essas propriedades dos operadores definem algumas características dos estados resultantes da aplicação dos mesmos. Tais propriedades são verificadas pura e simplesmente pela análise sintática dos operadores, segundo os critérios das definições a seguir.

Definição 18 *Duas proposições são inconsistentes se não existir um estado válido que as contenha simultaneamente. Dois conjuntos de proposições P e P', são inconsistentes se, para cada proposição de P existir uma proposição inconsistente a ela em P'.*

Segundo a Definição 18, duas proposições inconsistentes, ou dois conjuntos de proposições inconsistentes, nunca podem ser verdadeiros em qualquer estado válido. Essa definição é utilizada como uma condição necessária para as demais definições apresentadas.

Definição 19 *Seja uma tarefa de planejamento* $< \Delta, \Sigma, \Gamma >$. *Uma ação* $\delta \in \Delta$ *é* ***inversível*** *se:*

1. *existe uma ação* $\overline{\delta} \in \Delta$ *tal que:*
   (a) $pre(\overline{\delta}) \subseteq (pre(\delta) \cup add(\delta)) \setminus del(\delta)$
   (b) $add(\overline{\delta}) = del(\delta)$
   (c) $del(\overline{\delta}) = add(\delta)$
2. $add(\delta)$ *é inconsistente com* $pre(\delta)$
3. $del(\delta) \subseteq pre(\delta)$

Para uma ação δ inversível e aplicável em um estado s, tendo s' como estado resultante, a ação $\overline{\delta}$ *inversa* de δ é, por definição, aplicável em s', e tem s como estado resultante. Tal característica é garantida pela primeira condição da Definição 19. A segunda condição garante que os efeitos adicionados pela ação δ não eram verdadeiros no estado s em que essa foi aplicada. Por serem inconsistentes com as proposições da pré-condição, nenhuma das proposições adicionadas pela ação δ podem ser verdadeiras no estado s. Essa garantia é imprescindível para que a ação inversa $\overline{\delta}$ ao desfazer os efeitos da ação δ retorne exatamente ao estado s. A terceira condição garante que os efeitos removidos pela ação δ não eram falsos no estado s, pois só podem ser removidas proposições existentes na pré-condição da ação. Por satisfazer as três condições, a ação inversa $\overline{\delta}$ permite desfazer todos os efeitos da aplicação da ação δ, ou seja, o estado resultante de sua aplicação é exatamente o mesmo estado em que a ação δ foi aplicada.

Lema 1 *Seja uma tarefa de planejamento* $< \Delta, \Sigma, \Gamma >$. *Se todas as ações* $\delta \in \Delta$ *são inversíveis, então o espaço de estados para a tarefa não possui becos sem saída e é bidirecional (Definição 15 - 6.2).*

De acordo com o Lema 1 é possível determinar que um domínio está livre de becos sem saída, bastando demonstrar que todas as ações do mesmo são inversíveis, segundo os critérios da Definição 19. A prova desse lema foi apresentada por Hoffmann [19].

Definição 20 *Seja uma tarefa de planejamento* $< \Delta, \Sigma, \Gamma >$. *Uma ação* $\delta \in \Delta$ *é* ***minimamente inversível*** *se existe uma ação* $\overline{\delta} \in \Delta$ *tal que:*

1. $pre(\overline{\delta}) \subseteq (pre(\delta) \cup add(\delta)) \setminus del(\delta)$
2. $add(\overline{\delta}) \supseteq del(\delta)$
3. $del(\overline{\delta})$ *é inconsistente com* $pre(\delta)$

Para uma ação δ minimamente inversível e aplicável em um estado s, tendo s' como estado resultante, a ação $\overline{\delta}$ *inversa minimal* de δ é, por definição, aplicável em s', e tem s'' como estado resultante. Por definição, a ação δ é aplicável no novo estado s''. Tal característica é garantida pelas duas primeiras condições da definição. A terceira condição garante que os efeitos removidos pela ação $\overline{\delta}$ não eram verdadeiros no estado s. Isso significa dizer que a ação inversa minimal $\overline{\delta}$, apesar de não retornar ao estado em que a ação δ foi aplicada, permite desfazer todos os efeitos da aplicação da ação δ. A segunda condição da definição indica que a ação $\overline{\delta}$ pode adicionar novos predicados ao estado resultante, sem com isso comprometer a aplicabilidade da ação δ nesse novo estado resultante.

Definição 21 *Seja uma tarefa de planejamento* $< \Delta, \Sigma, \Gamma >$. *Uma ação* $\delta \in \Delta$ *tem* ***efeitos de adição estáticos*** *se:*

$$add(\delta) \cap \bigcup_{\delta' \in \Delta} del(\delta') = \emptyset$$

Uma ação $\delta \in \Delta$ *tem* ***efeitos de remoção irrelevantes*** *se:*

$$del(\delta) \cap (\Gamma \cup \bigcup_{\delta \neq \delta' \in \Delta} pre(\delta')) = \emptyset$$

Uma ação δ tem efeitos de adição estáticos se as proposições que adiciona ao estado resultante não puderem ser removidas por nenhuma outra ação do domínio. Ou seja, qualquer efeito estático não podem ser desfeito. Analogamente, uma ação δ tem efeitos de remoção irrelevantes se as proposições removidas não fizerem parte da pré-condição de nenhuma outra ação do domínio, nem tampouco do objetivo do problema. Isso significa que, qualquer efeito irrelevante não representa um conflito à aplicabilidade de outras ações, nem a remoção de uma proposição do objetivo.

Lema 2 *Seja uma tarefa de planejamento* $< \Delta, \Sigma, \Gamma >$. *Se valer para todas as ações* $\delta \in \Delta$ *uma das seguintes propriedades:*

1. δ *é minimamente inversível, ou*

2. δ *tem efeitos de adição estáticos e efeitos de remoção irrelevantes*

então o espaço de estados para a tarefa não possui becos sem saída e pode ser bidirecional (Definição 15 - 6.2) ou inofensivo (Definição 15 - 6.3).

Se uma tarefa de planejamento satisfaz as condições apresentadas no Lema 2 e não satisfaz as condições apresentadas no Lema 1, o espaço de estados para essa tarefa é inofensivo (Definição 15 - 6.3). Qualquer tarefa de planejamento que satisfaz as condições apresentadas no Lema 1, também satisfaz as condições apresentadas no Lema 2, e o espaço de estados para essa tarefa é bidirecional (Definição 15 - 6.2). A prova desse lema também foi apresentada por Hoffmann [19].

As duas propriedade apresentadas pelos Lemas 1 e 2, que permitem classificar os espaços de estados quanto a existência de becos sem saída, são próprias da tarefa de planejamento, ou seja, não dependem da função heurística $h^+(s)$. Por sua vez, a classificação de um espaço de estados com becos sem saída reconhecidos depende totalmente da função heurística.

A partir da descrição dos operadores de um domínio, a verificação da existência de becos sem saída nos espaços de estado das tarefas nesse domínio deve seguir o seguinte procedimento de prova:

1. Para cada ação do domínio, utilizar os critérios da Definição 19 para verificar aquelas que são inversíveis. Se essa propriedade for verificada para todas as ações, qualquer tarefa nesse domínio não apresenta becos sem saída, de acordo com o Lema 1.

2. Para cada ação do domínio que não é inversível, utilizar os critérios da Definição 20 para verificar aquelas que são minimamente inversíveis.

3. Para cada ação do domínio que não é inversível nem minimamente inversível, utilizar os critérios da Definição 21 para verificar aquelas que tem efeitos de adição estáticos ou efeitos de remoção irrelevantes. Se qualquer uma dessas três propriedades for verificada para todas as ações, qualquer tarefa nesse domínio não apresenta becos sem saída, de acordo com o Lema 2.

6.2.2 Verificação de mínimos locais

As definições e lemas apresentadas a seguir são critérios suficientes para identificar tarefas de planejamento que não contém mínimos locais, de acordo com a avaliação de uma função heurística. Para esse trabalho é considerada a função heurística do planejador *FF*.

Definição 22 *Seja uma tarefa de planejamento* $< \Delta, \Sigma, \Gamma >$. *Uma ação* $\delta \in \Delta$ *é* ***minimamente inversível relaxada*** *se, existir uma ação* $\overline{\delta} \in \Delta$ *tal que:*

1. $pre(\overline{\delta}) \subseteq (pre(\delta) \cup add(\delta)) \setminus del(\delta)$
2. $add(\overline{\delta}) \supseteq del(\delta)$

Quando é utilizada uma ação relaxada $\overline{\delta}$ para inverter os efeitos de uma ação δ, basta que essa ação relaxada $\overline{\delta}$ adicione novamente as proposições que foram removidos pela ação δ. Sendo $\overline{\delta}$ um ação relaxada, seus efeitos de remoção são ignorados e portanto a ação δ não necessita adicionar qualquer proposição para desfazê-los.

Definição 23 *Seja uma tarefa de planejamento* $< \Delta, \Sigma, \Gamma >$. *Uma ação* $\delta \in \Delta$ *é* ***respeitada pela relaxação*** *se, para qualquer estado* s *tal que* δ *inicia um plano ótimo*

para a tarefa $< \Delta, s, \Gamma >$, *existe um plano relaxado ótimo para essa tarefa que também é iniciado por* δ.

Um exemplo, para melhor compreender a definição acima, pode ser apresentado com as ações do domínio do Satélite. A única alternativa possível de obter uma imagem de um alvo, desde que o instrumento esteja devidamente calibrado, obrigatoriamente inicia-se com a ação que movimenta o Satélite de forma a apontá-lo para o alvo do objetivo. Ou seja, qualquer plano, relaxado ou não, precisa necessariamente iniciar com a ação que movimenta o Satélite. Essa afirmação foi provada para a utilização da heurística do planejador *FF*, cujo processo para encontrar um plano relaxado ótimo é bem conhecido.

Lema 3 *Seja uma tarefa de planejamento* $< \Delta, \Sigma, \Gamma >$ *que tenha uma solução, e que o espaço de estados não apresente becos sem saída não-reconhecidos. Se valer para todas as ações* $\delta \in \Delta$ *uma das seguintes propriedades:*

1. *é respeitada pela relaxação (de acordo com a avaliação da função heurística* h^+*) e é minimamente inversível relaxada, ou*
2. *tem efeitos de remoção irrelevantes*

então não existem mínimos locais no espaço de estados.

A prova desse lema foi apresentada por Hoffmann [19].

6.3 Extensão da teoria para domínios numéricos

A formalização teórica proposta por Hoffmann, e apresentada na seção 6.2 trata apenas de problemas descritos pela linguagem `STRIPS` e `ADL`. Uma extensão à teoria de Hoffmann é necessária para que possam ser analisados domínios que envolvem variáveis numéricas. Dessa maneira, é apresentada a seguir uma definição e um lema que permitem identificar propriedades de ações com efeitos numéricos de maneira similar ao que foi apresentado para ações do tipo `STRIPS`. A extensão proposta aqui destina-se

exclusivamente a problemas descritos pela linguagem numérica restrita apresentada na Subseção 5.4.1 do Capítulo 5, o que é suficiente para os domínios considerados neste trabalho.

Conforme a formalização apresentada na Subseção 5.4.1, um estado de uma tarefa de planejamento com variáveis numéricas é descrito por (α, β), sendo α um vetor de atribuições de valores verdade às variáveis proposicionais de estado V_P, e β um vetor de atribuições de números racionais às variáveis numéricas de estado V_N. Para uma ação qualquer δ do domínio, as restrições numéricas da pré-condição de δ serão representadas por $Pre(\delta)$, e os efeitos numéricos por $Eff(\delta)$.

Definição 24 *Seja uma tarefa de planejamento com variáveis numéricas* $T = (V_P, V_N, \Delta, \Sigma, \Gamma)$. *Uma ação* $\delta \in \Delta$ *tem* ***efeitos numéricos irrelevantes*** *se:*

$\forall v_i \in V_N | v_i$ *é afetado pelos efeitos* $Eff(\delta) \rightarrow \forall \delta' \in \Delta$, v_i *não é parte das restrições de* $Pre(\delta')$

Uma ação δ tem efeitos numéricos irrelevantes se todas as variáveis numéricas afetadas pelos efeitos de sua aplicação não fizerem parte das restrições numéricas da pré-condição de nenhuma ação do domínio. Ou seja, qualquer efeito numérico irrelevante não representa um conflito à aplicabilidade de qualquer ação.

Lema 4 *Dada uma tarefa de planejamento* $T = (V_P, V_N, \Delta, \Sigma, \Gamma)$. *Se valer para todas as ações* $\delta \in \Delta$ *uma das seguintes propriedades:*

1. δ *não tem efeitos numéricos e, é minimamente inversível, ou*
2. δ *não tem efeitos numéricos e, tem efeitos de adição estáticos e efeitos de remoção irrelevantes, ou*
3. δ *tem efeitos numéricos irrelevantes e, é minimamente inversível, ou*
4. δ *tem efeitos numéricos irrelevantes e, tem efeitos de adição estáticos e efeitos de remoção irrelevantes*

então o espaço de estados para a tarefa ***não possui becos sem saída.***

A propriedade apresentada pelo Lema 4 é muito semelhante àquela do Lema 2, e permite classificar os espaços de estados com variáveis numéricas quanto à existência de becos sem saída. Se forem eliminadas as variáveis numéricas, o Lema 4 reduz-se facilmente ao Lema 2. A seguir, a demonstração do Lema 4.

Demonstração do Lema 4:

Seja uma tarefa de planejamento $T = (V_P, V_N, \Delta, \Sigma, \Gamma)$. Para que essa tarefa não tenha becos sem saída, para qualquer estado $s = (\alpha, \beta) \in \mathcal{S}$, deve valer $gd(s) < \infty$. Isso pode ser provado se, para qualquer estado alcançável $s = Result(\Sigma, < \delta_1, \delta_2, \ldots, \delta_k >) \in \mathcal{S}$ existir um plano solução Π para a tarefa $(V_P, V_N, \Delta, s, \Gamma)$.

De acordo com o Lema 2, se para qualquer ação $\delta \in \Delta$ do domínio valer a condição 1 ou a condição 2 do Lema 4, o plano Π para a tarefa $(V_P, V_N, \Delta, s, \Gamma)$ existe, e essa tarefa não possui becos sem saída.

Se no entanto existirem ações com efeitos numéricos, de modo que para qualquer ação $\delta \in \Delta$ do domínio valer a condição 1, 2 ou 3 do Lema 4, nesse caso, seja o plano $\Pi =< \pi_1, \ldots, \pi_i, \ldots, \pi_k >$. Se para qualquer ação π_i do plano Π valer a condição 1 ou 2 do Lema 4, vale a conclusão anterior. No entanto, se para qualquer ação π_i do plano Π valer a condição 3 do Lema 4, exite uma ação $\overline{\pi_i}$, inversa minimal da ação π_i. A ação $\overline{\pi_i}$ permite desfazer os efeitos proposicionais da ação π_i, o que, de acordo com o Lema 2, faz com que a tarefa não tenha becos sem saída. Pela definição 24, os efeitos numéricos da ação π_i não afetam a aplicabilidade da ação $\overline{\pi_i}$, não comprometendo assim as propriedades dessa ação caso ela não tivesse efeitos numéricos. Dessa maneira, o plano Π para a tarefa $(V_P, V_N, \Delta, s, \Gamma)$ é válido, e essa tarefa não possui becos sem saída.

Finalmente, se existirem ações com efeitos numéricos, de modo que para qualquer ação $\delta \in \Delta$ do domínio valer a condição 1, 2, 3 ou 4 do Lema 4, nesse caso, seja o plano $\Pi =< \pi_1, \ldots, \pi_i, \ldots, \pi_k >$. Se para qualquer ação π_i do plano Π valer a condição 1, 2 ou 3 do Lema 4, vale a conclusão anterior. No entanto, se para qualquer ação π_i do plano Π valer a condição 4 do Lema 4, os efeitos proposicionais dessa ação não afetam a aplicabilidade de outras ações, o que, de acordo com o Lema 2, faz com que a tarefa não tenha becos sem saída. Pela definição 24, os efeitos numéricos da ação π_i também

não afetam a aplicabilidade de qualquer outra ação do domínio, não comprometendo assim as propriedades dessa ação caso ela não tivesse efeitos numéricos. Dessa maneira, o plano Π para a tarefa $(V_P, V_N, \Delta, s, \Gamma)$ é válido, e essa tarefa não possui becos sem saída ■

6.4 Análise dos domínios estudados

Hoffmann demonstrou cada um dos Lemas de sua teoria [19], e empregou-os na tarefa de classificar os domínios [22] utilizados nas avaliações de desempenho das competições *IPC-1998* e *IPC-2000*. Essa classificação permitiu a classificação desses domínios segundo uma taxonomia (Figura 6.1).

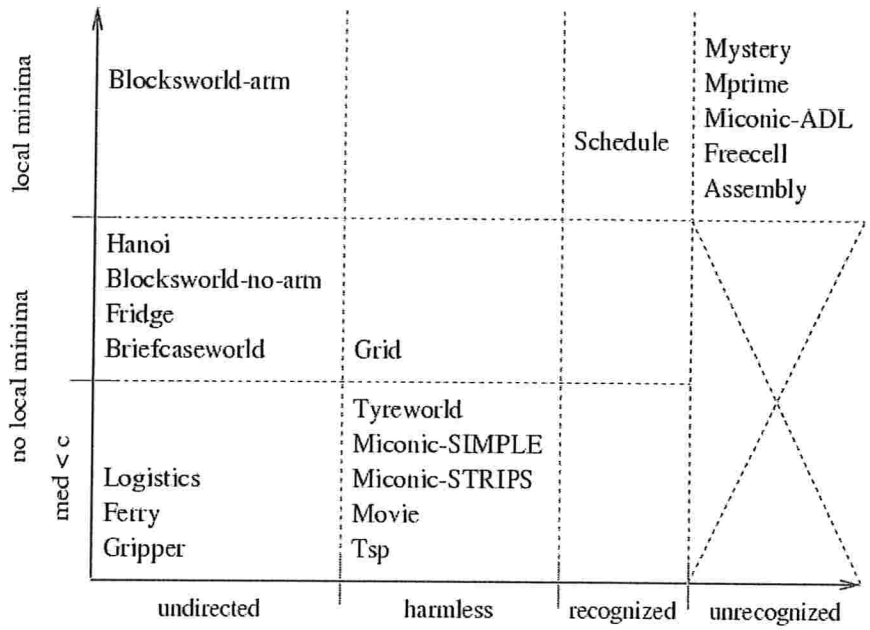

Figura 6.1: Taxonomia dos domínios da competição *IPC-2000*

Um dos resultados importantes da análise feita por Hoffmann [22] é que a maioria dos domínios de teste utilizados nas competições de planejamento e escalonamento, as tarefas de planejamento têm espaços de estados do tipo bidirecional ou inofensivo. Também foi verificado que a grande maioria dos domínios não apresenta sequer mínimos

locais. Além disso, Hoffmann também constatou que para instâncias de problemas em vários desses domínios, a distância entre um estado qualquer e a saída do platô é limitada por um valor constante. Isso significa dizer que, por mais complexo que sejam os problemas para esses domínios, a distância de qualquer estado para a saída do platô é constante, o que faz com que esses problemas sejam facilmente resolvidos pelos sistemas de planejamento baseados em busca local.

Como conseqüência dessas conclusões, surgiu a seguinte questão: foram identificadas propriedades estruturais que tornam o planejamento uma tarefa factível ou, os domínios utilizados até então para fins de teste são muito fáceis ? Provavelmente ambas, mas na opinião de Hoffmann, os domínios de teste são muito mais simplistas e artificiais do que realistas.

Os domínios analisados por Hoffmann [22] são anteriores aos novos domínios criados para a competição *IPC-2002*. Foi necessário então, utilizar a formalização teórica apresentada na seção 6.2 para obter uma análise dos domínios da competição *IPC-2002* de interesse para este trabalho.

Nas subseções a seguir são apresentadas algumas proposições que permitem tirar algumas conclusões sobre as características dos domínios estudados. Tais conclusões serão importantes para ratificar os resultados obtidos de forma experimental, que serão apresentados no Capítulo 7.

6.4.1 Domínio do mundo dos blocos

A seguir, algumas proposições referentes à análise da topologia do domínio do mundo dos blocos, conforme a descrição STRIPS apresentado na figura 2.6. As demonstrações para as proposições estão detalhadas na seção D.1 dos apêndices.

Proposição 1 *O operador $Pegar(x)$ do domínio do mundo dos blocos é inversível.*

Proposição 2 *O operador $Soltar(x)$ do domínio do mundo dos blocos é inversível.*

Proposição 3 *O operador $Desempilhar(x, y)$ do domínio do mundo dos blocos é inversível.*

Proposição 4 *O operador Empilhar(x, y) do domínio do mundo dos blocos é inversível.*

Proposição 5 *Qualquer problema no domínio do mundo dos blocos é uma tarefa de planejamento bidirecional.*

Proposição 6 *Tarefas de planejamento no domínio do mundo dos blocos contém mínimos locais quando estados de seu espaço são avaliados pela heurística do planejador FF.*

A Proposição 5 nos permite concluir que os problemas nesse domínio devem ser facilmente solucionados pelo planejador *FF*. No entanto, a Proposição 6 indica que, para certas instâncias de problemas, a tarefa de planejamento pode apresentar estados de busca que sejam um mínimo local. De acordo com a definição de um mínimo local apresentada na Subseção 6.1.2, esse é um tipo de platô que não tem saídas. Se o sistema *FF*, durante o processo de planejamento, escolher uma ação para o plano solução cujo estado resultante seja um mínimo local, o processo de planejamento irá falhar. Ou seja, mesmo o domínio do mundo dos blocos sendo bidirecional, o sistema *FF* não tem como reverter o efeito de uma ação que conduziu a busca para um estado que é um mínimo local, pois isso significa regressar a uma estado de maior valor de heurística. Por esse motivo, o sistema *FF* pode não ser capaz de resolver todos os problemas nesse domínio.

6.4.2 Domínio do Satélite

A seguir, algumas proposições referentes à análise da topologia do domínio do Satélite, conforme a descrição PDDL para a versão STRIPS do problema apresentada na Seção C.1 dos apêndices. As demonstrações para as proposições estão detalhadas na Seção D.2 dos apêndices.

