Universidade Federal do Paraná
Setor de Ciências Exatas
Departamento de Informática
Programa de Pós-Graduação em Informática

Algoritmo para Planificação Baseada em *STRIPS*

Fabiano Silva

Dissertação de Mestrado do Programa de Pós-Graduação em Informática da Universidade Federal do Paraná.
Orientação: Marcos Alexandre Castilho

Curitiba, 4 de Setembro de 2000.

Resumo

Este trabalho apresenta uma nova abordagem para a classe de problemas de planificação em inteligência artificial baseados na representação proposicional *STRIPS*, reconhecidamente um problema de complexidade PSPACE-Completo.

Com o objetivo de contextualizar o cenário onde nosso trabalho se insere, apresentamos uma revisão do estado da arte na área, abordando os mais recentes algoritmos para o problema.

Propomos uma tradução do problema de planificação para um problema de alcançabilidade de sub-marcação em uma rede de Petri. Mostramos como resolver este último usando programação inteira, e como obter a solução para o problema de planificação original.

O algoritmo resultante desta abordagem, o Petriplan, é comparado com alguns planificadores atuais através de um conjunto de experimentos realizados nos moldes da competição de planificadores do AIPS de 1998.

Abstract

This work presents a new approach to the artificial intelligence planning problem based on STRIPS representation, known to be PSPACE-Complete.

We describe some recent algorithms to the planning problem related to this work, and compare some of these to our algorithm, the Petriplan.

Our approach define the planning problem as a sub-marking reachability problem in a Petri net, which is solved by the use of standard integer programming methods. The solution to planning problem is then obtained by the solution of the IP problem.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Suponhamos que alguém esteja assistindo televisão na sala e queira comer um bolo. Esta pessoa se levanta, vai até a cozinha, coloca farinha num recipiente, quebra alguns ovos, mistura ingredientes até obter a massa, que então coloca na forma e leva ao forno por 30 minutos. Finalmente, desenforma o bolo e pronto!

Consideremos que esta pessoa esteja na sala, não tenha o bolo e deseje tê-lo, mas não faz a menor idéia de quais são as ações e nem a ordem correta para realizá-las. Então ela tem em mãos um *problema de planificação*.

Um *planificador* é um sistema que recebe como entrada um estado inicial do mundo, um objetivo e um conjunto de ações que podem ser aplicadas e gera como saída uma seqüência de ações. Por exemplo, recebe o estado inicial "Estar na sala sem o bolo", o objetivo "Ter o bolo", diversas descrições de ações, e retorna uma seqüência de ações, como a descrita acima, que é chamada de *plano*.

Formalmente, um algoritmo que resolve um problema de planificação possui três entradas, codificadas em alguma linguagem formal:

1. uma descrição do mundo;
2. uma descrição do objetivo do agente; e
3. uma descrição das possíveis ações a serem executadas, freqüentemente chamada de *teoria do domínio*.

A saída de um planificador é uma seqüência de ações que, quando executadas em algum mundo satisfazendo a descrição do estado inicial, irá alcançar o objetivo. Note que esta formulação do problema de planificação é muito abstrata. De fato, ela realmente especifica uma *classe* de problemas de planificação parametrizados por linguagens usadas para representar o mundo, objetivos e ações.

Em geral, existem linguagens cada vez mais expressivas para representar o mundo, os objetivos de um agente e suas possíveis ações. Evidentemente a tarefa de escrever um

algoritmo de planificação é mais difícil para linguagens de representação mais expressivas e o tempo de resposta do algoritmo resultante aumenta proporcionalmente. O fato é que é difícil encontrar um bom compromisso entre o poder de representação e a eficiência do algoritmo.

Por exemplo, pode-se usar lógica proposicional para descrever os efeitos de ações, mas isto torna impossível a descrição de ações com efeitos quantificados universalmente. Ainda, pode-se descrever os efeitos de ações através do cálculo de predicado de primeira ordem, mas ele assume que todos os efeitos são determinísticos. Pode ser muito difícil representar precisamente os efeitos de uma ação não determinística sem alguma forma de representação probabilística.

A primeira proposta de solução para o problema de planificação integrando um sistema formal e um algoritmo correspondente foi apresentada por Fikes e Nilsson em 1971, o sistema *STRIPS* [FN71]. O algoritmo proposto trata o problema de planificação como um problema de busca tradicional. Em outras palavras, o algoritmo procura por uma solução utilizando uma busca em uma árvore de estados do mundo de tamanho exponencial. Apesar de simples, o algoritmo trata na prática um número pequeno de problemas.

A grande relevância do trabalho de Fikes e Nilsson é o sistema formal proposto, que permite representar ações e estados de maneira simples, permitindo uma independência entre a linguagem e o algoritmo que resolve o problema. Todos os planificadores descritos neste documento são baseados nesta representação, que é reconhecidamente o mais popular sistema formal adotado pela comunidade.

Por 20 anos o problema de planificação foi tratado basicamente por duas abordagens: procedimentos de busca baseados em *STRIPS* e procedimentos de prova de teoremas. Ambas ineficientes em termos práticos, caindo em buscas exaustivas e explosões combinatoriais. Uma vez que a complexidade do problema de planificação em *STRIPS* é PSPACE-Completo [ENV91, Byl94].

No entanto, com o surgimento de métodos rápidos para tratamento de problemas de satisfabilidade (SAT) [SLM92, SKC94], a abordagem baseada em prova de teoremas foi retomada. Em 1992, Kautz e Selman apresentaram uma tradução da representação *STRIPS* para cálculo proposicional, resultando em um planificador extremamente rápido, o *Satplan* [KS92, KS96].