Proposição 7 *O operador turn_to(s, x, y) do domínio do Satélite é inversível.*

Proposição 8 *O operador switch_on(i, s) do domínio do Satélite é minimamente inversível.*

Proposição 9 *O operador switch_off(i, s) do domínio do Satélite é minimamente inversível.*

Proposição 10 *O operador calibrate(s, i, x) do domínio do Satélite é minimamente inversível.*

Proposição 11 *O operador take_image(s, x, i, m) do domínio do Satélite tem efeitos de adição estáticos.*

Proposição 12 *Qualquer problema no domínio do Satélite é uma tarefa de planejamento livre de becos sem saída.*

Proposição 13 *Tarefas de planejamento no domínio do Satélite não contém mínimos locais quando estados de seu espaço são avaliados pela heurística do sistema FF.*

A Proposição 12 nos permite concluir que os problemas nesse domínio devem ser facilmente solucionados pelo sistema *FF*. Isso fica patente pela Proposição 13, que garante que o sistema *FF* não irá falhar ao tentar solucionar qualquer instância de problema nesse domínio, pois o espaço de estados não contém mínimos locais.

6.4.3 Domínio do Robô de Exploração

A seguir, algumas proposições referentes à análise da topologia do domínio do Robô de Exploração, conforme a descrição PDDL para a versão STRIPS do problema apresentada na Seção C.5 dos apêndices. As demonstrações para as proposições estão detalhadas na Seção D.3 dos apêndices.

Proposição 14 *O operador navigate(r, x, y) do domínio do Robô de Exploração é inversível.*

Proposição 15 *O operador sample_soil(r, s, x) do domínio do Robô de Exploração tem efeitos de adição estáticos e efeitos de remoção irrelevantes.*

Proposição 16 *O operador sample_rock(r, s, x) do domínio do Robô de Exploração tem efeitos de adição estáticos e efeitos de remoção irrelevantes.*

Proposição 17 *O operador drop(r, s) do domínio do Robô de Exploração é minimamente inversível.*

Proposição 18 *O operador calibrate(r, c, o, x) do domínio do Robô de Exploração é minimamente inversível.*

Proposição 19 *O operador take_image(r, x, o, c, m) do domínio do Robô de Exploração tem efeitos de adição estáticos.*

Proposição 20 *O operador communicate_soil_data(r, l, x, y, z) do domínio do Robô de Exploração tem efeitos de adição estáticos.*

Proposição 21 *O operador communicate_rock_data(r, l, x, y, z) do domínio do Robô de Exploração tem efeitos de adição estáticos.*

Proposição 22 *O operador communicate_image_data(r, l, o, m, x, y) do domínio do Robô de Exploração tem efeitos de adição estáticos.*

Proposição 23 *Qualquer problema no domínio do Robô de Exploração é uma tarefa de planejamento livre de becos sem saída.*

Proposição 24 *Tarefas de planejamento no domínio do Robô de Exploração não contém mínimos locais quando estados de seu espaço são avaliados pela heurística do sistema FF.*

A Proposição 23 nos permite concluir que os problemas nesse domínio devem ser facilmente solucionados pelo sistema *FF*. Isso fica patente pela Proposição 24, que garante que o sistema *FF* não irá falhar ao tentar solucionar qualquer instância de problema nesse domínio, pois o espaço de estados não contém mínimos locais.

Pela análise desses três domínios STRIPS, o planejador *FF* deve resolver todos os problemas no domí nio do Satélite e no domínio do Robô de Exploração mas, se atingir mínimos locais, não será capaz de resolver todos os problemas no domínio do mundo dos blocos.

6.4.4 Domínio do Satélite com variáveis numéricas

A seguir, algumas proposições referentes à análise da topologia do domínio do Satélite com variáveis numéricas, conforme a descrição PDDL para a versão numérica do problema apresentada na Seção C.3 dos apêndices.

Proposição 25 *O operador $turn_to(s, x, y)$ do domínio do Satélite numérico não tem efeitos numéricos irrelevantes.*

Demonstração da Proposição 25:

O operador $turn_to(s, x, y)$ do domínio do Satélite decrementa de uma constante o valor da variável $fuel$. Essa variável é utilizada em uma pré-condição da própria ação $turn_to(s, x, y)$, o que não satisfaz a condição da Definição 24. Portanto, essa essa ação não tem efeitos numéricos irrelevantes ■

Proposição 26 *Qualquer problema no domínio do Satélite com variáveis numéricas é uma tarefa de planejamento com becos sem saída.*

Demonstração da Proposição 26:

De acordo com a Proposição 25, o operador $turn_to(s, x, y)$ do domínio do Satélite numérico não tem efeitos numéricos irrelevantes. Como essa ação não satisfaz as condições do Lema 4, conclui-se que qualquer problema nesse domínio é uma tarefa de planejamento com becos sem saída ■

Proposição 27 *A função heurística do sistema Metric-FF não consegue reconhecer todos os becos sem saída em problemas no domínio do Satélite com variáveis numéricas.*

Demonstração da Proposição 27:

Sem perda de generalidade, considere um estado s em que um único satélite x está apontando na direção d_0. Existem apenas dois objetivos $have_image(d_1, m)$ e $have_image(d_2, m)$. Se qualquer ação para movimentar o satélite x entre dois alvos quaisquer consumir a mesma quantidade q de combustível, e se a quantidade de combustível no estado s for igual a $2q - 1$, claramente não há um plano solução para essa sub-tarefa de planejamento. Logo, o estado s é um beco sem saída.

Se for utilizado o algoritmo de construção do grafo de planejamento para a tarefa numérica relaxada empregada pelo planejador *Metric-FF*, as ações $turn_to(x, d_0, d_1)$ e $turn_to(x, d_0, d_2)$ serão aplicáveis no primeiro nível do grafo. Como ignora efeitos de decremento das duas ações $turn_to$, o consumo de combustível das mesmas é ignorado no grafo para a tarefa relaxada. Assim, as duas ações serão escolhidas pelo algoritmo de busca da solução para a tarefa relaxada, e o beco sem saída não é identificado pela função heurística do planejador *Metric-FF* ■

A função heurística do sistema *Metric-FF* não consegue reconhecer todos os becos sem saída, pois os efeitos de decremento são ignorados para a tarefa relaxada. Em função disso, um plano qualquer para uma tarefa relaxada pode conter ações que, quando escolhidas para o plano solução levam ao esgotamento de alguma variável numérica, mesmo que a solução relaxada seja válida.

Todas essas características fazem desse domínio um desafio muito maior para um sistema de planejamento, pois a topologia do espaço de busca exige muito mais critério do planejador no momento em que esse faz uma escolha. Sistemas de planejamento como o *Metric-FF* podem ser seriamente afetados por tarefas de planejamento para esse domínio. Como faz planejamento por meio de um algoritmo de busca local, e que portanto não tem um mecanismo de retrocesso (*backtracking*), qualquer escolha incorreta inevitavelmente conduzirá a busca a um mínimo local, ou pior, a um beco sem saída não reconhecido. A análise apresentada nos permite concluir que o planejador *Metric-FF* não será capaz de solucionar todos os problemas nesse domínio.

6.4.5 Domínio do Robô de Exploração com variáveis numéricas

A seguir, algumas proposições referentes à análise da topologia do domínio do Robô de Exploração com variáveis numéricas, conforme a descrição PDDL para a versão numérica do problema apresentada na Seção C.7 dos apêndices.

Proposição 28 *O operador navigate(r, x, y) do domínio do Robô de Exploração numérico não tem efeitos numéricos irrelevantes.*

Demonstração da Proposição 28:

O operador *navigate*(r, x, y) do domínio do Satélite decrementa de uma constante o valor da variável *energy*. Essa variável é utilizada em uma pré-condição da própria ação *navigate*(r, x, y), o que não satisfaz a condição da Definição 24. Portanto, essa essa ação não tem efeitos numéricos irrelevantes ■

Proposição 29 *Qualquer problema no domínio do Robô de Exploração com variáveis numéricas é uma tarefa de planejamento com becos sem saída.*

Demonstração da Proposição 29:

De acordo com a Proposição 28, o operador *navigate*(r, x, y) do domínio do Robô de Exploração numérico não tem efeitos numéricos irrelevantes. Como essa ação não satisfaz as condições do Lema 4, conclui-se que qualquer problema nesse domínio é uma tarefa de planejamento com becos sem saída ■

Proposição 30 *A função heurística do sistema Metric-FF não consegue reconhecer todos os becos sem saída em problemas no domínio do Robô de Exploração com variáveis numéricas.*

Demonstração da Proposição 30:

Sem perda de generalidade, considere um estado s em que um único robô r está na

localidade d_0, e uma única estação-base l está na localidade d_1. Existe apenas um objetivo *communicated_soil_data*(d_2). Como a ação para movimentar o robô r entre dois alvos quaisquer consome a mesma quantidade q de energia, e se a quantidade de energia no estado s for igual a $2q - 1$, claramente não há um plano solução para essa sub-tarefa de planejamento. Logo, o estado s é um beco sem saída.

Se for utilizado o algoritmo de construção do grafo de planejamento para a tarefa numérica relaxada empregada pelo planejador *Metric-FF*, as ações *navigate*(r, d_0, d_1) e *navigate*(r, d_0, d_2) serão aplicáveis no primeiro nível do grafo. Como ignora efeitos de decremento das duas ações *navigate*, o consumo de energia das mesmas é ignorado no grafo para a tarefa relaxada. Assim, as duas ações serão escolhidas pelo algoritmo de busca da solução para a tarefa relaxada, e o beco sem saída não é identificado pela função heurística do planejador *Metric-FF* ■

A função heurística do sistema *Metric-FF* não consegue reconhecer todos os becos sem saída, pois os efeitos de decremento são ignorados para a tarefa relaxada. Em função disso, um plano qualquer para uma tarefa relaxada pode conter ações que, quando escolhidas para o plano solução levam ao esgotamento de alguma variável numérica, mesmo que a solução relaxada seja válida.

Assim como para o domínio do Satélite com variáveis numéricas, todas essas características fazem desse domínio um desafio muito maior para um sistema de planejamento. A análise apresentada nos permite concluir que o planejador *Metric-FF* não será capaz de solucionar todos os problemas nesse domínio.

6.4.6 Taxonomia dos domínios analisados

A partir dos resultados da análise dos domínios de planejamento, realizada neste capítulo, é possível classificá-los de acordo com uma taxonomia. A Figura 6.2 classifica os domínios analisados neste trabalho, distribuídos em linhas e colunas, definindo assim a taxonomia de domínios para os planejadores *FF* e *GMetric-FF*. Na primeira coluna estão os domínios com espaço de estados bidirecional, na segunda coluna os com espaço de estados inofensivo, na terceira coluna os com espaço de estados com becos sem saída reconhecíveis pela função heurística, e na quarta coluna os com becos sem saída não reconhecíveis. A linha superior apresenta os domínios cujos espaços de

estados possuem mínimos locais, enquanto a linha inferior apresenta os domínios sem mínimos locais. De acordo com a análise, os planejadores *FF* e *GMetric-FF* devem ser capazes de resolver todos os problemas cujos domínios estiverem nas três primeiras colunas da linha inferior, as regiões da figura em cinza. No entanto, qualquer domínio fora dessas regiões da taxonomia pode apresentar instâncias de problemas em que esses planejadores não são capazes de encontrar um plano solução.

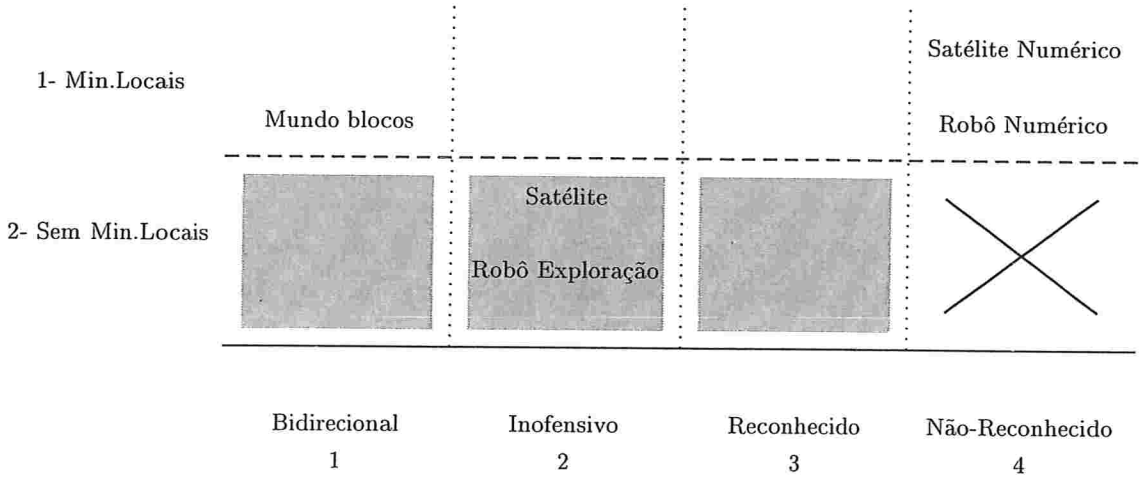

Figura 6.2: Taxonomia dos domínios analisados. As regiões em cinza indicam domínios em que os planejadores estudados devem ser capazes de resolver todas as instâncias de problemas. A região marcada com um X, indica que não existem domínios sem mínimos locais e com becos não-reconhecíveis. De acordo com a teoria estudada, todo domínio com becos não-reconhecíveis deve possuir mínimos locais.

Desta forma, podemos prever o comportamento dos planejadores *FF* e *GMetric-FF*, nos domínios classificados, a saber:

1. **Domínios com becos sem saída** (colunas 3 e 4). Algumas instâncias de problemas nesses domínios não podem ser resolvidas por planejadores que fazem busca local. Como foi visto, se a função heurística do planejador for capaz de detectar todos os estados que são becos sem saída, o planejador evitará tais estados, porém, só será possível resolver todos os problemas para esse tipo de domínio se

o mesmo não apresentar mínimos locais. Foi provado que os domínios STRIPS analisados, o Mundo dos Blocos, o Satélite e o Robô de Exploração, não possuem becos sem saída, e estão portanto nas categorias bidirecional e inofensivo da taxonomia (colunas 1 e 2). **Assim, espera-se que o planejador *FF* seja capaz de encontrar um plano solução para todas as instâncias de problemas nos domínios STRIPS do Satélite e o Robô de Exploração.** Também foi provado que os domínios numéricos analisados, o Satélite e o Robô de Exploração, possuem becos sem saída não-reconhecidos pela função heurística do planejador *GMetric-FF* (coluna 4). **Espera-se que o planejador *GMetric-FF* não encontre um plano solução para algumas instâncias de problemas nos domínios numéricos do Satélite e do Robô de Exploração.** Estas duas previsões serão verificadas nos resultados experimentais do próximo capítulo;

2. **Domínios com mínimos locais** (linha 1). Algumas instâncias de problemas nesses domínios não podem ser resolvidas por planejadores que fazem busca local se o caminho que liga o estado inicial ao objetivo passar por um estado que é um mínimo local. Foi provado que o domínio do Mundo dos Blocos possui mínimos locais quando é considerada a função heurística do planejador *FF*. Uma vez que os planejadores em geral são incapazes de evitar estados que são mínimos locais, **espera-se que o planejador *FF* não seja capaz de encontrar um plano solução para todas as instâncias de problemas no domínio Mundo dos Blocos.** Os domínios numéricos estudados possuem mínimos locais pois, da teoria de Hoffmann, quando um domínio tem becos sem saída não-reconhecidos também existem mínimos locais. **Espera-se que o planejador *GMetric-FF* também não seja capaz de encontrar um plano solução para todas as instâncias de problemas nos domínios numéricos do Satélite e do Robô de Exploração.** Estas previsões também serão verificadas nos resultados experimentais do próximo capítulo.

6.5 Considerações finais

Nesse capítulo foi apresentada a teoria de Hoffmann, desenvolvida para análises teóricas de domínios de planejamento descritos na linguagem STRIPS. Também foi apresentada

uma extensão dessa teoria para analisar domínios que envolvem variáveis numéricas. A teoria de Hoffmann e sua extensão foram utilizadas para analisar os domínios do Mundo dos Blocos, do Satélite e do Robô de Exploração, em suas versões STRIPS e numérica, tendo como resultado final uma taxonomia desses domínios. Essa taxonomia permite efetuar previsões sobre o comportamento dos planejadores *FF* e *GMetric-FF* nos domínios estudados.

No próximo capítulo são apresentados os resultados obtidos a partir de testes com as implementações dos planejadores *FF* e *GMetric-FF*, nos domínios estudados. Os resultados das análises teóricas apresentados nesse capítulo, e sintetizados na taxonomia dos domínios, são utilizados para explicar, a partir dos resultados empíricos, o comportamento dos planejadores de busca local nesses domínios.

Capítulo 7

Resultados experimentais

Neste capítulo são apresentados os resultados empíricos obtidos com os planejadores *FF* e *GMetric-FF* implementados para os domínios analisados.

De acordo com a análise teórica apresentada no capítulo anterior, o comportamento dos planejadores *FF* e *GMetric-FF* implementados pode ser interpretado da seguinte forma:

1. nos domínios do Satélite e do Robô de Exploração STRIPS, espera-se que **todas** as instâncias de problemas sejam resolvidas com o planejador *FF*;

2. no domínio do Mundo dos Blocos, espera-se que o planejador *FF* não seja capaz de encontrar um plano solução para **todas** as instâncias de problemas e pode-se afirmar que para tais problemas, o algoritmo de busca passa por um estado que pertence a um mínimo local ou o tempo para encontrar uma solução é muito grande;

3. nos domínios do Satélite e do Robô de Exploração com variáveis numéricas, espera-se que o planejador *GMetric-FF* não seja capaz de encontrar um plano solução para **todas** as instâncias de problemas. Pode-se afirmar então que para tais problemas, o algoritmo de busca passa por um estado que pertence a um mínimo local ou um beco sem saída, ou o tempo para encontrar uma solução é muito grande.

7.1 Metodologia empregada nos testes

O planejador *FF* foi implementado em linguagem `C` de acordo com a especificação do algoritmo descrito no Capítulo 5. Esta implementação recebe como entrada uma descrição de ações totalmente instanciadas, ou seja, livres de variáveis. Para tal fim, foi criado um programa auxiliar, implementado em linguagem `PERL`, que faz todas as possíveis instanciações para as ações de um domínio descrito na linguagem `PDDL`, transformando-as em ações proposicionais. Esse programa, cujo algoritmo foi descrito na Seção 5.3, recebe como entrada uma descrição `PDDL` de um domínio e uma descrição de um problema e, gera uma saída formatada que serve de entrada para o planejador *FF*.

O planejador *GMetric-FF* também foi implementado em linguagem `C`, tendo sido utilizadas todas as estruturas de dados do planejador *FF*. O programa que transforma em ações proposicionais as ações de um domínio descrito na linguagem `PDDL` também foi estendido. Essa extensão unifica as variáveis numéricas a cada instanciação dos predicados de uma ação `PDDL`.

Os dois planejadores foram submetidos a um conjunto de quinze problemas em cada domínio, sendo que foram avaliados os seguintes aspectos:

- tamanho do domínio em quantidade de ações proposicionais;
- tempo médio gasto para encontrar um plano solução;
- tamanho do plano solução encontrado.

O processo de transformação das ações `PDDL` em ações proposicionais foi executado uma única vez para cada problema. Já o processo de planejamento propriamente dito, foi executado dez vezes para o mesmo problema, sendo que foi considerado o tempo médio de planejamento para a elaboração dos gráficos de desempenho. A partir dos dados obtidos nos testes empíricos foram gerados alguns gráficos com o aplicativo `GNUplot`, apresentados na próxima seção.

Os problemas utilizados nos testes são os mesmos empregados nas competições *IPC-2000* e *IPC-2002*, e podem ser obtidos nos seguintes endereços eletrônicos: http://www.cs.toronto.edu/aip

e http://www.dur.ac.uk/d.p.long/competition.html. A seleção de problemas foi feita, pelos organizadores da competição, de forma que as dificuldades inerentes a cada um dos domínios estivessem presentes. Além disso, no conjunto de problemas propostos para cada domínio, essas dificuldades crescem de acordo com o tamanho da instância, embora isso não seja uma regra. No apêndice C, juntamente com as descrições PDDL dos domínios do *IPC-2002*, é apresentado um problema exemplo para cada um dos domínios.

7.2 Análise experimental nos domínios STRIPS

7.2.1 Sobre a qualidade do planejador *FF* implementado

Inicialmente, foi feita uma comparação do planejador *FF* implementado com o planejador utilizado na competição *IPC-2002*, a fim de avaliar a sua qualidade. Para tal, foram comparados os tamanhos dos planos gerados pelas duas implementações nos mesmos problemas. Os dados da execução do planejador *FF* utilizado na competição foram obtidos da página *Web* da competição, no endereço citado anteriormente.

As Figuras 7.1 e 7.2 mostram que o planejador implementado teve um comportamento muito semelhante ao do planejador *FF* usado na competição, nos domínios do Satélite e do Robô de Exploração respectivamente. As pequenas diferenças podem ser justificadas por uma técnica adicional empregada por Hoffmann em sua implementação, que faz uma análise e ordenação dos objetivos de forma a tornar o processo mais eficiente. Porém, considerando a aproximação das duas curvas nas Figuras 7.1 e 7.2, pode-se dizer que, para o propósito desse trabalho, a versão do planejador *FF* construída pelo autor, implementa o algoritmo proposto por Hoffmann e considerado na análise teórica.

7.2.2 Geração de ações proposições nos domínios STRIPS

A Figura 7.3 apresenta as quantidades de ações totalmente instanciadas dos problemas para os três domínios STRIPS estudados. A quantidade de ações de um domínio é importante pois, qualquer sistema de planejamento deve, a cada novo estado expandido,

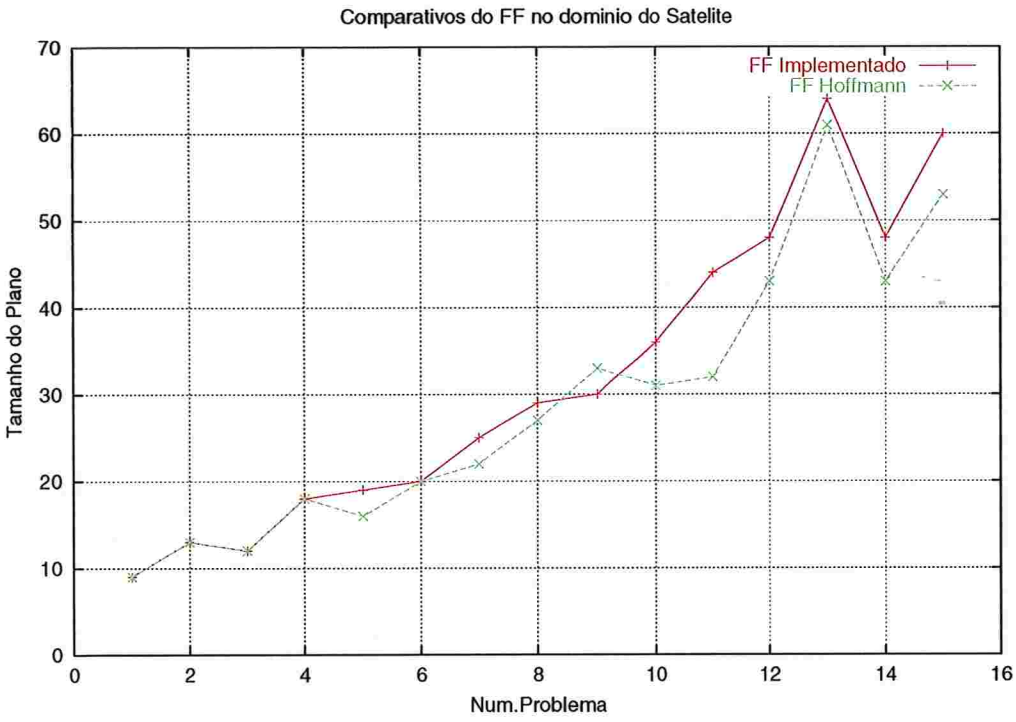

Figura 7.1: Tamanhos dos planos encontrados pelo planejador *FF* implementado e pelo planejador da competição em problemas no domínio do Satélite

percorrer todo o conjunto de ações para decidir quais são aplicáveis. Ou seja, o fator de ramificação da árvore de busca é diretamente proporcional à quantidade de ações aplicáveis em cada estado, e para testar a aplicabilidade é necessário testar a satisfação da pré-condição para cada uma das ações do problema. O fator de ramificação também tem uma influência direta na quantidade de memória utilizada pelo planejador *FF*, pois o algoritmo *Hill-Climbing* reforçado empregado por esse, utiliza uma busca em largura para encontrar um estado de melhor valor da função heurística (Seção 5.1).

Note que o crescimento exponencial da quantidade de ações do domínio do Satélite (Figura 7.3) sugere que este seja o domínio com maior dificuldade para se planejar. No entanto, é o domínio do Mundo dos Blocos que apresentou mais dificuldades para o planejador *FF*, como será visto a seguir.

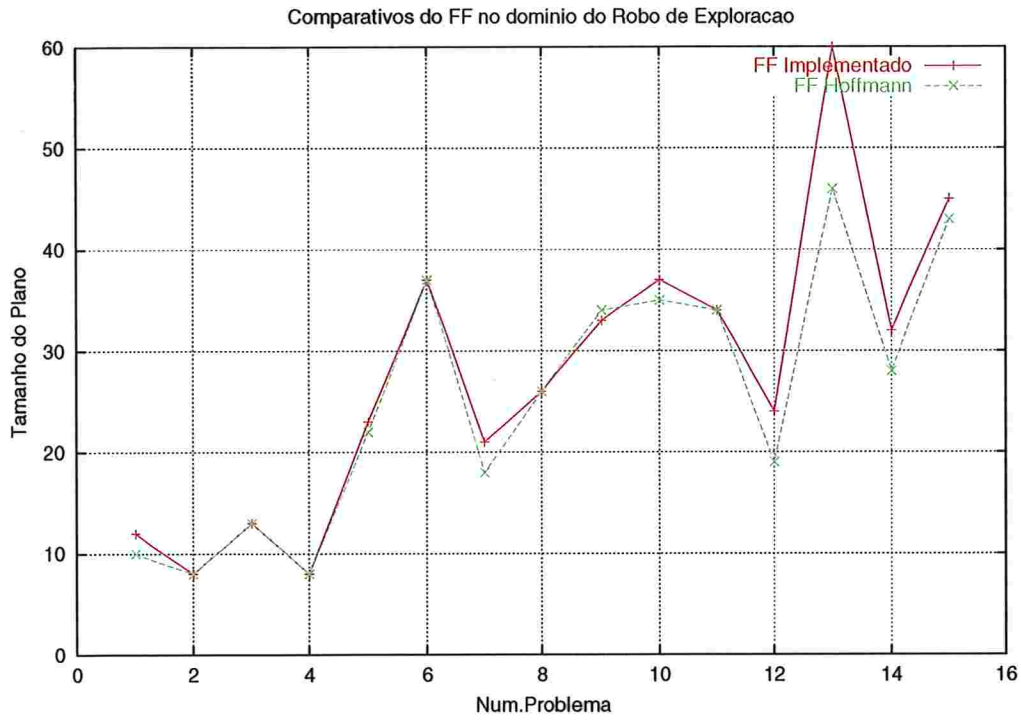

Figura 7.2: Tamanhos dos planos encontrados pelo planejador *FF* implementado e pelo planejador da competição em problemas no domínio do do Robô de Exploração

7.2.3 Problemas resolvidos nos domínios STRIPS

As Figuras 7.4 e 7.5 mostram, respectivamente, o tempo de planejamento e o tamanho dos planos encontrados pelo planejador *FF* nos domínios STRIPS. Conforme a convenção adotada nas competições *IPC-2000* e *IPC-2002*, tanto nas curvas de tempo de planejamento como nas de tamanho do plano, para os problemas em que o planejador não encontra uma solução, adota-se tempo e tamanho de plano iguais a zero.

Na Figura 7.4, problemas que consomem tempo próximo de zero podem ser: (i) problemas para os quais o planejador não encontrou uma solução ou (ii) problemas para os quais o tempo para encontrar uma solução foi muito pequeno. Para decidir em qual das duas situações um problema se encontra, é necessário analisar a Figura 7.5,

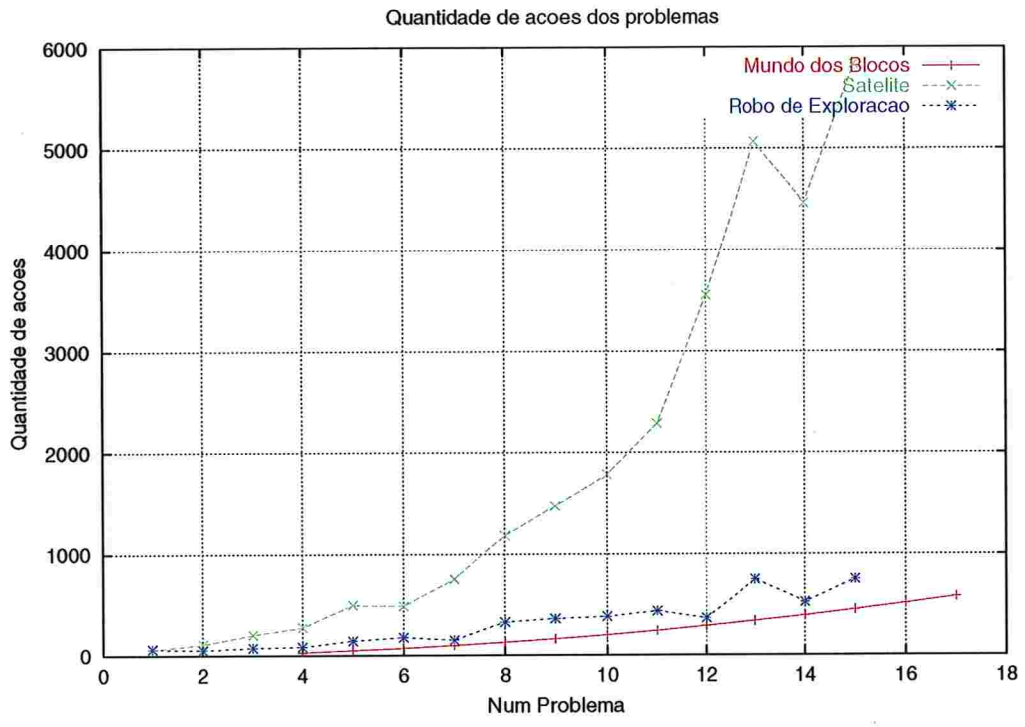

Figura 7.3: Quantidade de ações totalmente proposicionais dos problemas STRIPS resolvidos pelo planejador *FF*

em que os problemas para os quais o planejador *FF* encontrou uma solução, possuem tamanho de plano diferente de zero (de acordo com a convenção apresentada). Assim, pode-se observar que, **para todos os problemas nos domínios do Satélite e Robô de Exploração, o planejador *FF* encontrou uma solução**. Esse era exatamente o comportamento esperado segundo a análise do capítulo anterior, onde foi provado que esses domínios não apresentam becos sem saída, nem mínimos locais. Já para o domínio do Mundo dos Blocos (Figuras 7.5 e 7.4), o planejador FF não foi capaz de resolver todos os problemas: nos problemas com mais de 9 blocos, ou não foi possível encontrar uma solução (problemas 13, 15, 16 e 17), ou o tempo para encontrá-la foi muito grande. Esse também era o comportamento esperado, segundo a analise realizada, **para o domínio do Mundo dos Blocos que apesar de não apresentar becos sem saída, possui mínimos locais**.

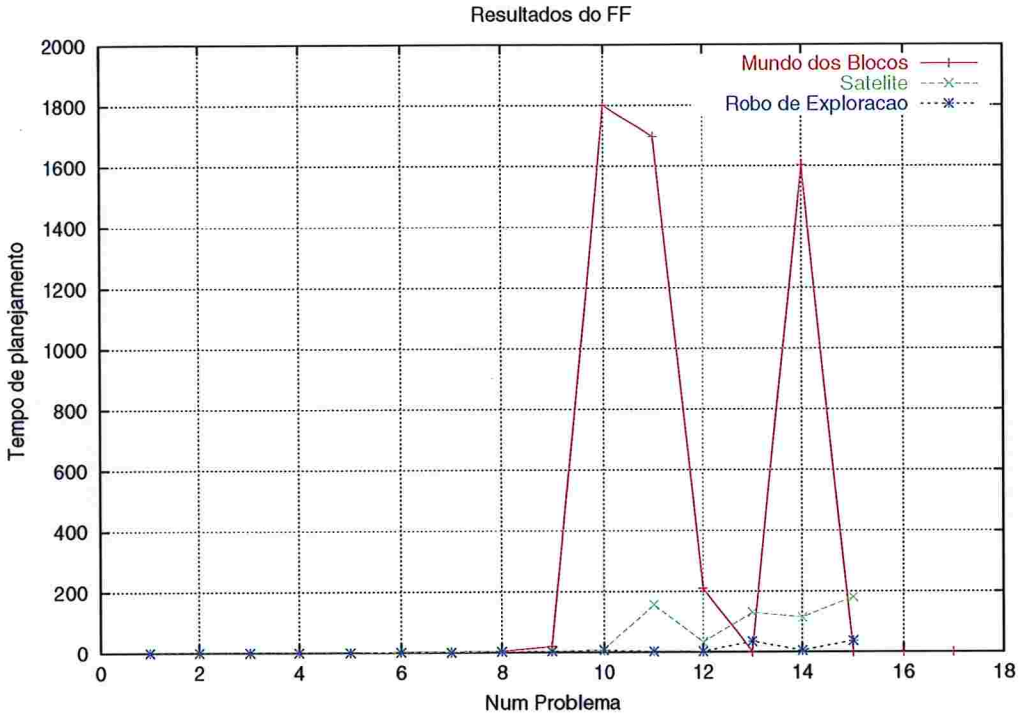

Figura 7.4: Tempo médio gasto para encontrar um plano para os problemas resolvidos pelo planejador *FF*. Os tempos foram calculados como a média dos tempos gastos em dez execuções do planejador para cada um dos problemas.

Apesar da análise deste trabalho se deter a prever a capacidade de um planejador em resolver todos os problemas de um determinado domínio, pode-se explicar porquê o FF gasta muito tempo para resolver alguns dos problemas no domínio do Mundo dos Blocos (problemas 10, 11 e 14). Isso se deve ao fato de que, para alguns problemas nesse domínio, o planejador *FF* necessita descer muitos níveis da busca em largura feita pelo algoritmo de busca *Hill-Climbing* reforçado. Tal comportamento está essencialmente relacionado com a estrutura desses problemas, ou seja, com a disposição dos blocos no estado inicial e no objetivo.

Com esses resultados, podemos concluir que as versões STRIPS dos domínios do Satélite e do Robô de Exploração são mais simples do que o domínio do Mundo dos

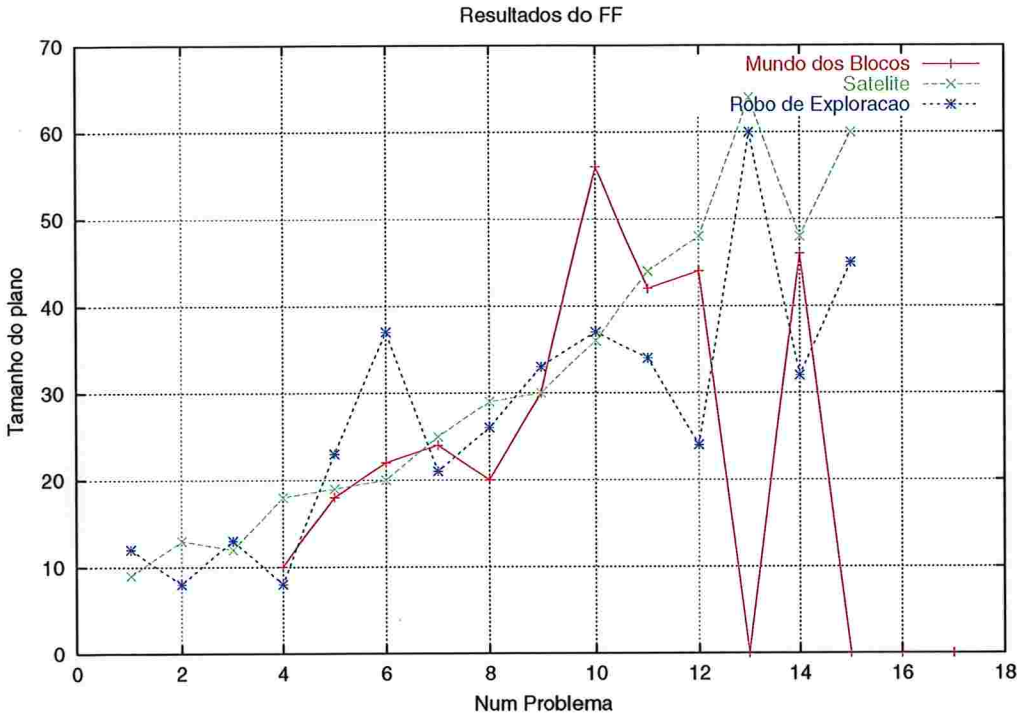

Figura 7.5: Tamanho do plano encontrado para os problemas resolvidos pelo planejador *FF*. Nos problemas em que não foi possível encontrar uma solução, o tamanho do plano é igual a zero.

Blocos, já que esse último apresenta mínimos locais, fazendo com que nem todas as instâncias de problemas nesse domínio sejam resolvidas pelo planejador *FF*.

Este é um resultado que pode surpreender a grande maioria dos pesquisadores, inclusive da área de planejamento. Novos domínios de planejamento foram criados para a competição *IPC-2002*, exatamente para propor desafios maiores que os domínios artificiais estudados até então, entre eles, o Mundo dos Blocos. No entanto, pelo menos em suas versões `STRIPS`, os domínios do Satélite e do Robô de Exploração são mais simples do que o domínio do Mundo dos Blocos.

7.3 Análise experimental nos domínios numéricos

7.3.1 Geração de ações proposições nos domínios numéricos

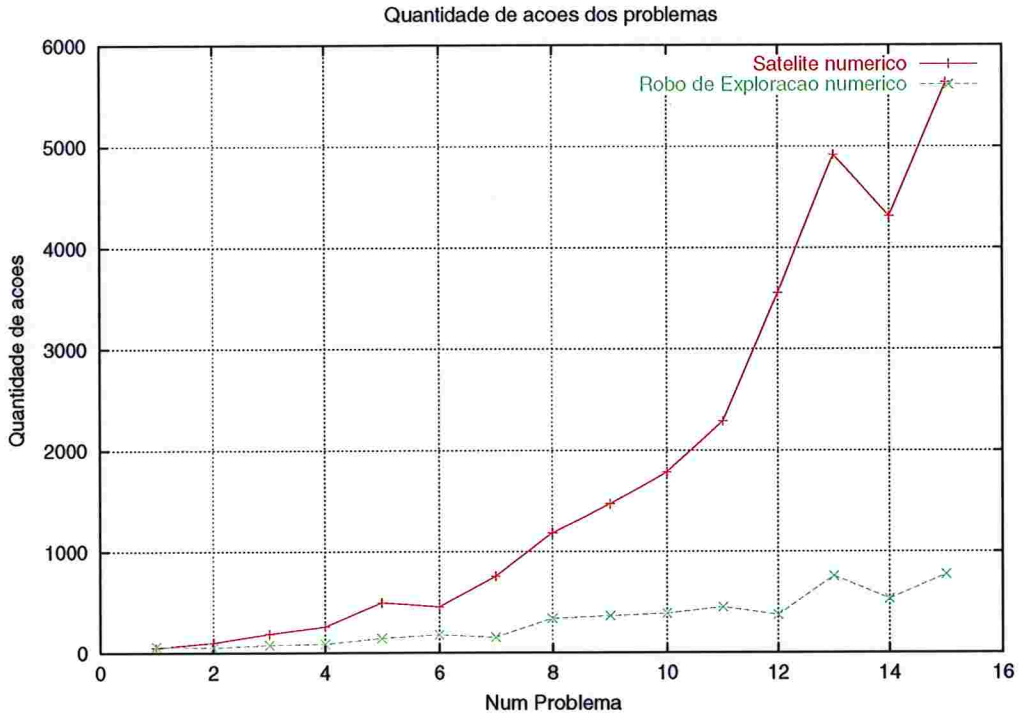

Figura 7.6: Quantidade de ações totalmente proposicionais dos problemas numéricos resolvidos pelo planejador *GMetric-FF*

A Figura 7.6, que apresenta as quantidades de ações totalmente instanciadas dos problemas para os dois domínios numéricos estudados, sugere que os problemas no domínio do Satélite apresentam maior dificuldade do que os problemas no domínio do Robô de Exploração. A mesma justificação apresentada na Subseção 7.2.2 para os domínios STRIPS é válida para os domínios numéricos, ou seja, pelo fato dos problemas no domínio do Satélite possuírem uma quantidade maior de ações proposicionais que os problemas no domínio do Robô de Exploração isso pode indicar maior dificuldade. No entanto, essa pode ser uma conclusão prematura, como será visto a seguir.

7.3.2 Problemas resolvidos nos domínios numéricos

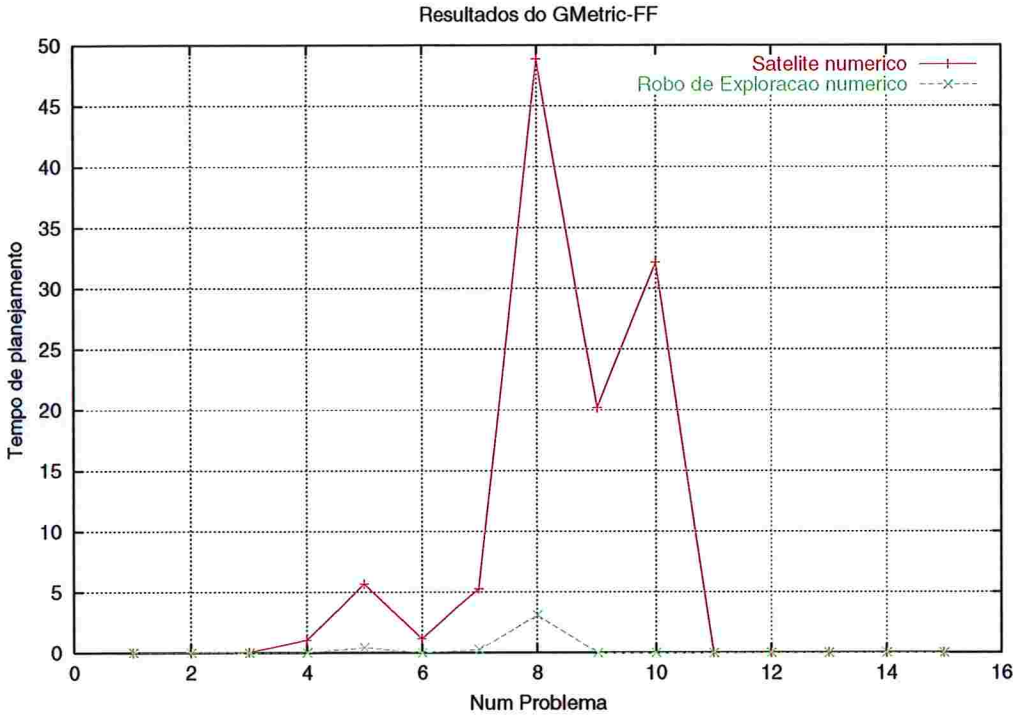

Figura 7.7: Tempo médio gasto para encontrar um plano para os problemas resolvidos pelo planejador *GMetric-FF*

As Figuras 7.7 e 7.8 mostram, respectivamente, o tempo médio de planejamento e o tamanho dos planos encontrados pelo planejador *GMetric-FF* nos domínios numéricos. **Observa-se que o planejador não resolveu todos os problemas para os domínios numéricos do Satélite e Robô de Exploração**. Esse era exatamente o comportamento esperado pois, de acordo com a análise teórica, foi demonstrado que esses domínios apresentam becos sem saída não-reconhecíveis. Como conseqüência disso, problemas nesses dois domínios também possuem mínimos locais. Tais características podem ser constatadas, conforme a Figura 7.8, para os problemas de 11 a 15, do domínio do Satélite, e os problemas 3, 6, 9, 10, 11, 12, 13, 14, 15, do domínio do Robô de Exploração, que apresentam tamanho de plano igual a zero, ou

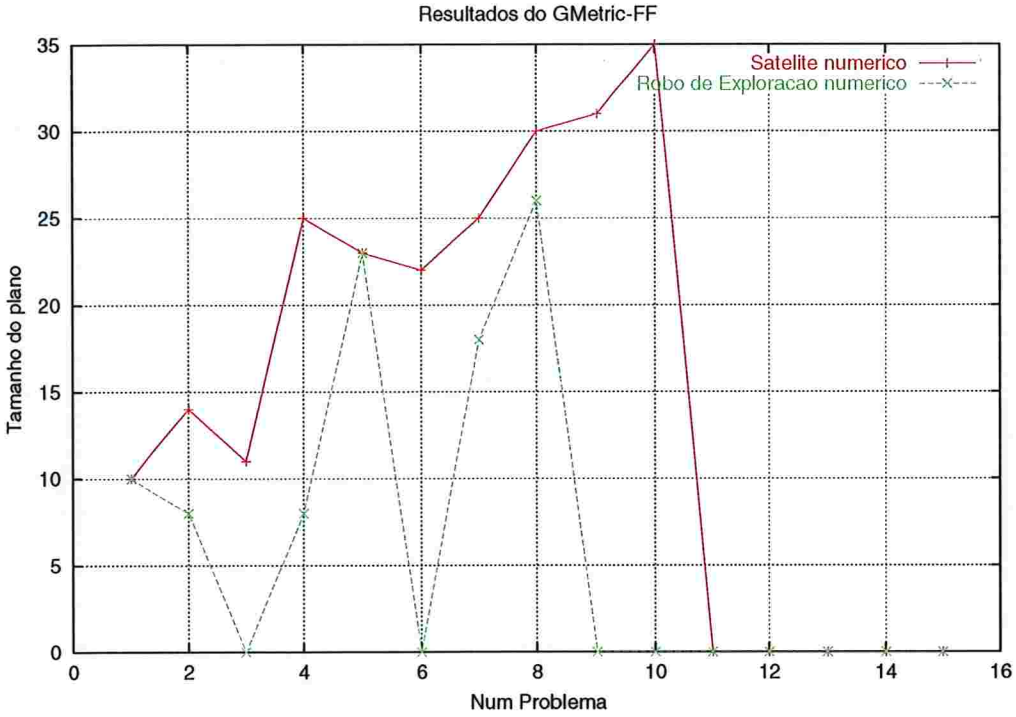

Figura 7.8: Tamanho do plano encontrado para os problemas resolvidos pelo planejador *GMetric-FF*

seja, não foram resolvidos pelo planejador implementado.

Nota-se ainda que o planejador *GMetric-FF* conseguiu resolver menos problemas no domínio do Robô de Exploração, ao contrário do que foi previsto na subseção anterior. Na seção 7.5 são dadas algumas possíveis justificativas para isso.

7.4 Resultados da competição IPC-2002

Os gráficos das Figuras 7.9 e 7.10 apresentam respectivamente, os tamanhos dos planos encontrados pelos planejadores que participaram da competição *IPC-2002* nos problemas nos domínios do Satélite e do Robô de Exploração com variáveis numéricas. Esses problemas são os mesmos que foram utilizados para os testes empíricos realizados neste

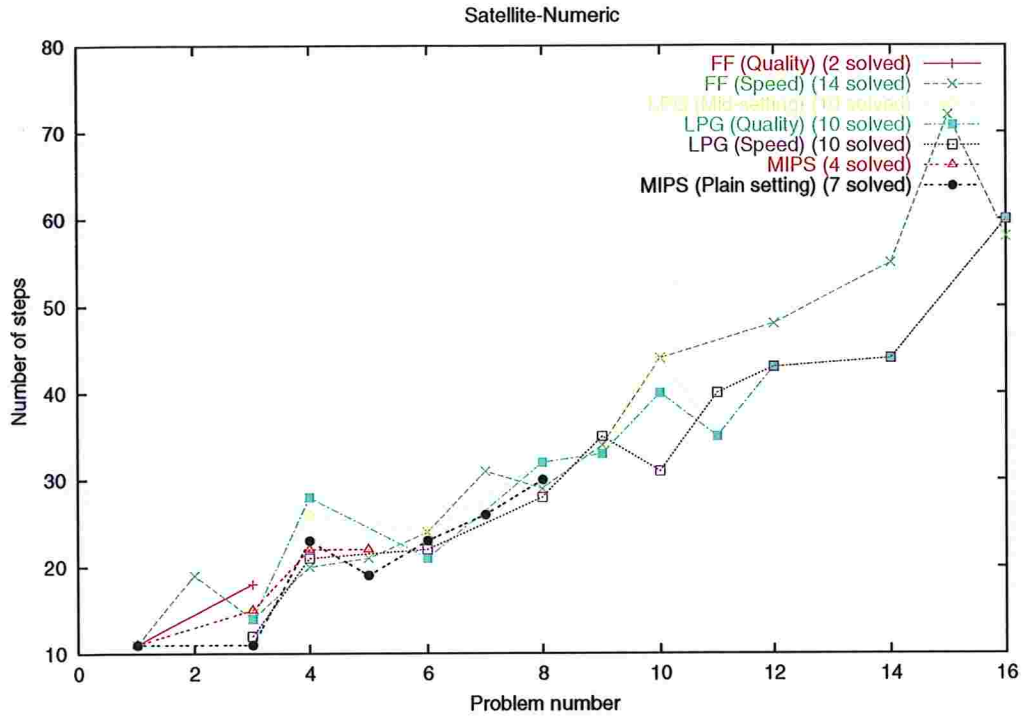

Figura 7.9: Resultados da competição *IPC-2002* para o domínio do Satélite com variáveis numéricas

trabalho. Os dados para a construção desses dois gráficos foram obtidos a partir de dados fornecidos pela página *Web* da competição. Para clareza da figura, não estão representados os casos em que os competidores falharam na busca por uma solução (problemas sem marca de plotagem).

Como pode ser observado, os planejadores participantes da competição também não tiveram sucesso na resolução de todos problemas nesses domínios, o que confirma os resultados da análise teórica realizada. O planejador *Metric-FF*, representado nos dois gráficos simplesmente como *FF*, foi o planejador que obteve os melhores resultados, resolvendo quatorze problemas no domínio do Satélite e dez problemas no domínio do Robô de Exploração. Na competição, foram fornecidos vinte problemas para cada domínio. O baixo desempenho se deve ao fato dos planejadores da competição também

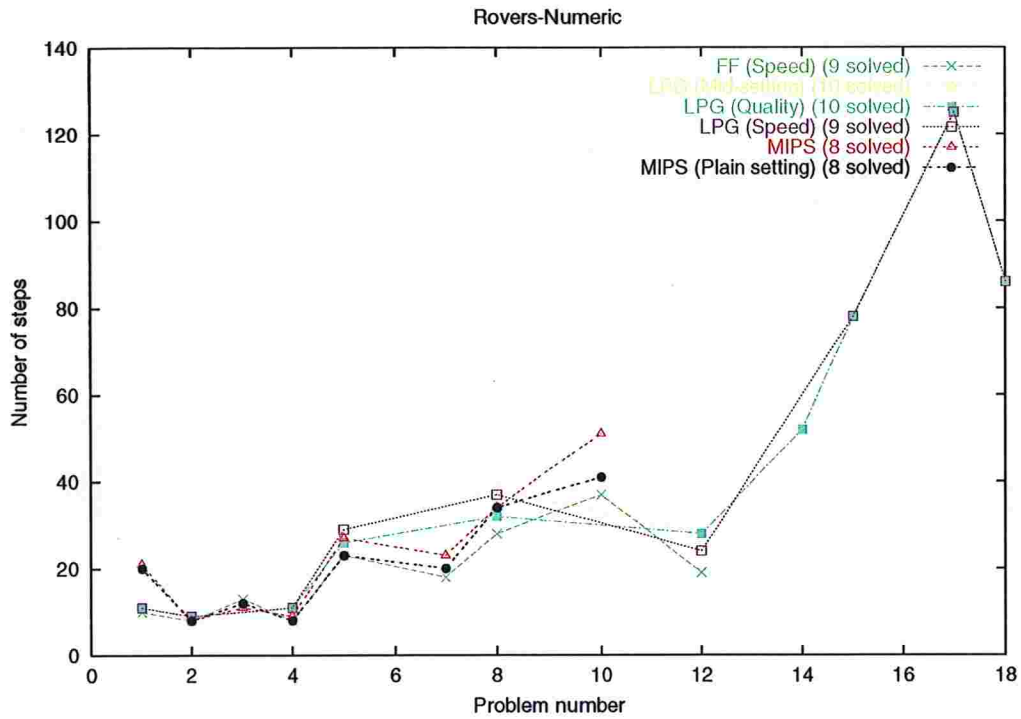

Figura 7.10: Resultados da competição *IPC-2002* para o domínio do Robô de Exploração com variáveis numéricas

não reconhecerem os becos sem saída, nem terem como contornar os mínimos locais. Isso significa que, os domínios com variáveis numéricas propostos de fato representam um grande desafio para os sistemas de planejamento atuais.

É importante salientar que os mesmos problemas não resolvidos pelo planejador *GMetric-FF* implementado também não foram resolvidos pela maioria dos competidores, e que no geral, o planejador implementado teve um desempenho bem semelhante ao dos competidores mais bem sucedidos. Isso indica que, no espaço de busca de tais problemas, existem caminhos que passam por mínimos locais ou becos sem saída, difíceis de serem detectados pela função heurística.

7.5 A heurística gulosa do *GMetric-FF*

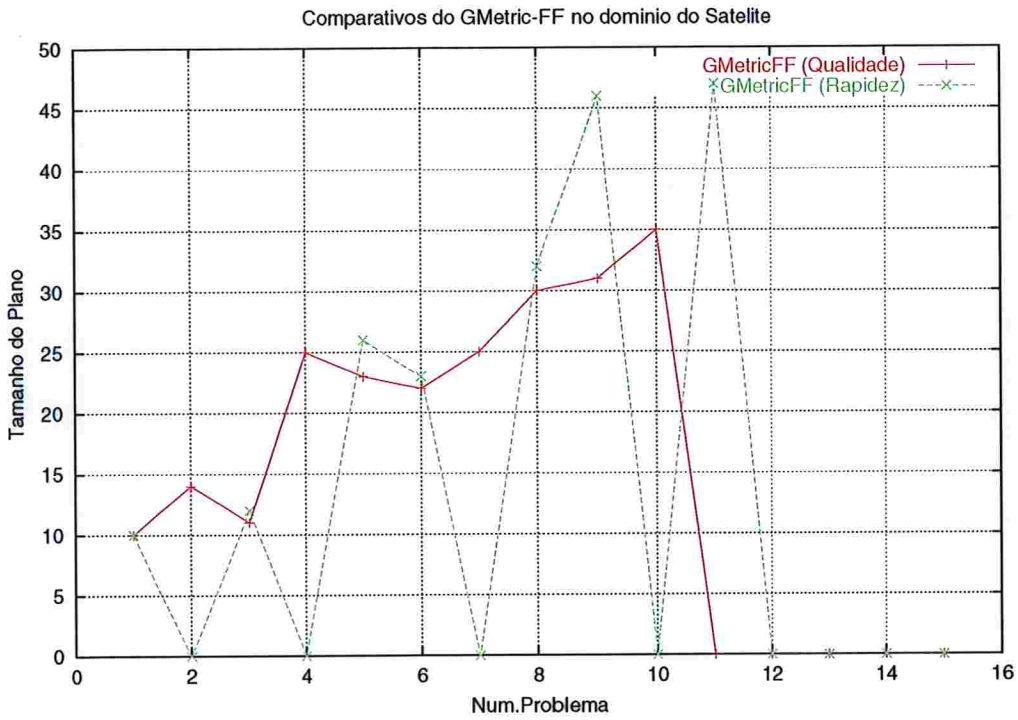

Figura 7.11: Tamanhos dos planos encontrados pelo planejador *GMetric-FF* implementado executado nos modos qualidade e rapidez em problemas no domínio do Satélite

Todos os resultados apresentados anteriormente foram obtidos executando o planejador implementado no modo qualidade. No entanto, conforme descrito na Seção 5.5, o planejador implementado possui dois modos de execução, a saber:

- **Modo Rapidez**. Encerra a busca em largura do algoritmo *Hill-Climbing* reforçado assim que encontrar um estado de menor heurística.
- **Modo Qualidade**. Expande todos os nós de uma mesma profundidade do algoritmo *Hill-Climbing* reforçado e, em caso de empate, escolhe o estado que melhor atende o critério de otimização.

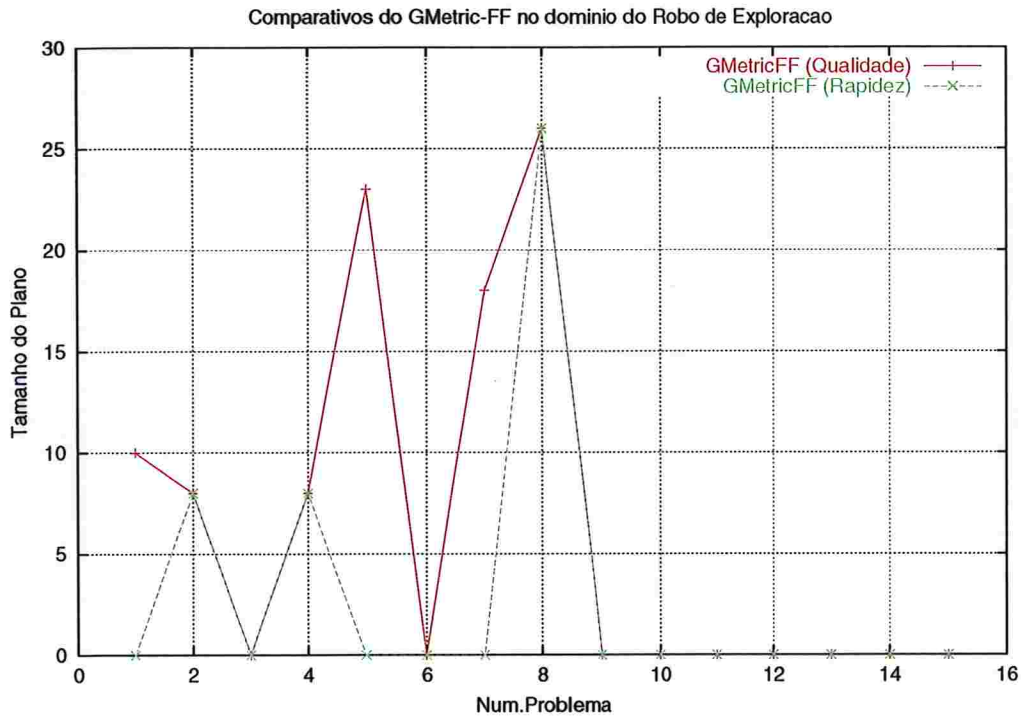

Figura 7.12: Tamanhos dos planos encontrados pelo planejador *GMetric-FF* implementado executado nos modos qualidade e rapidez em problemas no domínio do Robô de Exploração

As Figuras 7.11 e 7.12 apresentam as comparações da execução do planejador *GMetric-FF* nos domínios do Satélite e do Robô de Exploração com variáveis numéricas quando é considerado o critério de otimização (modo qualidade), e quando esse é ignorado (modo rapidez). A partir dessas figuras, pode-se notar que nos dois domínios, o modo qualidade permitiu resolver um maior número de problemas do que o modo rapidez. Isso sugere que o uso de uma heurística que leve em conta o critério de otimização e a expansção de mais nós na busca em largura (modo qualidade), podem fazer com que o planejador resolva mais problemas, evitando assim um beco sem saída, ou um mínimo local.

Além disso, na maioria dos problemas resolvidos nos dois domínios, o tamanho do

plano é maior no modo rapidez. Isso porque, para utilizar o critério de otimização, o planejador deve ser executado no modo qualidade, o que conforme as consideração apresentadas na Seção 5.5, pode levar a soluções menores. No entanto, a Figura 7.11 mostra que o problema 11 foi resolvido apenas no modo rapidez, o que indica que, como o modo qualidade utiliza uma heurística gulosa para escolher entre dois estados de mesmo valor da função heurística, a escolha local pode não significar uma escolha global ótima, e com isso comprometer a resolução do problema.

Numa análise mais específica dos domínios estudados, temos:

1. No domínio do Satélite o critério de otimização é minimizar o combustível gasto pelo conjunto de satélites (variável $fuel\text{–}used$). A escolha entre estados de mesmo valor da função heurística é feita selecionando-se o estado com menor valor para essa variável. A variável $fuel - used$ é incrementada apenas quando a variável $fuel(s)$ é decrementada, e a variável $fuel(s)$ serve para garantir a aplicabilidade das ações de movimentação do satélite s. Dessa forma, fazer a escolha da ação que tem o menor consumo de combustível pode garantir a aplicabilidade das ações de movimentação de forma a evitar becos sem saída, que podem ocorrer quando o combustível de um satélite é totalmente consumido. Os resultados da Figura 7.11 mostram que foi possível resolver mais problemas no domínio do Satélite quando o critério de otimização foi considerado. O problema 11, que foi resolvido apenas quando o critério de otimização não foi considerado, é um caso em que a escolha local não conduziu a busca a uma solução.

2. No domínio do Robô de Exploração o critério de otimização é minimizar o número de recargas feitas nos robôs (variável $recharges$). Apesar dos resultados da Figura 7.12 darem a impressão de que o critério de otimização permitiu ao planejador resolver mais problemas nesse domínio, isso não é o que de fato ocorreu. Uma verificação dos planos encontrados para cada problema permitiu identificar que em nenhum dos problemas resolvidos foi utilizada a ação *recharge(r)*, responsável pela recarga da energia de um robô móvel.

   Uma explicação é que, como o planejador *GMetric-FF* segue por caminhos que melhoram a função heurística, qualquer caminho que resolve um objetivo do problema tem melhor valor de heurística do que o de uma operação de recarga.

Assim, para que seja executada, a ação *recharge(r)* é preciso que o robô esteja em uma localidade sob insolação. Isso significa que, após a execução de uma operação de recarga o valor da função heurística é no mínimo duas unidades maior do que qualquer outro estado que esteja mais próximo do objetivo, e que portanto isso implica em um estado que nunca será selecionado. Além disso, considerar o critério de otimização significa minimizar a quantidade de recargas, ou seja, para os problemas da competição esse foi mais um motivo para o planejador não efetuar a recarga dos robôs. Infelizmente, a não utilização da operação de recarga pode conduzir a busca a um beco sem saída não identificável, razão pela qual o planejador implementado não teve bons resultados nesse domínio. De qualquer maneira, a execução do planejador no modo qualidade permitiu resolver mais problemas, pois expandindo todos os nós da busca em largura fez com que fossem encontrados estados de heurística ainda melhor do que o primeiro encontrado (e escolhido pelo planejador quando executado em modo rapidez), independentemente da aplicação do critério de otimização.

As constatações feitas aqui também possibilitam explicar porquê o planejador *GMetric-FF* resolveu menos problemas no domínio do Robô de Exploração do que no domínio do Satélite. Como não há produção do recurso consumível, para que exista uma solução para um problema no domínio do Satélite, é necessário que os satélites disponíveis tenham combustível suficiente para atingir o objetivo. Neste caso o modo qualidade foi suficiente.

Já para os problemas no domínio do Robô de Exploração, a operação de recarga é a responsável pela produção do recurso consumível. Dessa maneira, uma solução trivial para evitar os becos sem saída neste domínio, é utilizar a operação de recarga mas, como foi discutido, o processo de busca local tende a evitar essa solução. Isso explica porque o modo qualidade não teve o mesmo sucesso que para o domínio do Satélite.

Capítulo 8

Conclusão

Esse trabalho apresentou um estudo sobre problemas e algoritmos de planejamento, em especial as abordagens envolvendo variáveis numéricas. Dentre algumas dessas abordagens, esse trabalho enfocou os planejadores *Graphplan* e *FF*, por serem dois dos mais bem sucedidos planejadores participantes das competições internacionais.

Com a intenção de resolver problemas de planejamento que envolvem variáveis numéricas, foi apresentada uma nova formalização da tarefa de planejamento a fim de incorporar esse novo desafio. Também foi apresentado o planejador *Metric-FF*, uma extensão do planejador *FF* capaz de tratar problemas de planejamento com variáveis numéricas.

Com a intenção de analisar novos domínios da competição *IPC-2002*, foi implementado o planejador *GMetric-FF*, uma pequena variação do planejador *Metric-FF*.

Foi apresentada a teoria de Hoffmann [22], desenvolvida para análises teóricas de domínios de planejamento descritos na linguagem STRIPS. Também foi apresentada uma extensão dessa teoria para analisar domínios que envolvem variáveis numéricas. A teoria de Hoffmann e sua extensão foram utilizadas para analisar os domínios do Mundo dos Blocos, do Satélite e do Robô de Exploração, em suas versões STRIPS e numérica, tendo como resultado final uma taxonomia desses domínios. Essa taxonomia permite efetuar previsões sobre o comportamento dos planejadores *FF* e *GMetric-FF* nos domínios estudados.

As conclusões alcançadas a partir da análise teórica, foram então verificadas empiricamente por meio dos resultados obtidos pelas implementações dos planejadores construídos. A Figura 8.1 sintetiza os resultados encontrados.

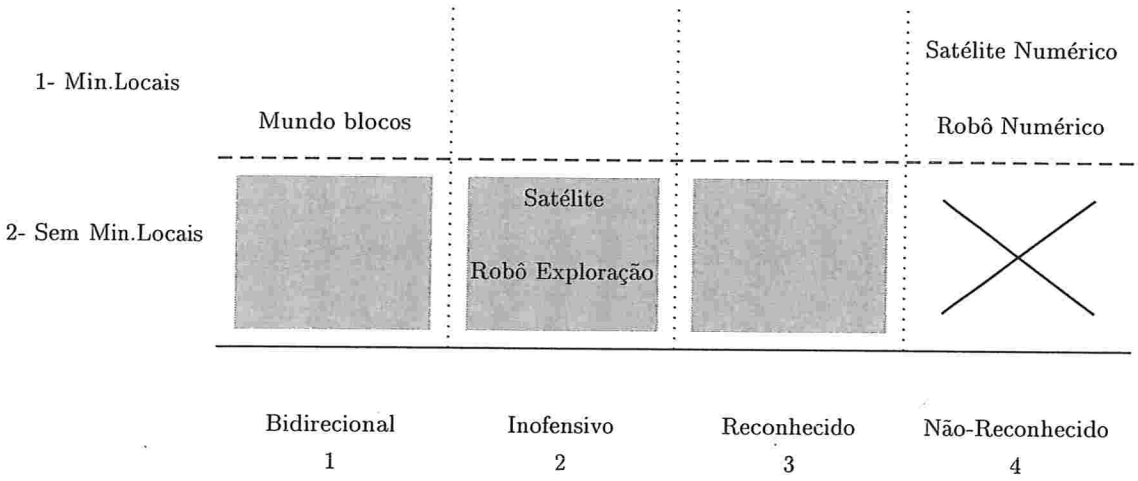

Figura 8.1: Taxonomia dos domínios analisados.

O principal resultado obtido foi o de que os novos desafios propostos pela inclusão de variáveis numéricas tornou os problemas mais próximos dos problemas reais do que os problemas demasiadamente artificiais existentes até então. Ainda são poucos os sistemas capazes de tratar dos aspectos numéricos existentes para essas novas classes de problemas, e com certeza ainda há muito o que ser estudado e aperfeiçoado na área de planejamento em IA a fim de superar esses novos desafios.

8.1 Principais contribuições

Dentre as contribuições apresentadas nesse trabalho podem destacar-se as seguintes:

- Utilização de uma heurística gulosa adicional para aprimorar o desempenho do planejador *Metric-FF* e fazer com que sejam considerados os critérios de otimização dos problemas de planejamento com variáveis numéricas. Tal adição

engendrou o que foi chamado nesse trabalho de planejador *GMetric-FF*.

- Utilização da teoria desenvolvida por Hoffmann para analisar e classificar os domínios do Satélite e do Robô de Exploração. Tais domínios foram propostos posteriormente ao trabalho em que Hoffmann apresentou sua teoria, e portanto, não foram considerados por este.

- Extensão da teoria de Hoffmann para analisar e classificar domínios que envolvem variáveis numéricas.

- Utilização da extensão da teoria de Hoffmann para analisar e classificar os domínios do Satélite e do Robô de Exploração com variáveis numéricas, também não analisados anteriormente.

- Por meio da criação de uma nova heurística, diferente da empregada pelo *Metric-FF*, foi possível justificar porque os problemas no domínio do Robô de Exploração apresentaram maior dificuldade que os problemas no domínio do Satélite;

- Constatação teórica e empírica de que os novos problemas propostos na competição *IPC-2002* que envolvem variáveis numéricas representam um grande desafio aos planejadores.

8.2 Trabalhos futuros

Algumas lacunas foram deixadas por este trabalho, e certamente podem resultar em novos e interessantes trabalhos futuros, entre as quais destacam-se as seguintes:

1. Estender a teoria de Hoffmann de forma a considerar domínios ainda mais complexos, onde o critério de inversibilidade de uma ação não limita-se a uma única ação, dita ação inversa. Qualquer circuito orientado dentro do grafo do espaço de estados de um problema é uma maneira possível de desfazer os efeitos da aplicação de uma ação, e isso também pode caracterizar espaços de estado bidirecionais ou inofensivos.

2. Estender os estudos dos novos desafios dos domínios de planejamento de forma a considerar os problemas que envolvam ações durativas e tempo contínuo, ou pelo

menos, maneiras de reduzir tais problemas a problemas conhecidos, por exemplo, reduzir problemas temporais em problemas com variáveis numéricas discretas ou conínuas.

3. Estudar e propor soluções para melhorar ainda mais o desempenho do planejador *GMetric-FF* para domínios que envolvem recursos e variáveis numéricas, por exemplo, propor novas heurísticas que levem em consideração critérios de otimização;

4. Estudar maneiras de estender o planejador *GMetric-FF* para resolver problemas de planejamento que envolvem tempo, ou seja, cujas ações tem durações diferentes o que exige o raciocínio sobre intervalos de planejamento, por exemplo, a construção de grafos de planejamento temporais.

Referências Bibliográficas

[1] Scott Andrews, Brian Kettler, Kutluhan Erol, and James Hendler. UM translog: A planning domain for the development and benchmarking of planning systems. Technical Report CS-TR-3487, 1995. Disponível em http://www.cs.umd.edu/projects/plus/UMT/umt.ps

[2] A. Barrett and D. S. Weld. Partial-order planning: Evaluating possible efficiency gains. Technical Report TR-92-05-01, 1992.

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

[4] B. Bonet and H. Geffner. Planning as heuristic search, 2001. Disponível em http://www.ldc.usb.ve/ hector/reports/hsp-aij.ps

[5] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1-2):165–204, 1994. Disponível em citeseer.nj.nec.com/bylander94computational.html

[6] T. Dean and S. Kambhampati. Planning and scheduling, 1996. Disponível em http://www.cs.brown.edu/people/tld/postscript/DeanandKambhampatiCRC-96.ps

[7] Silvio do Lago Pereira and Leliane Nunes de Barros. Eficiência num planejador abdutivo. In *Anais do Segundo Congresso de Lógica Aplicada à Tecnologia*, 2001.

[8] Michael Ernst, Todd D. Millstein, and Daniel S. Weld. Automatic SAT-compilation of planning problems. In *IJCAI*, pages 1169–1177, 1997.

[9] Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application for theorem proving to problem solving. In *Advance Papers of the Second International Joint Conference on Artificial Intelligence*, pages 608–620, Edinburgh, Scotland, 1971.

[10] Eugene Fink and Manuela Veloso. Prodigy planning algorithm. Technical Report CMU-CS-93-123, 1994. Disponível em http://www.cs.cmu.edu/afs/cs/user/mmv/www/papers/FinVel-ewsp95.ps.gz

[11] Jeremy Frank, Peter Cheeseman, and John Stutz. When gravity fails: Local search topology. *Journal of Artificial Intelligence Research*, 7:249–281, 1997. Disponível em citeseer.nj.nec.com/frank97when.html

[12] A. Garrido and M. A. Salido F. Barber. Scheduling in a planning environment. Disponível em http://www-is.informatik.uni-oldenburg.de/ sauer/puk2000/papers/Garrido_Scheduling.pdf

[13] Michael P. Georgeff. Planning, 1987.

[14] C. Cordell Green. Application of theorem proving to problem solving. In Donald E. Walker and Lewis M. Morton, editors, *Proceedings of the First International Joint Conference on Artificial Intelligence*, pages 219–239, Bedford, Massachusetts, 1969. International Joint Conference on Artificial Intelligence.

[15] Naresh Gupta and Dana S. Nau. On the complexity of blocks-world planning. *Artificial Intelligence*, 56(2-3):223–254, 1992.

[16] Malte Helmert. Decidability and undecidability results for planning with numerical state variables. In *Sixth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2002)*, 2002.

[17] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search, 2001.

[18] Jörg Hoffmann. FF: The fast-forward planning system. Disponível em http://www.informatik.uni-freiburg.de/ hoffmann/papers/aimag01.ps.gz

[19] Jörg Hoffmann. Local search topology in planning benchmarks: A theoretical analysis. Technical report, 16 2001. Disponível em ftp://ftp.informatik.uni-freiburg.de/documents/reports/report165/report00165.ps.gz

[20] Jörg Hoffmann. Local search topology in planning benchmarks: An empirical analysis. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*, pages 453–458, Seattle, Washington, USA, August 2001.

[21] Jörg Hoffmann. Extending ff to numerical state variables. In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI-02)*, Lyon, France, July 2002. To appear.

[22] Jörg Hoffmann. Local search topology in planning benchmarks: A theoretical analysis. In *Proceedings of the 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS-02)*, Toulouse, France, April 2002. 379-387.

[23] Subbarao Kambhampati. Refinement planning as a unifying framework for plan synthesis. *AI Magazine*, 18(2):67–97, 1997. Disponível em http://rakaposhi.eas.asu.edu/pub/rao/kambhampati-aimag.ps

[24] H. Kautz and B. Selman. Blackbox: A new approach to the application of theorem proving to problem solving, 1998.

[25] Henry Kautz, David McAllester, and Bart Selman. Encoding plans in propositional logic. In Luigia Carlucci Aiello, Jon Doyle, and Stuart Shapiro, editors, *KR'96: Principles of Knowledge Representation and Reasoning*, pages 374–384. Morgan Kaufmann, San Francisco, California, 1996.

[26] Henry Kautz and Bart Selman. Planning as satisfiability. In J. LLoyd, editor, *Proceedings of the Tenth European Conference on Artificial Intelligence*, pages 359–379, 1992.

[27] Henry Kautz and Bart Selman. Unifying SAT-based and graph-based planning. In Jack Minker, editor, *Workshop on Logic-Based Artificial Intelligence, Washington, DC, June 14–16, 1999*, College Park, Maryland, 1999. Computer Science Department, University of Maryland.

[28] Henry A. Kautz and Joachim P. Walser. State-space planning by integer optimization. In *AAAI/IAAI*, pages 526–533, 1999. Disponível em http://www.cs.washington.edu/homes/kautz/papers/ilpplan.ps

[29] J. Koehler, B. Nebel, J. Hoffmann, and Y. Dimopoulos. Extending planning graphs to an ADL subset. Technical Report report00088, 1, 1997. Disponível em http://www-i5.informatik.rwth-aachen.de/LuFG/Lehre/WS97/PLAN/ipTwo.ps.gz

[30] Jana Koehler. Planning under resource constraints. In *European Conference on Artificial Intelligence*, pages 489–493, 1998. Disponível em ftp://ftp.informatik.uni-freiburg.de/documents/papers/ki/koehler-ecai98.ps.gz

[31] Richard E. Korf. Planning as search: A quantitative approach. *Artificial Intelligence*, 33(1):65–88, 1987.

[32] P. Laborie and M. Ghallab. Ixtet: an integrated approach for plan generation and scheduling, 1995. Disponível em ftp://ftp.laas.fr/pub/ria/malik/articles/etfa.pdf

[33] D. Long, M. Fox, L. Sebastia, and A. Coddington. An examination of resources in planning, 2000. Disponível em http://www.dur.ac.uk/ dcs0www/research/stanstuff/Papers/sigres.ps.gz

[34] D. McDermott. PDDL – the planning domain definition language. Technical Report TR-98-003, Yale Center for Computational Vision and Control, OCT 1998. Disponível em http://www.cs.yale.edu/homes/dvm/

[35] M.Fox and D.Long. The aips-2002 international planning competition. In *AIPS*, 2002.

[36] Dana S. Nau, Satyandra K. Gupta, and William C. Regli. AI planning versus manufacturing-operation planning: A case study. pages 1670–1676, 1995. Disponível em http://www.cs.umd.edu/ nau/papers/mfg-ijcai95.ps

[37] E. Pednault. Adl: Exploring the middle ground between strips and the situation calculus, 1989.

[38] J. Scott Penberthy and Daniel S. Weld. UCPOP: A sound, complete, partial order planner for ADL. In Bernhard Nebel, Charles Rich,

and William Swartout, editors, *KR'92. Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*, pages 103–114. Morgan Kaufmann, San Mateo, California, 1992. Disponível em ftp://ftp.cs.pitt.edu/pollack/cs2711/ucpop/doc/ucpop-kr92.ps

[39] J. Scott Penberthy and Daniel S. Weld. Temporal planning with continuous change. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, volume 2, pages 1010–1015, Seattle, Washington, USA, 1994. AAAI Press/MIT Press. Disponível em ftp://ftp.cs.washington.edu/pub/ai/test/zeno-aaai94.ps.Z

[40] Stuart Russel and Peter Norvig. *Artificial Intelligence - A Modern Approach.* Prentice Hall, 1995.

[41] Bart Selman, Henry Kautz, and Bram Cohen. Local search strategies for satisfiability testing. In *AAAI-92: Proceedings 10th National Conference on AI.* American Mathematical Society, 1995.

[42] Bart Selman, Hector J. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In Paul Rosenbloom and Peter Szolovits, editors, *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440–446, Menlo Park, California, 1992. AAAI Press.

[43] Murray Shanahan. *Logical Foundations*, chapter 2, pages 27–56.

[44] Murray Shanahan. *What Is the Frame Problem*, chapter 1, pages 1–26.

[45] Murray Shanahan. An abductive event calculus planner. *Journal of Logic Programming*, 44(1-3):207–240, 2000. Disponível em http://www-ics.ee.ic.ac.uk/ mpsha/planningJLP.ps.Z

[46] D. Smith, J. Frank, and A. Jónsson. Bridging the gap between planning and scheduling, 2000. Disponível em http://ic.arc.nasa.gov/ic/people/de2smith/publications/KER00.pdf

[47] S. Smith. Integrating planning and scheduling: Towards effective coordination in complex, resource-constrained domains, 1993. Disponível em http://www.ri.cmu.edu/pub_files/pub1/smith_stephen_1993_1/smith_stephen_1993_1.ps.gz

[48] T. Vossen, M. Ball, A. Lotem, and D. Nau. Applying integer programming to ai planning, 2000. Disponível em http://www.cs.umd.edu/ nau/papers/knowledge-engr-review-2000.pdf

[49] Joachim P. Walser. Solving Linear Pseudo-Boolean Constraint Problems with Local Search. In *Proceedings of the fourteenth national Conference on Artificial Intelligence (AAAI '97)*, 1997. Disponível em http://www.ps.uni-sb.de/ walser/AAAI97.ps

[50] Daniel S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27–61, 1994. Disponível em ftp://ftp.cs.washington.edu/pub/ai/pi.ps

[51] Daniel S. Weld. Recent advances in AI planning. *AI Magazine*, 20(2):93–123, 1999. Disponível em http://www.salford.ac.uk/planning/papers/pi2.pdf

[52] Steven A. Wolfman and Daniel S. Weld. The LPSAT engine its application to resource planning. In *IJCAI*, pages 310–317, 1999. Disponível em ftp://ftp.cs.washington.edu/pub/ai/ijcai99-lpsat.ps

Apêndice A

O planejador *Satplan*

A.1 Algoritmos de satisfatibilidade

Sem um algoritmo eficiente para encontrar uma satisfação para uma CNF, a tradução de um problema de planejamento em um problema SAT é inútil [51]. O algoritmo *GSat* efetua uma busca gulosa local por uma atribuição de valores verdade às variáveis de um conjunto de cláusulas proposicionais. O algoritmo inicia gerando uma atribuição randômica de valores verdade. Então, o valor verdade de uma variável é invertido (de verdadeiro para falso, e vice-versa) de modo a obter o maior aumento no número total de cláusulas satisfeitas. Essa inversão é repetida até que uma atribuição satisfaça todas as cláusulas, ou que uma quantidade máxima de trocas tenham sido efetuadas. Todo esse processo é repetido até um número máximo de tentativas, quando o algoritmo então encerra com falha. Apesar de correto quando encontra uma atribuição válida, devido a sua natureza randômica *GSat* não é um algoritmo completo, pois pode falhar na busca por uma atribuição válida, mesmo que ela exista.

A seguir, é apresentada a descrição do algoritmo *GSat*:

Algoritmo GSat (α, M_t, M_u)
Entradas:
uma CNF α
O número de máximo de inversões M_i
O número de máximo de tentativas M_t
Saída:
Falha ou uma atribuição para as variáveis de α

---

1. Para i variando de 1 até M_t fazer
2. Seja T uma atribuição randômica para as variáveis de α
3. Para j variando de 1 até M_i fazer
4. Se T satisfaz α, encerrar com T
5. Seja p uma variável de α tal que, a inversão de seu valor verdade atribuído em T leva ao maior incremento no número de cláusulas satisfeitas em α
6. Inverter a atribuição de p em T
7. Fim-fazer
8. Fim-fazer
9. Encerrar com falha.

A.2 O algoritmo *Satplan*

Os experimentos iniciais de Kautz & Selman com o planejador *Satplan* utilizavam um processo manual de codificar um problema como uma CNF [1]. Mais tarde, Ernst, Millstein & Weld [8] descreveram uma maneira automática de gerar toda a codificação de um problema de planejamento em uma CNF a partir de uma descrição do domínio e do problema baseados na linguagem STRIPS. Durante a compilação de um problema de planejamento descrito na linguagem STRIPS, *Satplan* cria uma CNF com a descrição do

[1] O autor, durante uma avaliação do sistema *Satplan* disponibilizado por FTP, verificou ser necessário escrever um pequeno programa para gerar a codificação de cada domínio

estado inicial, a descrição dos efeitos das possíveis ações que podem ser escolhidas em cada instante do plano, e a descrição do estado objetivo.

Como o comprimento do plano solução não é conhecido de antemão, *Satplan* constrói uma fórmula que representa um possível plano com uma comprimento arbitrário, e faz um número determinado de tentativas de satisfazê-la. Caso a tentativa de satisfazer a fórmula não seja bem sucedida, o comprimento do possível plano é então iterativamente incrementado. Desse modo, o processo de geração da CNF e a respectiva tentativa de satisfazê-la é repetido até que uma fórmula satisfatível seja encontrada, ou que o comprimento do possível plano atinja o valor de um comprimento de corte. Se uma fórmula satisfatível for encontrada, a tradução da representação lógica novamente para a representação STRIPS deve ser efetuada, a fim de que o planejador forneça como resposta um plano descrito como uma seqüência de ações.

A figura A.1 apresenta em blocos a arquitetura do planejador *Satplan*. O compilador recebe como entrada as descrições do domínio, do estado inicial, e do estado objetivo, e os compila como uma CNF, utilizando um tamanho arbitrário para o plano solução. Um processo simplificador transforma essa CNF de forma a aumentar o desempenho do algoritmo SAT-Solver, responsável por encontrar um modelo para a CNF compilada e simplificada. Caso o algoritmo SAT-Solver não encontre um modelo para a CNF, o compilador irá compilar novamente o problema, incrementando em um passo o tamanho do plano a ser gerado. Finalmente, o decodificador traduz novamente a representação lógica da CNF na representação da linguagem STRIPS, gerando como saída do processo uma solução na forma de uma seqüência ordenada de ações.

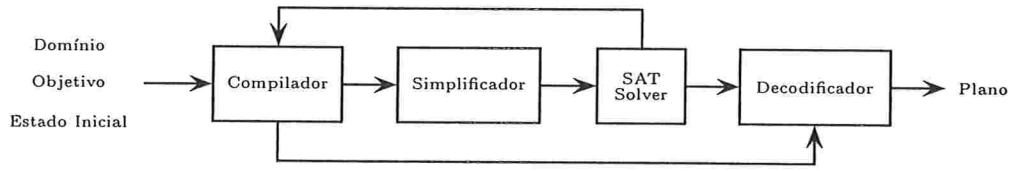

Figura A.1: Arquitetura do planejador *Satplan*

A.3 Compilação do problema em uma CNF

Na abordagem do planejamento como satisfatibilidade o problema de planejamento não é um teorema a ser provado, mas simplesmente um conjunto de axiomas com a propriedade de que qualquer modelo dos axiomas corresponde um plano válido [26]. Dessa forma, conforme já descrito na seção anterior, para que um problema de planejamento seja resolvido como um problema SAT, é necessário codificar o problema como um conjunto de axiomas. Para tal, utiliza-se um modelo padrão de fluentes, onde o tempo assume valores inteiros positivos [51]. Os fluentes de estados ocorrem nos instantes de tempo cuja numeração é par, e os fluentes que denotam a execução de ações ocorrem nos instantes ímpares.

- **Estado Inicial:**

$$\text{Lixo} \wedge \text{MaosLimpas} \wedge \text{Silencio}$$

- **Objetivo:**

$$\text{Jantar} \wedge \text{Presente} \wedge \neg\, \text{Lixo}$$

- **Teoria do Domínio:**

Operador$_1$: Cozinhar
Pré-condição: MaosLimpas
Lista adição: { Jantar }

Operador$_2$: Embrulhar
Pré-condição: Silencio
Lista adição: { Presente }

Operador$_3$: CarregarLixo
Pré-condição: Lixo
Lista remoção: { Lixo, MaosLimpas }

Operador$_4$: TriturarLixo
Pré-condição: Lixo
Lista remoção: { Lixo, Silencio }

Figura A.2: Representação STRIPS do estado inicial e do estado objetivo para o problema do jantar surpresa, além das ações do domínio

Para ilustrar o processo de compilação, será utilizado o domínio do jantar surpresa proposto por Weld [51]. No problema apresentado na figura A.2 juntamente com esse domínio, um marido deve preparar um jantar surpresa para sua esposa, além de embrulhar um presente e se livrar do lixo da cozinha. A esposa está dormindo, e para não estragar a surpresa, ela não deve ser acordada com qualquer forma de barulho até que

as três tarefas tenham sido realizadas.

Há várias maneira de se compilar o problema de planejamento em uma CNF, e o *Satplan* originalmente propôs a codificação que mais tarde foi denominada codificação linear [25]. A codificação linear é muito semelhante a versão proposicional do cálculo de situações, com alguns axiomas adicionais que evitam os chamados "modelos anômalos", que não ocorrem no planejamento dedutivo mas são problemáticos no planejamento como satisfatibilidade. A codificação linear utiliza os seguintes tipos de cláusulas:

1. O estado inicial é completamente especificado.

   $\text{Lixo}(1) \wedge \text{MaosLimpas}(1) \wedge \text{Silencio}(1) \wedge \neg\ \text{Jantar}(1) \wedge \neg\ \text{Presente}(1)$

2. O estado objetivo pode ser totalmente ou parcialmente especificado.

   $\text{Jantar}(2) \wedge \text{Presente}(2) \neg \wedge \text{Lixo}(2)$

3. Se uma ação acontece em um instante de tempo i, suas pré-condições devem ser válidas no instante i, os fluentes de sua lista de adição devem ser verdadeiros no instante $i+1$, e os fluentes de sua lista de remoção devem ser falsos, ou negados no instante $i+1$. Note-se que os efeitos de remoção não são descritos por proposições negadas. Isso porquê, devido ao problema do quadro, não há persistência dos fluentes existentes em um instante i para o instante $i+1$, a menos que isso seja explicitamente representado. Dessa maneira, segundo a descrição acima as proposições que são removidas por uma ação no instante i simplesmente deixam de ser persistidas no instante $i+1$.

   Para remover a possibilidade de uma ação ser executada quando suas pré-condições são falsas, chamado de modelo anômalo, faz se com que a ação implique em suas pré-condições bem como seus efeitos.

   $\forall\ I \cdot \text{Cozinhar}(I) \rightarrow (\ \text{MaosLimpas}(I) \wedge \text{Jantar}(I+1)\ )$
   $\forall\ I \cdot \text{CarregarLixo}(I) \rightarrow \text{Lixo}(I)$

4. O problema do quadro [44] deve ser explicitamente resolvido nessa abordagem. Assim, se uma ação α não afeta um fluente f, então esse fluente f deve ser persistido entre os instantes i e $i+1$ caso a ação α seja executada no instante i.

   $\forall\ I \cdot \text{Cozinhar}(I) \wedge \text{Lixo}(I) \rightarrow \text{Lixo}(I+1)$
   $\forall\ I \cdot \text{Cozinhar}(I) \wedge \neg\ \text{MaosLimpas}(I) \rightarrow \neg\ \text{MaosLimpas}(I+1)$

5. Uma única ação deve ser executada a cada instante.

   $\forall\ I \cdot \text{Cozinhar}(I) \rightarrow \neg\ \text{Embrulhar}(I) \wedge \neg\ \text{CarregarLixo}(I) \wedge \neg\ \text{TriturarLixo}(I)$
   $\forall\ I \cdot \text{Embrulhar}(I) \rightarrow \neg\ \text{Cozinhar}(I) \wedge \neg\ \text{CarregarLixo}(I) \wedge \neg\ \text{TriturarLixo}(I)$

$\forall\ I \cdot \text{CarregarLixo}(I) \rightarrow \neg\ \text{Cozinhar}(I) \wedge \neg\ \text{Embrulhar}(I) \wedge \neg\ \text{TriturarLixo}(I)$

$\forall\ I \cdot \text{TriturarLixo}(I) \rightarrow \neg\ \text{Cozinhar}(I) \wedge \neg\ \text{Embrulhar}(I) \wedge \neg\ \text{CarregarLixo}(I)$

$\forall\ I \cdot \text{Cozinhar}(I) \vee \text{Embrulhar}(I) \vee \text{CarregarLixo}(I) \vee \text{TriturarLixo}(I)$

Apêndice B

Especificação Completa do Graphplan

A especificação a seguir apresenta todos os detalhes do algoritmo de construção do grafo de planejamento a partir de um problema de planejamento STRIPS.

A função apresentada abaixo é a função principal do algoritmo. Essa função recebe como entrada um problema de planejamento STRIPS proposicional e, inicialmente cria um grafo de planejamento a partir das proposições do estado inicial do problema. Iterativamente são adicionados um par de níveis ao grafo, sendo um nível de ações e um de proposições. Para cada um dos dois níveis adicionados, são inseridas em um conjunto, arestas de relações de exclusão mútua entre pares de ações e pares de proposições de um mesmo nível do grafo de planejamento. A função termina quando o último nível do grafo contiver todas as proposições do estado objetivo do problema, e essas não forem mutuamente exclusivas entre sí.

Algoritmo GrafoPlanejamento (Δ, Σ, Γ)
Entradas:
Uma descrição de um domínio Δ
Uma conjunção de proposições do estado inicial Σ
Uma conjunção de proposições do estado objetivo Γ
Saída:
Falha ou um grafo de planejamento $G(V, A)$

---

1. **G** $\longleftarrow$ *criaNivelZeroGrafo* (Σ)
2. Criar um conjunto de arestas de exclusao mútua A_e vazio
3. **i** $\longleftarrow$ 0
4. Se *objetivoSatisfeito* (**G**, **i**, A_e , Γ), então
5. Encerrar retornando o grafo **G**
6. Fim Se
7. **G** $\longleftarrow$ *expandeGrafo* (**G**, **i** , Δ)
8. Se o nível **i** + **1** do grafo não possui vértices, então
9. Encerrar retornando **falha**
10. Fim Se
11. $A_e \longleftarrow$ *relacoesExclusaoAcoes* (G, i, A_e)
12. $A_e \longleftarrow$ *relacoesExclusaoProposicoes* (G, i, A_e)
13. **i** $\longleftarrow$ **i** + 2
14. Voltar para 4

A função apresentada abaixo recebe como entrada as proposições do estado inicial do problema de planejamento STRIPS proposicional, e cria o nível 0 do grafo de planejamento. O nível 0 do grafo possui um vértice para cada proposição do estado inicial do problema.

Algoritmo criaNivelZeroGrafo (Σ)
Entradas:
Uma conjunção de proposições do estado inicial Σ
Saída:
Um grafo G

---

1. Criar um grafo **G(V, A)** vazio
2. Adicionar o nível **0** no grafo **G**
3. Para cada proposição **p** do estado inicial Σ, fazer
4. Adicionar um vértice em **V** com a proposição **p** no nível **0** do grafo **G**
5. Fim Para
6. Encerrar retornando o grafo **G**

A função apresentada abaixo recebe como entrada as proposições do estado objetivo do problema de planejamento STRIPS proposicional, o grafo de planejamento construído até o nível **i**, e o conjunto das arestas de exclusão mútua do grafo. Essa função verifica se todas as proposições do estado objetivo do problema estão presentes no nível **i** do grafo, e se não há arestas de exclusão mútua entre quaisquer duas proposições do objetivo.

Algoritmo objetivoSatisfeito (G, i, A_e, Γ)
Entradas:
Um grafo $G(V, A)$
O último nível do grafo i
Um conjunto de arestas com relações de exclusão mútua A_e
Uma conjunção de proposições do estado objetivo Γ
Saída:
TRUE ou FALSE

---

1. Para cada proposição **p** do estado objetivo Γ, fazer
2. Se **p** não pertence ao nível **i** do grafo **G**, então
3. Encerrar retornando FALSE
4. Fim Se
5. Fim Para
6. Para cada proposição **p** do estado objetivo Γ, fazer
7. Para cada proposição **q** do estado objetivo Γ, fazer
8. Se **p** $\neq$ **q**, sendo que **p** e **q** pertencem ao nível **i** do conjunto A_e, então
9. Encerrar retornando FALSE
10. Fim Se
11. Fim Para
12. Fim Para
13. Encerrar retornando TRUE

A função apresentada abaixo recebe como entrada uma descrição do domínio do problema de planejamento STRIPS proposicional, e o grafo de planejamento construído até o nível **i**. Essa função adiciona ao grafo de planejamento vértices de ações NO-OP que persistem as proposições existentes no nível **i** do grafo, além de vértices com ações aplicáveis no nível **i** + **1** do grafo. A função também adiciona ao grafo de planejamento vértices com as proposições resultantes das ações, bem como todas as arestas necessárias ao grafo, ou seja, arestas de pré-condição e arestas de efeito.

Algoritmo expandeGrafo (G, i, Δ)
Entradas:
Um grafo $G(V, A)$
O último nível do grafo i
Uma descrição de um domínio Δ
Saída:
Um grafo G

---

1. Adicionar os níveis **i** + **1** e **i** + **2** do grafo **G**
2. Para cada proposição **p** do nível **i** do grafo, fazer
3. Adicionar um vértice com uma ação **NO-OP(p)** no nível **i** + **1** do grafo
4. Adicionar um vértice com a proposição **p** no nível **i** + **2** do grafo
5. Adicionar uma aresta conectando o vértice da proposição **p** no nível **i** do grafo com o vértice da ação **NO-OP(p)** no nível **i** + **1** do grafo
6. Adicionar uma aresta conectando o vértice da ação **NO-OP(p)** no nível **i** + **1** do grafo com o vértice da proposição **p** no nível **i** + **2** do grafo
7. Fim Para
8. Para cada ação **a** do domínio Δ, fazer
9. **aplicável** $\longleftarrow$ TRUE
10. Para cada proposição **p** da pré-condição da ação **a**, fazer
11. Se **p** não pertence ao nível **i** do grafo, então
12. **aplicável** $\longleftarrow$ FALSE
13. Fim Se
14. Fim Para
15. Se **aplicável** = TRUE, então
16. Adicionar um vértice com a ação **a** no nível **i** + **1** do grafo

17. Para cada proposição **p** da pré-condição da ação **a**, fazer

18. Adicionar uma aresta conectando o vértice da proposição **p** no nível **i** do grafo com o vértice da ação **a** no nível **i + 1** do grafo

19. Fim Para

20. Para cada proposição **p** dos efeitos da ação **a**, fazer

21. Adicionar um vértice com a proposição **p** no nível **i + 2** do grafo

22. Adicionar uma aresta conectando o vértice da ação **a** no nível **i + 1** do grafo com o vértice da proposição **p** no nível **i + 2** do grafo

23. Fim Para

24. Fim Se

25. Fim Para

26. Encerrar retornando o grafo **G**

A função apresentada abaixo recebe como entrada o grafo de planejamento construído até o nível $\mathbf{i} + \mathbf{2}$, e o conjunto das arestas de exclusão mútua do grafo. Essa função adiciona ao conjunto de arestas de exclusão mútua qualquer par de ações do nível $\mathbf{i} + \mathbf{1}$ do grafo que não puderem ser executadas simultaneamente por um dos seguintes motivos: efeitos inconsistentes, interferência, ou necessidades conflitantes.

Algoritmo relacoesExclusaoAcoes (G, i, A_e)
Entradas:
Um grafo $G(V, A)$
O antepenúltimo nível do grafo i
Um conjunto de arestas com relações de exclusão mútua A_e
Saída:
Um conjunto de arestas A_e

```
Para cada ação a do nível i + 1 do grafo, fazer
    Para cada ação b do nível i + 1 do grafo, fazer
        Se a ≠ b, então
            Para cada proposição p dos efeitos da ação a, fazer
                Se ¬ p é uma proposição dos efeitos da ação b, então
                    Adicionar as ações a e b em A_e no nível i + 1
                Fim Se
                Se ¬ p é uma proposição das pré-condições da ação b, então
                    Adicionar as ações a e b em A_e no nível i + 1
                Fim Se
            Fim Para
            Para cada proposição p das pré-condições da ação a, fazer
                Para cada proposição q das pré-condições da ação b, fazer
                    Se p ≠ q, sendo que p e q pertencem ao nível i do conjunto A_e, então
                        Adicionar as ações a e b em A_e no nível i + 1
                    Fim Se
                Fim Para
            Fim Para
```

19. Fim Se

20. Fim Para

21. Fim Para

22. Encerrar retornando A_e

A função apresentada abaixo recebe como entrada o grafo de planejamento construído até o nível **i** + **2**, e o conjunto das arestas de exclusão mútua do grafo. Essa função adiciona ao conjunto de arestas de exclusão mútua qualquer par de proposições do nível **i** + **2** do grafo que não puderem ser verdadeiras simultaneamente porque qualquer par de ações que as adicionam são mutuamente exclusivas.

Algoritmo relacoesExclusaoProposicoes (G, i, A_e)
Entradas:
Um grafo $G(V, A)$
O antepenúltimo nível do grafo i
Um conjunto de arestas com relações de exclusão mútua A_e
Saída:
Um conjunto de arestas A_e

```
Para cada proposição p do nível i + 2 do grafo, fazer
    Para cada proposição q do nível i + 2 do grafo, fazer
        Se p ≠ q, então
            exclusivas ⟵ TRUE
            Para cada ação a do nível i + 1 do grafo que adiciona p, fazer
                Para cada ação b do nível i + 1 do grafo que adiciona q, fazer
                    Se a e b não pertencem ao nível i + 1 do conjunto A_e, então
                        exclusivas ⟵ FALSE
                    Fim Se
                Fim Para
            Fim Para
            Se exclusivas = TRUE, então
                Adicionar as proposições p e q em A_e no nível i + 2
            Fim Se
        Fim Se
    Fim Para
Fim Para
Encerrar retornando A_e
```

Apêndice C

Domínios da competição AIPS-2002

C.1 O domínio do satélite STRIPS

```
( define ( domain satellite )
    ( :requirements :strips :equality :typing )
    ( :types satellite direction instrument mode )
    ( :predicates
        ( on_board ?i - instrument ?s - satellite )
        ( supports ?i - instrument ?m - mode )
        ( pointing ?s - satellite ?d - direction )
        ( power_avail ?s - satellite )
        ( power_on ?i - instrument )
        ( calibrated ?i - instrument )
        ( have_image ?d - direction ?m - mode )
        ( calibration_target ?i - instrument ?d - direction )
    )
    ( :action turn_to
        :parameters ( ?s - satellite ?d_new - direction ?d_prev - direction )
        :precondition ( and
            ( pointing ?s ?d_prev )
```

```
            ( not ( = ?d_new ?d_prev ) )
        )
        :effect ( and
            ( pointing ?s ?d_new )
            ( not ( pointing ?s ?d_prev ) )
        )
    )
    ( :action switch_on
        :parameters ( ?i - instrument ?s - satellite )
        :precondition ( and
            ( on_board ?i ?s )
            ( power_avail ?s )
        )
        :effect ( and
            ( power_on ?i )
            ( not ( calibrated ?i ) )
            ( not ( power_avail ?s ) )
        )
    )
    ( :action switch_off
        :parameters ( ?i - instrument ?s - satellite )
        :precondition ( and
            ( on_board ?i ?s )
            ( power_on ?i )
        )
        :effect ( and
            ( not ( power_on ?i ) )
            ( power_avail ?s )
        )
    )
    ( :action calibrate
        :parameters ( ?s - satellite ?i - instrument ?d - direction )
        :precondition ( and
```

```
            ( on_board ?i ?s )
            ( calibration_target ?i ?d )
            ( pointing ?s ?d )
            ( power_on ?i )
        )
        :effect ( calibrated ?i )
    )
    ( :action take_image
        :parameters ( ?s - satellite ?d - direction ?i - instrument ?m - mode )
        :precondition ( and
            ( calibrated ?i )
            ( on_board ?i ?s )
            ( supports ?i ?m )
            ( power_on ?i )
            ( pointing ?s ?d )
        )
        :effect ( have_image ?d ?m )
    )
)
```

C.2 Exemplo de problema para a versão STRIPS

O problema para a versão STRIPS do domínio do satélite descrito a seguir possui um único satélite *satellite0* que carrega a bordo um único instrumento *instrument1*, capaz de suportar os modos *image2* e *infrared0*.

Inicialmente o satélite está apontando para *planet4* e deve ser apontado para *groundstation2* a fim de ser calibrado para uso. O objetivo do problema é obter imagens nos modos *infrared0* dos alvos *star7*, *planet4*, *planet3*, e *phenomenon6*, além de uma imagem no modo *image2* do alvo *phenomenon5*.

```
( define ( problem satellite-pfile2 )
    ( :domain satellite )
```

```
    ( :objects
        satellite0 - satellite
        groundstation2 planet3 planet4 phenomenon5 phenomenon6 star7 - direction
        instrument1 - instrument
        infrared0 image2 - mode
    )
    ( :init ( satellite satellite0 )
        ( direction groundStation2 )
        ( direction planet3 )
        ( direction planet4 )
        ( direction phenomenon5 )
        ( direction phenomenon6 )
        ( direction star7 )
        ( instrument instrument1 )
        ( mode infrared0 )
        ( mode image2 )
        ( on_board instrument1 satellite0 )
        ( supports instrument1 image2 )
        ( supports instrument1 infrared0 )
        ( pointing satellite0 planet4 )
        ( power_avail satellite0 )
        ( calibration_target instrument1 groundstation2 )
    )
    ( :goal ( and
        ( have_image star7 infrared0 )
        ( have_image planet4 infrared0 )
        ( have_image planet3 infrared0 )
        ( have_image phenomenon6 infrared0 )
        ( have_image phenomenon5 image2 )
        )
    )
)
```

C.3 O domínio do satélite com variáveis numéricas

```
( define ( domain satellite )
    ( :requirements :typing :fluents :equality )
    ( :types satellite direction instrument mode )
    ( :predicates
        ( on_board ?i - instrument ?s - satellite )
        ( supports ?i - instrument ?m - mode )
        ( pointing ?s - satellite ?d - direction )
        ( power_avail ?s - satellite )
        ( power_on ?i - instrument )
        ( calibrated ?i - instrument )
        ( have_image ?d - direction ?m - mode )
        ( calibration_target ?i - instrument ?d - direction )
    )
    ( :functions
        ( data_capacity ?s - satellite )
        ( data ?d - direction ?m - mode )
        ( slew_time ?a ?b - direction )
        ( data-stored )
        ( fuel ?s - satellite )
        ( fuel-used )
    )
    ( :action turn_to
        :parameters ( ?s - satellite ?d_new - direction ?d_prev - direction )
        :precondition ( and
            ( pointing ?s ?d_prev )
            ( not ( = ?d_new ?d_prev ) )
            ( >= ( fuel ?s ) ( slew_time ?d_new ?d_prev ) )
        )
        :effect ( and
            ( pointing ?s ?d_new )
            ( not ( pointing ?s ?d_prev ) )
```

```
            ( decrease ( fuel ?s ) ( slew_time ?d_new ?d_prev ) )
            ( increase ( fuel-used ) ( slew_time ?d_new ?d_prev ) )
        )
    )
    ( :action switch_on
        :parameters ( ?i - instrument ?s - satellite )
        :precondition ( and
            ( on_board ?i ?s )
            ( power_avail ?s )
        )
        :effect ( and
            ( power_on ?i )
            ( not ( calibrated ?i ) )
            ( not ( power_avail ?s ) )
        )
    )
    ( :action switch_off
        :parameters ( ?i - instrument ?s - satellite )
        :precondition ( and
            ( on_board ?i ?s )
            ( power_on ?i )
        )
        :effect ( and
            ( not ( power_on ?i ) )
            ( power_avail ?s )
        )
    )
    ( :action calibrate
        :parameters ( ?s - satellite ?i - instrument ?d - direction )
        :precondition ( and
            ( on_board ?i ?s )
            ( calibration_target ?i ?d )
            ( pointing ?s ?d )
```

```
            ( power_on ?i )
        )
        :effect ( calibrated ?i )
    )
    ( :action take_image
        :parameters ( ?s - satellite ?d - direction ?i - instrument ?m - mode )
        :precondition ( and
            ( calibrated ?i )
            ( on_board ?i ?s )
            ( supports ?i ?m )
            ( power_on ?i )
            ( pointing ?s ?d )
            ( >= ( data_capacity ?s ) ( data ?d ?m ) )
        )
        :effect ( and
            ( decrease ( data_capacity ?s ) ( data ?d ?m ) )
            ( have_image ?d ?m )
            ( increase ( data-stored ) ( data ?d ?m ) )
        )
    )
)
```

C.4 Exemplo de problema para a versão númerica

O problema para a versão númerica do domínio do satélite descrito a seguir é uma simplificação do primeiro problema nesse domínio da competição *IPC-2002*. O problema possui um único satélite *satellite0* que carrega a bordo um único instrumento *instrument0*, capaz de suportar apenas o modo *thermograph0*. De acordo com a função *data_capacity*, a capacidade inicial de armazenamento de dados do satélite é 1000 unidades de memória e, segundo a função *fuel*, a capacidade inicial de combustível é de 118 unidades. Algumas funções *data* especificam as capacidades de armazenamento de dados necessárias para a obtenção de imagens de alvos diversos em modos diversos e, funções *slew_time* especificam as quantidades de combustível necessárias para a

movimentação do satélite entre possíveis combinações de alvo origem e alvo destino.

Inicialmente o satélite está apontando para *Phenomenon6* e deve ser apontado para *GroundStation2* a fim de ser calibrado para uso. O objetivo do problema é obter imagens no modo *thermograph0* dos alvos *Phenomenon4*, *Star5*, e *Phenomenon6* de modo a minimizar a quantidade de combustível utilizada.

```
( define ( problem numeric-sat-x-1 )
    ( :domain satellite )
    ( :objects
        satellite0 - satellite
        instrument0 - instrument
        thermograph0 - mode
        image1 - mode
        spectrograph2 - mode
        Star0 - direction
        GroundStation1 - direction
        GroundStation2 - direction
        Phenomenon3 - direction
        Phenomenon4 - direction
        Star5 - direction
        Phenomenon6 - direction
    )
    ( :init ( supports instrument0 thermograph0 )
        ( calibration_target instrument0 GroundStation2 )
        ( on_board instrument0 satellite0 )
        ( power_avail satellite0 )
        ( pointing satellite0 Phenomenon6 )
        ( = ( data_capacity satellite0 ) 1000 )
        ( = ( fuel satellite0 ) 118 )
        ( = ( data Phenomenon3 image1 ) 22 )
        ( = ( data Phenomenon4 image1 ) 120 )
        ( = ( data Star5 image1 ) 203 )
        ( = ( data Phenomenon6 image1 ) 144 )
```

```
( = ( data Phenomenon3 spectrograph2 ) 125 )
( = ( data Phenomenon4 spectrograph2 ) 196 )
( = ( data Star5 spectrograph2 ) 68 )
( = ( data Phenomenon6 spectrograph2 ) 174 )
( = ( data Phenomenon3 thermograph0 ) 136 )
( = ( data Phenomenon4 thermograph0 ) 134 )
( = ( data Star5 thermograph0 ) 273 )
( = ( data Phenomenon6 thermograph0 ) 219 )
( = ( slew_time GroundStation1 Star0 ) 18.17 )
( = ( slew_time Star0 GroundStation1 ) 18.17 )
( = ( slew_time GroundStation2 Star0 ) 38.61 )
( = ( slew_time Star0 GroundStation2 ) 38.61 )
( = ( slew_time GroundStation2 GroundStation1 ) 68.04 )
( = ( slew_time GroundStation1 GroundStation2 ) 68.04 )
( = ( slew_time Phenomenon3 Star0 ) 14.29 )
( = ( slew_time Star0 Phenomenon3 ) 14.29 )
( = ( slew_time Phenomenon3 GroundStation1 ) 89.48 )
( = ( slew_time GroundStation1 Phenomenon3 ) 89.48 )
( = ( slew_time Phenomenon3 GroundStation2 ) 33.94 )
( = ( slew_time GroundStation2 Phenomenon3 ) 33.94 )
( = ( slew_time Phenomenon4 Star0 ) 35.01 )
( = ( slew_time Star0 Phenomenon4 ) 35.01 )
( = ( slew_time Phenomenon4 GroundStation1 ) 31.79 )
( = ( slew_time GroundStation1 Phenomenon4 ) 31.79 )
( = ( slew_time Phenomenon4 GroundStation2 ) 39.73 )
( = ( slew_time GroundStation2 Phenomenon4 ) 39.73 )
( = ( slew_time Phenomenon4 Phenomenon3 ) 25.72 )
( = ( slew_time Phenomenon3 Phenomenon4 ) 25.72 )
( = ( slew_time Star5 Star0 ) 36.56 )
( = ( slew_time Star0 Star5 ) 36.56 )
( = ( slew_time Star5 GroundStation1 ) 8.59 )
( = ( slew_time GroundStation1 Star5 ) 8.59 )
( = ( slew_time Star5 GroundStation2 ) 62.86 )
```

```
        ( = ( slew_time GroundStation2 Star5 ) 62.86 )
        ( = ( slew_time Star5 Phenomenon3 ) 10.18 )
        ( = ( slew_time Phenomenon3 Star5 ) 10.18 )
        ( = ( slew_time Star5 Phenomenon4 ) 64.5 )
        ( = ( slew_time Phenomenon4 Star5 ) 64.5 )
        ( = ( slew_time Phenomenon6 Star0 ) 77.07 )
        ( = ( slew_time Star0 Phenomenon6 ) 77.07 )
        ( = ( slew_time Phenomenon6 GroundStation1 ) 17.63 )
        ( = ( slew_time GroundStation1 Phenomenon6 ) 17.63 )
        ( = ( slew_time Phenomenon6 GroundStation2 ) 50.73 )
        ( = ( slew_time GroundStation2 Phenomenon6 ) 50.73 )
        ( = ( slew_time Phenomenon6 Phenomenon3 ) 14.75 )
        ( = ( slew_time Phenomenon3 Phenomenon6 ) 14.75 )
        ( = ( slew_time Phenomenon6 Phenomenon4 ) 2.098 )
        ( = ( slew_time Phenomenon4 Phenomenon6 ) 2.098 )
        ( = ( slew_time Phenomenon6 Star5 ) 29.32 )
        ( = ( slew_time Star5 Phenomenon6 ) 29.32 )
        ( = ( data-stored ) 0 )
        ( = ( fuel-used ) 0 )
    )
    ( :goal ( and
        ( have_image Phenomenon4 thermograph0 )
        ( have_image Star5 thermograph0 )
        ( have_image Phenomenon6 thermograph0 ) )
    )
    ( :metric minimize ( fuel-used ) )
)
```

C.5 O domínio do robô de exploração STRIPS

```
( define ( domain Rover )
    ( :requirements :typing )
    ( :types rover waypoint store camera mode lander objective )
```

```
( :predicates
    ( at ?x - rover ?y - waypoint )
    ( at_lander ?x - lander ?y - waypoint )
    ( can_traverse ?r - rover ?x - waypoint ?y - waypoint )
    ( equipped_for_soil_analysis ?r - rover )
    ( equipped_for_rock_analysis ?r - rover )
    ( equipped_for_imaging ?r - rover )
    ( empty ?s - store )
    ( have_rock_analysis ?r - rover ?w - waypoint )
    ( have_soil_analysis ?r - rover ?w - waypoint )
    ( full ?s - store )
    ( calibrated ?c - camera ?r - rover )
    ( supports ?c - camera ?m - mode )
    ( visible ?w - waypoint ?p - waypoint )
    ( have_image ?r - rover ?o - objective ?m - mode )
    ( communicated_soil_data ?w - waypoint )
    ( communicated_rock_data ?w - waypoint )
    ( communicated_image_data ?o - objective ?m - mode )
    ( at_soil_sample ?w - waypoint )
    ( at_rock_sample ?w - waypoint )
    ( visible_from ?o - objective ?w - waypoint )
    ( store_of ?s - store ?r - rover )
    ( calibration_target ?i - camera ?o - objective )
    ( on_board ?i - camera ?r - rover )
)
( :action navigate
    :parameters ( ?x - rover ?y - waypoint ?z - waypoint )
    :precondition ( and
        ( can_traverse ?x ?y ?z )
        ( at ?x ?y )
        ( visible ?y ?z )
    )
    :effect ( and
```

```
            ( not ( at ?x ?y ) )
            ( at ?x ?z )
        )
    )
    ( :action sample_soil
        :parameters ( ?x - rover ?s - store ?p - waypoint )
        :precondition ( and
            ( at ?x ?p )
            ( at_soil_sample ?p )
            ( equipped_for_soil_analysis ?x )
            ( store_of ?s ?x )
            ( empty ?s )
        )
        :effect ( and
            ( not ( empty ?s ) )
            ( full ?s )
            ( have_soil_analysis ?x ?p )
            ( not ( at_soil_sample ?p ) )
        )
    )
    ( :action sample_rock
        :parameters ( ?x - rover ?s - store ?p - waypoint )
        :precondition ( and
            ( at ?x ?p )
            ( at_rock_sample ?p )
            ( equipped_for_rock_analysis ?x )
            ( store_of ?s ?x )
            ( empty ?s )
        )
        :effect ( and
            ( not ( empty ?s ) )
            ( full ?s )
            ( have_rock_analysis ?x ?p )
```

```
            ( not ( at_rock_sample ?p ) )
        )
    )
    ( :action drop
        :parameters ( ?x - rover ?y - store )
        :precondition ( and
            ( store_of ?y ?x )
            ( full ?y )
        )
        :effect ( and
            ( not ( full ?y ) )
            ( empty ?y )
        )
    )
    ( :action calibrate
        :parameters ( ?r - rover ?i - camera ?t - objective ?w - waypoint )
        :precondition ( and
            ( equipped_for_imaging ?r )
            ( calibration_target ?i ?t )
            ( at ?r ?w )
            ( visible_from ?t ?w )
            ( on_board ?i ?r )
        )
        :effect ( calibrated ?i ?r )
    )
    ( :action take_image
        :parameters ( ?r - rover ?p - waypoint ?o - objective ?i - camera
            ?m - mode )
        :precondition ( and
            ( calibrated ?i ?r )
            ( on_board ?i ?r )
            ( equipped_for_imaging ?r )
            ( supports ?i ?m )
```

```
            ( visible_from ?o ?p )
            ( at ?r ?p )
        )
        :effect ( and
            ( have_image ?r ?o ?m )
            ( not ( calibrated ?i ?r ) )
        )
    )
    ( :action communicate_soil_data
        :parameters ( ?r - rover ?l - lander ?p - waypoint ?x - waypoint
            ?y - waypoint )
        :precondition ( and
            ( at ?r ?x )
            ( at_lander ?l ?y )
            ( have_soil_analysis ?r ?p )
            ( visible ?x ?y )
        )
        :effect ( and
            ( communicated_soil_data ?p )
        )
    )
    ( :action communicate_rock_data
        :parameters ( ?r - rover ?l - lander ?p - waypoint ?x - waypoint
            ?y - waypoint )
        :precondition ( and
            ( at ?r ?x )
            ( at_lander ?l ?y )
            ( have_rock_analysis ?r ?p )
            ( visible ?x ?y )
        )
        :effect ( and
            ( communicated_rock_data ?p )
        )
```

```
    )
    ( :action communicate_image_data
        :parameters ( ?r - rover ?l - lander ?o - objective ?m - mode
            ?x - waypoint ?y - waypoint )
        :precondition ( and
            ( at ?r ?x )
            ( at_lander ?l ?y )
            ( have_image ?r ?o ?m )
            ( visible ?x ?y )
        )
        :effect ( and
            ( communicated_image_data ?o ?m )
        )
    )
)
```

C.6 Exemplo de problema para a versão STRIPS

O problema exemplo para a versão STRIPS do domínios do robô de exploração descrito a seguir possui as seguintes características: um único robô *rover0* equipado para análises de solo e de rocha, além de estar equipado para a obtenção de imagens, e uma única estação-base *general*. O robô *rover0* carrega a bordo uma câmera *camera0* com suporte aos modos *colour* e *high_res*.

Inicialmente o robô está posicionado na localidade *waypoint3*, seu compartimento está vazio e sua câmera está descalibrada. O objetivo do problema é obter uma amostra de solo da localidade *waypoint2*, uma amostra de rocha da localidade *waypoint3*, e uma imagem no modo *high_res* do alvo *objective1*, devendo enviar os dados das mesmas à estação-base *general* situada na localidade *waypoint0*.

```
( define ( problem roverprob1234 )
    ( :domain Rover )
    ( :objects
```

```
        general - Lander
        colour - Mode
        high_res - Mode
        low_res - Mode
        rover0 - Rover
        rover0store - Store
        waypoint0 - Waypoint
        waypoint1 - Waypoint
        waypoint2 - Waypoint
        waypoint3 - Waypoint
        camera0 - Camera
        objective0 - Objective
        objective1 - Objective
    )
    ( :init
        ( visible waypoint1 waypoint0 )
        ( visible waypoint0 waypoint1 )
        ( visible waypoint2 waypoint0 )
        ( visible waypoint0 waypoint2 )
        ( visible waypoint2 waypoint1 )
        ( visible waypoint1 waypoint2 )
        ( visible waypoint3 waypoint0 )
        ( visible waypoint0 waypoint3 )
        ( visible waypoint3 waypoint1 )
        ( visible waypoint1 waypoint3 )
        ( visible waypoint3 waypoint2 )
        ( visible waypoint2 waypoint3 )
        ( at_soil_sample waypoint0 )
        ( at_rock_sample waypoint1 )
        ( at_soil_sample waypoint2 )
        ( at_rock_sample waypoint2 )
        ( at_soil_sample waypoint3 )
        ( at_rock_sample waypoint3 )
```

```
        ( at_lander general waypoint0 )
        ( at rover0 waypoint3 )
        ( store_of rover0store rover0 )
        ( empty rover0store )
        ( equipped_for_soil_analysis rover0 )
        ( equipped_for_rock_analysis rover0 )
        ( equipped_for_imaging rover0 )
        ( can_traverse rover0 waypoint3 waypoint0 )
        ( can_traverse rover0 waypoint0 waypoint3 )
        ( can_traverse rover0 waypoint3 waypoint1 )
        ( can_traverse rover0 waypoint1 waypoint3 )
        ( can_traverse rover0 waypoint1 waypoint2 )
        ( can_traverse rover0 waypoint2 waypoint1 )
        ( on_board camera0 rover0 )
        ( calibration_target camera0 objective1 )
        ( supports camera0 colour )
        ( supports camera0 high_res )
        ( visible_from objective0 waypoint0 )
        ( visible_from objective0 waypoint1 )
        ( visible_from objective0 waypoint2 )
        ( visible_from objective0 waypoint3 )
        ( visible_from objective1 waypoint0 )
        ( visible_from objective1 waypoint1 )
        ( visible_from objective1 waypoint2 )
        ( visible_from objective1 waypoint3 )
    )
    ( :goal ( and
        ( communicated_soil_data waypoint2 )
        ( communicated_rock_data waypoint3 )
        ( communicated_image_data objective1 high_res ) )
    )
)
```

C.7 O domínio do robô de exploração com variáveis numéricas

```
( define ( domain Rover )
    ( :requirements :typing :fluents )
    ( :types rover waypoint store camera mode lander objective )
    ( :predicates
        ( at ?x - rover ?y - waypoint )
        ( at_lander ?x - lander ?y - waypoint )
        ( can_traverse ?r - rover ?x - waypoint ?y - waypoint )
        ( equipped_for_soil_analysis ?r - rover )
        ( equipped_for_rock_analysis ?r - rover )
        ( equipped_for_imaging ?r - rover )
        ( empty ?s - store )
        ( have_rock_analysis ?r - rover ?w - waypoint )
        ( have_soil_analysis ?r - rover ?w - waypoint )
        ( full ?s - store )
        ( calibrated ?c - camera ?r - rover )
        ( supports ?c - camera ?m - mode )
        ( visible ?w - waypoint ?p - waypoint )
        ( have_image ?r - rover ?o - objective ?m - mode )
        ( communicated_soil_data ?w - waypoint )
        ( communicated_rock_data ?w - waypoint )
        ( communicated_image_data ?o - objective ?m - mode )
        ( at_soil_sample ?w - waypoint )
        ( at_rock_sample ?w - waypoint )
        ( visible_from ?o - objective ?w - waypoint )
        ( store_of ?s - store ?r - rover )
        ( calibration_target ?i - camera ?o - objective )
        ( on_board ?i - camera ?r - rover )
        ( in_sun ?w - waypoint )
    )
    ( :functions
        ( energy ?r - rover )
```

```
        ( recharges )
    )
    ( :action navigate
        :parameters ( ?x - rover ?y - waypoint ?z - waypoint )
        :precondition ( and
            ( can_traverse ?x ?y ?z )
            ( at ?x ?y )
            ( visible ?y ?z )
            ( >= ( energy ?x ) 8 )
        )
        :effect ( and
            ( decrease ( energy ?x ) 8 )
            ( not ( at ?x ?y ) )
            ( at ?x ?z )
        )
    )
    ( :action recharge
        :parameters ( ?x - rover ?w - waypoint )
        :precondition ( and
            ( at ?x ?w )
            ( in_sun ?w )
            ( <= ( energy ?x ) 80 )
        )
        :effect ( and
            ( increase ( energy ?x ) 20 )
            ( increase ( recharges ) 1 )
        )
    )
    ( :action sample_soil
        :parameters ( ?x - rover ?s - store ?p - waypoint )
        :precondition ( and
            ( at ?x ?p )
            ( >= ( energy ?x ) 3 )
```

```
            ( at_soil_sample ?p )
            ( equipped_for_soil_analysis ?x )
            ( store_of ?s ?x )
            ( empty ?s )
        )
        :effect ( and
            ( not ( empty ?s ) )
            ( full ?s)
            ( decrease ( energy ?x ) 3 )
            ( have_soil_analysis ?x ?p )
            ( not ( at_soil_sample ?p ) )
        )
    )
    ( :action sample_rock
        :parameters ( ?x - rover ?s - store ?p - waypoint )
        :precondition ( and
            ( at ?x ?p )
            ( >= ( energy ?x ) 5 )
            ( at_rock_sample ?p )
            ( equipped_for_rock_analysis ?x )
            ( store_of ?s ?x )
            ( empty ?s )
        )
        :effect ( and
            ( not ( empty ?s ) )
            ( full ?s )
            ( decrease ( energy ?x ) 5 )
            ( have_rock_analysis ?x ?p )
            ( not ( at_rock_sample ?p ) )
        )
    )
    ( :action drop
        :parameters ( ?x - rover ?y - store )
```

```
        :precondition ( and
            ( store_of ?y ?x )
            ( full ?y )
        )
        :effect ( and
            ( not ( full ?y ) )
            ( empty ?y )
        )
    )
    ( :action calibrate
        :parameters ( ?r - rover ?i - camera ?t - objective ?w - waypoint )
        :precondition ( and
            ( equipped_for_imaging ?r )
            ( >= ( energy ?r ) 2 )
            ( calibration_target ?i ?t )
            ( at ?r ?w )
            ( visible_from ?t ?w )
            ( on_board ?i ?r )
        )
        :effect ( and
            ( decrease ( energy ?r ) 2 )
            ( calibrated ?i ?r )
        )
    )
    ( :action take_image
        :parameters ( ?r - rover ?p - waypoint ?o - objective ?i - camera
            ?m - mode )
        :precondition ( and
            ( calibrated ?i ?r )
            ( on_board ?i ?r )
            ( equipped_for_imaging ?r )
            ( supports ?i ?m )
            ( visible_from ?o ?p )
```

```
            ( at ?r ?p )
            ( >= ( energy ?r ) 1 )
        )
        :effect ( and
            ( have_image ?r ?o ?m )
            ( not ( calibrated ?i ?r ) )
            ( decrease ( energy ?r ) 1 )
        )
    )
    ( :action communicate_soil_data
        :parameters ( ?r - rover ?l - lander ?p - waypoint ?x - waypoint
            ?y - waypoint )
        :precondition ( and
            ( at ?r ?x )
            ( at_lander ?l ?y )
            ( have_soil_analysis ?r ?p )
            ( visible ?x ?y )
            ( >= ( energy ?r ) 4 )
        )
        :effect ( and
            ( communicated_soil_data ?p )
            ( decrease ( energy ?r ) 4 )
        )
    )
    ( :action communicate_rock_data
        :parameters ( ?r - rover ?l - lander ?p - waypoint ?x - waypoint
            ?y - waypoint )
        :precondition ( and
            ( at ?r ?x )
            ( at_lander ?l ?y )
            ( have_rock_analysis ?r ?p )
            ( >= ( energy ?r ) 4 )
            ( visible ?x ?y )
```

```
        )
        :effect ( and
            ( communicated_rock_data ?p )
            ( decrease ( energy ?r ) 4 )
        )
    )
    ( :action communicate_image_data
        :parameters ( ?r - rover ?l - lander ?o - objective ?m - mode
            ?x - waypoint ?y - waypoint )
        :precondition ( and
            ( at ?r ?x )
            ( at_lander ?l ?y )
            ( have_image ?r ?o ?m )
            ( visible ?x ?y )
            ( >= ( energy ?r ) 6 )
        )
        :effect ( and
            ( communicated_image_data ?o ?m )
            ( decrease ( energy ?r ) 6 )
        )
    )
)
```

C.8 Exemplo de problema para a versão númerica

O problema exemplo para a versão numérica do domínio do robô de exploração descrito a seguir possui as seguintes características: um único robô *rover0* equipado para análises de solo e de rocha, além de estar equipado para a obtenção de imagens, e uma única estação-base *general*. O robô *rover0* carrega a bordo uma câmera *camera0* com suporte aos modos *colour* e *high_res*.

Inicialmente o robô está posicionado na localidade *waypoint3*, seu compartimento está vazio e sua câmera está descalibrada. O objetivo do problema é obter uma amostra

de solo da localidade *waypoint2*, uma amostra de rocha da localidade *waypoint3*, e uma imagem no modo *high_res* do alvo *objective1*, devendo enviar os dados das mesmas à estação-base *general* situada na localidade *waypoint0*. De acordo com a função *energy*, a quantidade inicial de energia do robô é de 50 unidades. A energia do robô pode ser recarregada de 20 unidades na localidade *waypoint0*, mas um plano solução para o problema deve ser gerado de modo a minimizar a quantidade de recargas de energia.

```
( define ( problem roverprob1234 )
    ( :domain Rover )
    ( :objects
        general - Lander
        colour - Mode
        high_res - Mode
        low_res - Mode
        rover0 - Rover
        rover0store - Store
        waypoint0 - Waypoint
        waypoint1 - Waypoint
        waypoint2 - Waypoint
        waypoint3 - Waypoint
        camera0 - Camera
        objective0 - Objective
        objective1 - Objective
    )
    ( :init ( and
        ( visible waypoint1 waypoint0 )
        ( visible waypoint0 waypoint1 )
        ( visible waypoint2 waypoint0 )
        ( visible waypoint0 waypoint2 )
        ( visible waypoint2 waypoint1 )
        ( visible waypoint1 waypoint2 )
        ( visible waypoint3 waypoint0 )
        ( visible waypoint0 waypoint3 )
```

```
( visible waypoint3 waypoint1 )
( visible waypoint1 waypoint3 )
( visible waypoint3 waypoint2 )
( visible waypoint2 waypoint3 )
( = ( recharges ) 0 )
( at_soil_sample waypoint0 )
( in_sun waypoint0 )
( at_rock_sample waypoint1 )
( at_soil_sample waypoint2 )
( at_rock_sample waypoint2 )
( at_soil_sample waypoint3 )
( at_rock_sample waypoint3 )
( at_lander general waypoint0 )
( = ( energy rover0 ) 50 )
( at rover0 waypoint3 )
( store_of rover0store rover0 )
( empty rover0store )
( equipped_for_soil_analysis rover0 )
( equipped_for_rock_analysis rover0 )
( equipped_for_imaging rover0 )
( can_traverse rover0 waypoint3 waypoint0 )
( can_traverse rover0 waypoint0 waypoint3 )
( can_traverse rover0 waypoint3 waypoint1 )
( can_traverse rover0 waypoint1 waypoint3 )
( can_traverse rover0 waypoint1 waypoint2 )
( can_traverse rover0 waypoint2 waypoint1 )
( on_board camera0 rover0 )
( calibration_target camera0 objective1 )
( supports camera0 colour )
( supports camera0 high_res )
( visible_from objective0 waypoint0 )
( visible_from objective0 waypoint1 )
( visible_from objective0 waypoint2 )
```

```
        ( visible_from objective0 waypoint3 )
        ( visible_from objective1 waypoint0 )
        ( visible_from objective1 waypoint1 )
        ( visible_from objective1 waypoint2 )
        ( visible_from objective1 waypoint3 ) )
    )
    ( :goal ( and
        ( communicated_soil_data waypoint2 )
        ( communicated_rock_data waypoint3 )
        ( communicated_image_data objective1 high_res ) )
    )
    ( :metric minimize ( recharges ) )
)
```

Apêndice D

Demonstrações

D.1 Domínio do mundo dos blocos

As demonstrações a seguir referem-se às proposições apresentadas na Subseção 6.4.1. As quatro primeiras demostrações permitem demonstrar a Proposição 5.

Demonstração da Proposição 1: O operador $Pegar(x)$ do domínio do mundo dos blocos é inversível.

Seja α o operador $Pegar(x)$ e $\overline{\alpha}$ o operador $Soltar(x)$. Segundo a Definição 19, três condições são necessárias para que a ação α seja inversível:

1. (a) $pre(\overline{\alpha}) \subseteq (pre(\alpha) \cup add(\alpha)) \setminus del(\alpha)$

   A partir da descrição STRIPS desses dois operadores tem-se:

$$\{Segurando(x)\} \subseteq \{Limpo(x), SobreMesa(x), BracoLivre\}$$
$$\cup\{Segurando(x)\} \setminus \{SobreMesa(x), Limpo(x), BracoLivre\} \quad \text{(D.1)}$$
$$\{Segurando(x)\} \subseteq \{Segurando(x)\} \quad \text{(D.2)}$$

   (b) $add(\overline{\alpha}) = del(\alpha)$

A partir da descrição STRIPS desses dois operadores tem-se:

$$\{SobreMesa(x), Limpo(x), BracoLivre\} = \{SobreMesa(x), Limpo(x), BracoLivre\} \quad (D.3)$$

(c) $del(\overline{\alpha}) = add(\alpha)$

A partir da descrição STRIPS desses dois operadores tem-se:

$$\{Segurando(x)\} = \{Segurando(x)\} \quad (D.4)$$

2. $add(\alpha)$ é inconsistente com $pre(\alpha)$

   A partir da descrição STRIPS desses dois operadores tem-se que, para que essa condição seja satisfeita, $\{Segurando(x)\}$ deve ser inconsistente com $\{Limpo(x), SobreMesa(x), BracoLivre\}$. De fato, $Segurando(x)$ é inconsistente com $BracoLivre$, pois não pode existir um estado que contenha esses dois predicados.

3. $del(\alpha) \subseteq pre(\alpha)$

   A partir da descrição STRIPS desses dois operadores tem-se:

$$\{SobreMesa(x), Limpo(x), BracoLivre\} \subseteq \{Limpo(x), SobreMesa(x), BracoLivre\} \quad (D.5)$$

Como satisfaz as três condições da Definição 19, o operador $Pegar(x)$ é inversível ■

Demonstração da Proposição 2: O operador $Soltar(x)$ do domínio do mundo dos blocos é inversível.

Seja α o operador $Soltar(x)$ e $\overline{\alpha}$ o operador $Pegar(x)$. Segundo a Definição 19, três condições são necessárias para que a ação α seja inversível:

1. (a) $pre(\overline{\alpha}) \subseteq (pre(\alpha) \cup add(\alpha)) \setminus del(\alpha)$

$$\{Limpo(x), SobreMesa(x), BracoLivre\} \subseteq \{Segurando(x)\} \cup \{SobreMesa(x), Limpo(x), BracoLivre\} \setminus \{Segurando(x)\} \quad (D.6)$$

$$\{Limpo(x), SobreMesa(x), BracoLivre\} \subseteq \{SobreMesa(x), Limpo(x), BracoLivre\} \quad (D.7)$$

(b) $add(\overline{\alpha}) = del(\alpha)$

$$\{Segurando(x)\} = \{Segurando(x)\} \tag{D.8}$$

(c) $del(\overline{\alpha}) = add(\alpha)$

$$\{SobreMesa(x), Limpo(x), BracoLivre\} = \{SobreMesa(x), Limpo(x), BracoLivre\} \tag{D.9}$$

2. $add(\alpha)$ é inconsistente com $pre(\alpha)$

   Para que essa condição seja satisfeita, $\{SobreMesa(x), Limpo(x), BracoLivre\}$ deve ser inconsistente com $\{Segurando(x)\}$. De fato, $BracoLivre$ é inconsistente com $Segurando(x)$, pois não pode existir um estado que contenha esses dois predicados.

3. $del(\alpha) \subseteq pre(\alpha)$

$$\{SobreMesa(x), Limpo(x), BracoLivre\} \subseteq \{Limpo(x), SobreMesa(x), BracoLivre\} \tag{D.10}$$

Como satisfaz as três condições da Definição 19, o operador $Soltar(x)$ é inversível ■

Demonstração da Proposição 3: O operador $Desempilhar(x, y)$ do domínio do mundo dos blocos é inversível.

De acordo com a Definição 19, seja α o operador $Desempilhar(x, y)$, e seja $\overline{\alpha}$ o operador $Empilhar(x, y)$. Segundo a Definição 19, três condições são necessárias para que a ação α seja inversível:

1. (a) $pre(\overline{\alpha}) \subseteq (pre(\alpha) \cup add(\alpha)) \setminus del(\alpha)$

$$\{Segurando(x), Limpo(y)\} \subseteq \{Sobre(x, y), Limpo(x), BracoLivre\} \cup \{Segurando(x), Limpo(y)\} \setminus \{Limpo(x), Sobre(x, y), BracoLivre\} \tag{D.11}$$

$$\{Segurando(x), Limpo(y)\} \subseteq \{Segurando(x), Limpo(y)\} \tag{D.12}$$

(b) $add(\overline{\alpha}) = del(\alpha)$

$$\{Limpo(x), Sobre(x,y), BracoLivre\} = \{Limpo(x), Sobre(x,y), BracoLivre\} \quad (D.13)$$

(c) $del(\overline{\alpha}) = add(\alpha)$

$$\{Segurando(x), Limpo(y)\} = \{Segurando(x), Limpo(y)\} \quad (D.14)$$

2. $add(\alpha)$ é inconsistente com $pre(\alpha)$

   Para que essa condição seja satisfeita, $\{Segurando(x), Limpo(y)\}$ deve ser inconsistente com $\{Sobre(x,y), Limpo(x), BracoLivre\}$. De fato, $Segurando(x)$ é inconsistente com $BracoLivre$, pois não pode existir um estado que contenha esses dois predicados. Pela mesma razão, $Limpo(y)$ é inconsistente com $Sobre(x,y)$.

3. $del(\alpha) \subseteq pre(\alpha)$

$$\{Limpo(x), Sobre(x,y), BracoLivre\} \subseteq \{Sobre(x,y), Limpo(x), BracoLivre\} \quad (D.15)$$

Como satisfaz as três condições da Definição 19, o operador $Desempilhar(x,y)$ é inversível ■

Demonstração da Proposição 4: O operador $Empilhar(x,y)$ do domínio do mundo dos blocos é inversível.

De acordo com a Definição 19, seja α o operador $Empilhar(x,y)$, e seja $\overline{\alpha}$ o operador $Desempilhar(x,y)$. Segundo a Definição 19, três condições são necessárias para que a ação α seja inversível:

1. (a) $pre(\overline{\alpha}) \subseteq (pre(\alpha) \cup add(\alpha)) \setminus del(\alpha)$

$$\{Sobre(x,y), Limpo(x), BracoLivre\} \subseteq \{Segurando(x), Limpo(y)\} \cup \{Limpo(x), Sobre(x,y), BracoLivre\} \setminus \{Segurando(x), Limpo(y)\} \quad (D.16)$$

$$\{Sobre(x,y), Limpo(x), BracoLivre\} \subseteq \{Limpo(x), Sobre(x,y), BracoLivre\} \quad (D.17)$$

(b) $add(\overline{\alpha}) = del(\alpha)$

$$\{Segurando(x), Limpo(y)\} = \{Segurando(x), Limpo(y)\} \quad \text{(D.18)}$$

(c) $del(\overline{\alpha}) = add(\alpha)$

$$\{Limpo(x), Sobre(x,y), BracoLivre\} = \{Limpo(x), Sobre(x,y), BracoLivre\} \quad \text{(D.19)}$$

2. $add(\alpha)$ é inconsistente com $pre(\alpha)$

   Para que essa condição seja satisfeita, $\{Limpo(x), Sobre(x,y), BracoLivre\}$ deve ser inconsistente com $\{Segurando(x), Limpo(y)\}$. De fato, $BracoLivre$ é inconsistente com $Segurando(x)$, pois não pode existir um estado que contenha esses dois predicados. Pela mesma razão, $Sobre(x,y)$ é inconsistente com $Limpo(y)$.

3. $del(\alpha) \subseteq pre(\alpha)$

$$\{Segurando(x), Limpo(y)\} \subseteq \{Segurando(x), Limpo(y)\} \quad \text{(D.20)}$$

Como satisfaz as três condições da Definição 19, o operador $Empilhar(x,y)$ é inversível ■

Demonstração da Proposição 5: Qualquer problema no domínio do mundo dos blocos é uma tarefa de planejamento biderecional.

O domínio do mundo dos blocos é composto por quatro operadores: $Pegar(x)$, $Soltar(x)$, $Desempilhar(x,y)$, e $Empilhar(x,y)$. De acordo com as Proposições 1, 2, 3, e 4, todos esses quatro operadores são inversíveis. Como isso satisfaz a condição do Lema 1, qualquer instância de problema nesse domínio é uma tarefa de planejamento bidirecional ■

Demonstração da Proposição 6: Tarefas de planejamento no domínio do mundo dos blocos contém mínimos locais quando estados de seu espaço são avaliados pela heurística do planejador *FF*.

O domínio do mundo dos blocos é composto por quatro operadores: $Pegar(x)$, $Soltar(x)$,

$Desempilhar(x, y)$, e $Empilhar(x, y)$. Nenhum desses quatro operadores tem efeitos de remoção irrelevantes, de acordo com a Definição 21. Segundo o Lema 3, basta demonstrar que pelo menos um dos operadores desse domínio não é respeitado pela relaxação.

Sem perda de generalidade, sejam três blocos A, B, e C de um problema com $n \geq 3$ blocos. Seja um estado s em que os blocos A e B estão limpos, o braço robótico está segurando o bloco C, e todos os demais blocos estão em uma pilha que tem os blocos A e B no topo. Sejam também $Sobre(B, A)$ e $Sobre(C, B)$ duas proposições do objetivo desse problema. Um plano ótimo para esse problema deve ser iniciado por uma ação $Soltar(C)$. No entanto, para a versão relaxada desse problema, duas ações são aplicáveis no primeiro nível do grafo de planejamento, $Soltar(C)$ e $Empilhar(C, B)$. A segunda dessas duas ações resolve a proposição $Sobre(C, B)$ do objetivo sem comprometer a aplicabilidade das demais ações necessárias ao plano ótimo para essa tarefa relaxada. Nessa situação, o plano para a tarefa relaxada não é iniciado pela ação $Soltar(C)$, o que significa dizer que a ação $Soltar(x)$ não é respeitada pela relaxação.

Como uma ação do domínio do Mundo dos Blocos não satisfaz uma condição do Lema 3, tarefas de planejamento nesse domínio contém mínimos locais quando estados de seu espaço são avaliados pela heurística do planejador FF ■

D.2 Domínio do satélite

As demonstrações a seguir referem-se às proposições apresentadas na Subseção 6.4.2. As cinco primeiras demostrações permitem demonstrar a Proposição 12.

Demonstração da Proposição 7: O operador $turn_to(s, x, y)$ do domínio do satélite é inversível.

De acordo com a Definição 19, seja α o operador $turn_to(s, x, y)$, e seja $\overline{\alpha}$ o operador $turn_to(s, y, x)$. Segundo a Definição 19, três condições são necessárias para que a ação α seja inversível:

1. (a) $pre(\overline{\alpha}) \subseteq (pre(\alpha) \cup add(\alpha)) \setminus del(\alpha)$

$$\{pointing(s,x)\} \subseteq \{pointing(s,y)\}$$
$$\cup\{pointing(s,x)\}$$
$$\setminus\{pointing(s,y)\} \tag{D.21}$$
$$\{pointing(s,x)\} \subseteq \{pointing(s,x)\} \tag{D.22}$$

(b) $add(\overline{\alpha}) = del(\alpha)$

$$\{pointing(s,y)\} = \{pointing(s,y)\} \tag{D.23}$$

(c) $del(\overline{\alpha}) = add(\alpha)$

$$\{pointing(s,x)\} = \{pointing(s,x)\} \tag{D.24}$$

2. $add(\alpha)$ é inconsistente com $pre(\alpha)$

   Para que essa condição seja satisfeita, $\{pointing(s,x)\}$ deve ser inconsistente com $\{pointing(s,y)\}$. De fato, $pointing(s,x)$ é inconsistente com $pointing(s,y)$, pois não pode existir um estado que contenha esses dois predicados.

3. $del(\alpha) \subseteq pre(\alpha)$

$$\{pointing(s,y)\} \subseteq \{pointing(s,y)\} \tag{D.25}$$

Como satisfaz as três condições da Definição 19, o operador $turn_to(s,x,y)$ é inversível ■

Demonstração da Proposição 8: O operador $switch_on(i,s)$ do domínio do satélite é minimamente inversível.

De acordo com a Definição 20, seja α o operador $switch_on(i,s)$, e seja $\overline{\alpha}$ o operador $switch_off(i,s)$. Segundo a Definição 20, três condições são necessárias para que a ação α seja minimamente inversível:

1. $pre(\overline{\alpha}) \subseteq (pre(\alpha) \cup add(\alpha)) \setminus del(\alpha)$

$$\{power_on(i)\} \subseteq \{power_avail(s)\}$$
$$\cup\{power_on(i)\}$$
$$\setminus\{calibrated(i), power_avail(s)\} \quad (D.26)$$
$$\{power_on(i)\} \subseteq \{calibrated(i), power_on(i)\} \quad (D.27)$$

2. $add(\overline{\alpha}) \supseteq del(\alpha)$

$$\{power_avail(s)\} \supseteq \{power_avail(s)\} \quad (D.28)$$

3. $del(\overline{\alpha})$ é inconsistente com $pre(\alpha)$

   Para que essa condição seja satisfeita, $\{power_on(i)\}$ deve ser inconsistente com $\{power_avail(s)\}$. De fato, $power_on(i)$ é inconsistente com $power_avail(s)$, pois não pode existir um estado que contenha esses dois predicados.

Como satisfaz as três condições da Definição 20, o operador $switch_on(i, s)$ é minimamente inversível ■

Demonstração da Proposição 9: O operador $switch_off(i, s)$ do domínio do satélite é minimamente inversível.

De acordo com a Definição 20, seja α o operador $switch_off(i, s)$, e seja $\overline{\alpha}$ o operador $switch_on(i, s)$.

Como satisfaz as três condições da Definição 20, o operador $switch_off(i, s)$ é minimamente inversível ■

Demonstração da Proposição 10: O operador $calibrate(s, i, x)$ do domínio do satélite é minimamente inversível.

De acordo com a Definição 20, seja α o operador $calibrate(s, i, x)$, e seja $\overline{\alpha}$ o operador $calibrate(s, i, x)$.

Como satisfaz as três condições da Definição 20, o operador $calibrate(s, i, x)$ é minimamente inversível ■

Demonstração da Proposição 11: O operador $take_image(s, x, i, m)$ do domínio do satélite tem efeitos de adição estáticos.

De acordo com a Definição 21, seja α o operador $take_image(s, x, i, m)$.

Como satisfaz a condição da Definição 21, pois o único efeito da ação α não é removido por nenhuma ação do domínio, o operador $take_image(s, x, i, m)$ tem efeitos de adição estáticos ■

Demonstração da Proposição 12: Qualquer problema no domínio do satélite é uma tarefa de planejamento livre de becos sem saída.

O domínio do Satélite é composto por cinco operadores: $turn_to(s, x, y)$, $switch_on(i, s)$, $switch_off(i, s)$, $calibrate(s, i, x)$ e $take_image(s, x, i, m)$. De acordo com as Proposições 7, 8, 9, 10 e 11, todos esses cinco operadores ou são inversíveis, ou minimamente inversíveis, ou tem efeitos de adição estáticos. Como isso satisfaz a condição do Lema 2, qualquer instância de problema nesse domínio é uma tarefa livre de becos sem saída ■

Demonstração da Proposição 13: Tarefas de planejamento no domínio do satélite não contém mínimos locais quando estados de seu espaço são avaliados pela heurística do sistema *FF*.

O domínio do Satélite é composto por cinco operadores: $turn_to(s, x, y)$, $switch_on(i, s)$, $switch_off(i, s)$, $calibrate(s, i, x)$ e $take_image(s, x, i, m)$. De acordo com as Proposições 7, 8, 9, 10 e 11, todos esses cinco operadores são minimamente inversíveis relaxados. Como esse cinco operadores também são respeitados pela relaxação, são satisfeitas as duas condição do Lema 3, e as tarefas de planejamento nesse domínio contém mínimos locais quando estados de seu espaço são avaliados pela heurística do planejador *FF* ■

D.3 Domínio do robô de exploração

As demonstrações a seguir referem-se às proposições apresentadas na Subseção 6.4.3. As nove primeiras demostrações permitem demonstrar a Proposição 23.

Demonstração da Proposição 14: O operador $navigate(r, x, y)$ do domínio do robô de exploração é inversível.

De acordo com a Definição 19, seja α o operador $navigate(r, x, y)$, e seja $\overline{\alpha}$ o operador $navigate(r, y, x)$. Segundo a Definição 19, três condições são necessárias para que a ação α seja inversível:

1. (a) $pre(\overline{\alpha}) \subseteq (pre(\alpha) \cup add(\alpha)) \setminus del(\alpha)$

$$\{at(r, y)\} \subseteq \{at(r, x)\} \cup \{at(r, y)\} \setminus \{at(r, x)\} \tag{D.29}$$

$$\{at(r, y)\} \subseteq \{at(r, y)\} \tag{D.30}$$

   (b) $add(\overline{\alpha}) = del(\alpha)$

$$\{at(r, x)\} = \{at(r, x)\} \tag{D.31}$$

   (c) $del(\overline{\alpha}) = add(\alpha)$

$$\{at(r, y)\} = \{at(r, y)\} \tag{D.32}$$

2. $add(\alpha)$ é inconsistente com $pre(\alpha)$

   Para que essa condição seja satisfeita, $\{at(r, y)\}$ deve ser inconsistente com $\{at(r, x)\}$. De fato, $at(r, y)$ é inconsistente com $at(r, x)$, pois não pode existir um estado que contenha esses dois predicados.

3. $del(\alpha) \subseteq pre(\alpha)$

$$\{at(r, x)\} \subseteq \{at(r, x)\} \tag{D.33}$$

Como satisfaz as três condições da Definição 19, o operador $navigate(r, x, y)$ é inversível ■

Demonstração da Proposição 15: O operador $sample_soil(r, s, x)$ do domínio do robô de exploração tem efeitos de adição estáticos e efeitos de remoção irrelevantes.

De acordo com a Definição 21, seja α o operador $sample_soil(r, s, x)$.

Como satisfaz a condição da Definição 21, pois o único efeito de adição da ação α, $\{have_soil_analysis(x, p)\}$, não é removido por nenhuma ação do domínio, e o único efeito de remoção da ação α, $\{at_soil_sample(p)\}$, não faz parte da pré-condição de nenhuma outra ação do domínio, o operador $sample_soil(r, s, x)$ tem efeitos de adição estáticos ■

Demonstração da Proposição 16: O operador $sample_rock(r, s, x)$ do domínio do robô de exploração tem efeitos de adição estáticos e efeitos de remoção irrelevantes.

De acordo com a Definição 21, seja α o operador $sample_rock(r, s, x)$.

Como satisfaz a condição da Definição 21, pois o único efeito de adição da ação α, $\{have_rock_analysis(x, p)\}$, não é removido por nenhuma ação do domínio, e o único efeito de remoção da ação α, $\{at_rock_sample(p)\}$, não faz parte da pré-condição de nenhuma outra ação do domínio, o operador $sample_rock(r, s, x)$ tem efeitos de adição estáticos ■

Demonstração da Proposição 17: O operador $drop(r, s)$ do domínio do robô de exploração é minimamente inversível.

De acordo com a Definição 20, seja α o operador $drop(r, s)$, e seja $\overline{\alpha}$ o operador $sample_rock(r, s, x)$. Segundo a Definição 20, três condições são necessárias para que a ação α seja minimamente inversível:

1. $pre(\overline{\alpha}) \subseteq (pre(\alpha) \cup add(\alpha)) \setminus del(\alpha)$

$$\{empty(s)\} \subseteq \{full(s)\} \cup \{empty(s)\} \setminus \{full(s)\} \tag{D.34}$$

$$\{empty(s)\} \subseteq \{empty(s)\} \tag{D.35}$$

2. $add(\overline{\alpha}) \supseteq del(\alpha)$

$$\{full(s)\} \supseteq \{full(s)\} \tag{D.36}$$

3. $del(\overline{\alpha})$ é inconsistente com $pre(\alpha)$

   Para que essa condição seja satisfeita, $\{empty(s)\}$ deve ser inconsistente com $\{full(s)\}$. De fato, $empty(s)$ é inconsistente com $full(s)$, pois não pode existir um estado que contenha esses dois predicados.

Como satisfaz as três condições da Definição 20, o operador $drop(r, s)$ é minimamente inversível ■

Demonstração da Proposição 18: O operador $calibrate(r, c, o, x)$ do domínio do robô de exploração é minimamente inversível.

De acordo com a Definição 20, seja α o operador $calibrate(r, c, o, x)$, e seja $\overline{\alpha}$ o operador $calibrate(r, c, o, x)$. Segundo a Definição 20, três condições são necessárias para que a ação α seja minimamente inversível:

1. $pre(\overline{\alpha}) \subseteq (pre(\alpha) \cup add(\alpha)) \setminus del(\alpha)$

$$\{at(r, x)\} \subseteq \{at(r, x)\} \cup \{calibrated(c, r)\} \setminus \{\} \tag{D.37}$$

$$\{at(r, x)\} \subseteq \{at(r, x), calibrated(c, r)\} \tag{D.38}$$

2. $add(\overline{\alpha}) \supseteq del(\alpha)$

$$\{calibrated(c, r)\} \supseteq \{\} \tag{D.39}$$

3. $del(\overline{\alpha})$ é inconsistente com $pre(\alpha)$

   Para que essa condição seja satisfeita, $\{\}$ deve ser inconsistente com $\{at(r, x)\}$.

Como satisfaz as três condições da Definição 20, o operador $calibrate(r, c, o, x)$ é minimamente inversível ■

Demonstração da Proposição 19: O operador $take_image(r, x, o, c, m)$ do domínio do robô de exploração tem efeitos de adição estáticos.

De acordo com a Definição 21, seja α o operador $take_image(r, x, o, c, m)$.

Como satisfaz a condição da Definição 21, pois o único efeito da ação α, $\{have_image(r, o, m)\}$, não é removido por nenhuma ação do domínio, o operador $take_image(r, x, o, c, m)$ tem efeitos de adição estáticos ■

Demonstração da Proposição 20: O operador $communicate_soil_data(r, l, x, y, z)$ do domínio do robô de exploração tem efeitos de adição estáticos.

De acordo com a Definição 21, seja α o operador $communicate_soil_data(r, l, x, y, z)$.

Como satisfaz a condição da Definição 21, pois o único efeito da ação α, $\{communicated_soil_data(p)\}$ não é removido por nenhuma ação do domínio, o operador $communicate_soil_data(r, l, x, y, z)$ tem efeitos de adição estáticos ■

Demonstração da Proposição 21: O operador $communicate_rock_data(r, l, x, y, z)$ do domínio do robô de exploração tem efeitos de adição estáticos.

De acordo com a Definição 21, seja α o operador $communicate_rock_data(r, l, x, y, z)$.

Como satisfaz a condição da Definição 21, pois o único efeito da ação α, $\{communicated_rock_data(p)$ não é removido por nenhuma ação do domínio, o operador $communicate_rock_data(r, l, x, y, z)$ tem efeitos de adição estáticos ■

Demonstração da Proposição 22: O operador $communicate_image_data(r, l, o, m, x, y)$ do domínio do robô de exploração tem efeitos de adição estáticos.

De acordo com a Definição 21, seja α o operador $communicate_image_data(r, l, o, m, x, y)$.

Como satisfaz a condição da Definição 21, pois o único efeito da ação α, $\{communicated_image_data($ não é removido por nenhuma ação do domínio, o operador $communicate_image_data(r, l, o, m, x, y)$ tem efeitos de adição estáticos ■

Demonstração da Proposição 23: Qualquer problema no domínio do Robô de Exploração é uma tarefa de planejamento livre de becos sem saída.

O domínio do Robô de Exploração é composto por nove operadores: $navigate(r, x, y)$, $sample_soil(r, s, x)$, $sample_rock(r, s, x)$, $drop(r, s)$, $calibrate(r, c, o, x)$, $take_image(r, x, o, c, m)$, $communicate_soil_data(r, l, x, y, z)$, $communicate_rock_data(r, l, x, y, z)$ e $communicate_image_data(r, l,$

De acordo com as Proposições 14, 15, 16, 17, 18, 19, 20, 21 e 22, todos esses nove operadores ou são inversíveis, ou minimamente inversíveis, ou tem efeitos de adição estáticos ou efeitos de remoção irrelevantes. Como isso satisfaz a condição do Lema 2, qualquer instância de problema nesse domínio é uma tarefa livre de becos sem saída ■

Demonstração da Proposição 24: Tarefas de planejamento no domínio do Robô de Exploração não contém mínimos locais quando estados de seu espaço são avaliados pela heurística do sistema *FF*.

O domínio do Robô de Exploração é composto por nove operadores: $navigate(r, x, y)$, $sample_soil(r, s, x)$, $sample_rock(r, s, x)$, $drop(r, s)$, $calibrate(r, c, o, x)$, $take_image(r, x, o, c, m)$, $communicate_soil_data(r, l, x, y, z)$, $communicate_rock_data(r, l, x, y, z)$ e $communicate_image_data(r, l$
De acordo com as Proposições 14, 15, 16, 17, 18, 19, 20, 21 e 22, todos esses nove operadores ou são inversíveis, ou minimamente inversíveis, ou tem efeitos de adição estáticos ou efeitos de remoção irrelevantes. Como esse nove operadores também são respeitados pela relaxação, são satisfeitas as duas condição do Lema 3, e as tarefas de planejamento nesse domínio contém mínimos locais quando estados de seu espaço são avaliados pela heurística do planejador *FF* ■