Mas a grande revolução veio três anos depois quando Blum e Furst apresentaram o *Graphplan* [BF95]. O *Graphplan* inova ao mostrar que, a partir de uma descrição *STRIPS*, pode-se construir um grafo que reduz sensivelmente o espaço de busca para o problema.

A partir do *Satplan* e do *Graphplan*, houve uma grande motivação por novas pesquisas em planificação, até então estagnadas, como nos mostra Weld em sua excelente revisão dos recentes avanços na área [Wel99][1]

[1]Neste trabalho, Weld apresenta o estado da arte na área até 1998, não contemplando as mais recentes abordagens para o problema, principalmente aquelas baseadas em restrições. Nosso texto complementa

Por exemplo, a idéia de construir um grafo para reduzir o espaço de busca foi bem aproveitada por Kautz e Selman, que mostraram que o grafo pode ser traduzido para uma instância SAT muito menor que a gerada pelo *Satplan*. O algoritmo resultante, *Blackbox* [KS99], foi a sensação do *AIPS*[2] *Planning Competition* de 1998 [McD98a]. Outro competidor de destaque, o *HSP* de Bonet e Geffner [BG98], também resgata antigas técnicas de IA no contexto de planificação.

Recentemente, a Programação Inteira (PI) também mostrou-se promissora na solução de problemas de planificação, através de sua modelagem como um conjunto de restrições sobre variáveis inteiras. Podemos encontrar resultados preliminares interessantes na literatura dos últimos dois anos. Bockmayr e Dimopoulos [BD98] usaram variáveis inteiras 0-1 de modo similar a abordagem SAT e examinaram o efeito de adicionar restrições redundantes ao problema de PI. Vossen e colegas [VBLN99, VBLN00] discutem a importância de encontrar a representação correta do problema de planificação em termos de um problema de PI. Eles mostraram várias formulações possíveis e suas vantagens. Kautz e Walser [KW99] tratam da solução dos problemas de PI resultantes usando algoritmos de busca local inteira.

Apesar dos bons resultados alcançados na aplicação de programação inteira, esta área ainda está aberta para novas investigações, explorando novas possibilidades do uso de métodos para PI na solução de problemas de planificação em IA.

A questão de tratar planificação como sistemas de restrições também foi considerada por van Beek e Chen [BC99], que propõem tratar ambas abordagens SAT e PI como um caso particular de outra técnica básica de IA: a técnica de Satisfação de Restrições (*Constraint Satisfaction Problem - CSP*). Eles mostraram como a sua abordagem é melhor em termos de tempo de processamento e utilização de memória.

Em suma, o que era um tema bem definido, apesar de relativamente abandonado pelos pesquisadores, hoje em dia é uma área de pesquisa extremamente complexa, que integra diversas áreas do conhecimento, indo desde tradicionais problemas de IA, como satisfabilidade e programação por restrições, até buscas heurísticas, teoria dos grafos e programação inteira. A figura 1.1 mostra os atuais relacionamentos destas áreas e os respectivos trabalhos que as relacionam.

Nosso trabalho dá mais uma contribuição neste contexto, mostrando como podemos definir um problema de planificação como um problema de alcançabilidade em redes de Petri [Mur89]. Mostramos como integrar planificação baseada em *STRIPS*, *Graphplan*, redes de Petri e programação inteira, abrindo caminho para uma melhor compreensão de como todos estes temas se relacionam.

O texto está estruturado como segue: no capítulo 2 é feita uma revisão bibliográfica dos trabalhos relacionados e os principais aspectos do problema de planificação baseado

[Wel99] neste sentido.

[2] *International Conference on Artificial Intelligence Planning Systems*

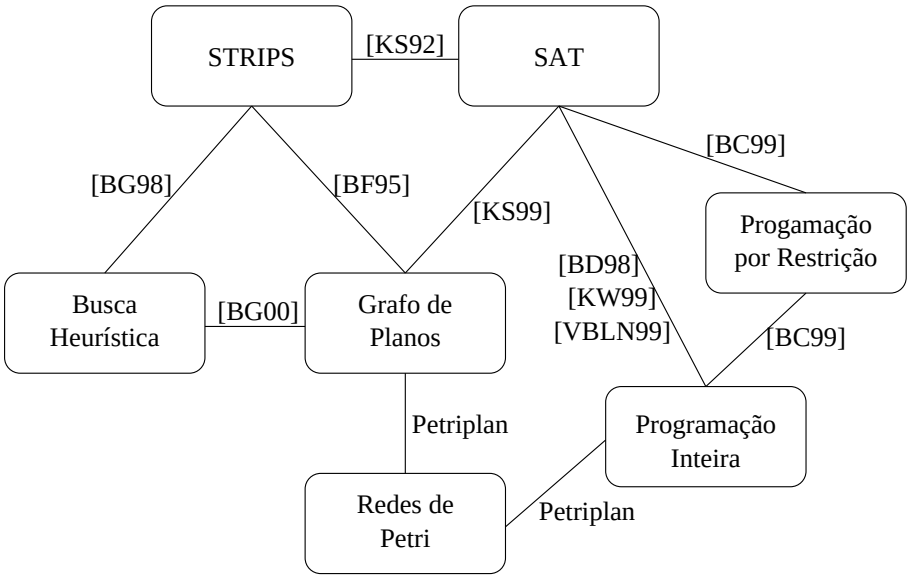

Figura 1.1: Relacionamento entre áreas do conhecimento aplicadas em planificação.

em *STRIPS*. No capítulo 3 introduzimos conceitos de Redes de Petri e de Programação Inteira que servem de base para o algoritmo proposto, o Petriplan. No capítulo 4 comparamos nosso algoritmo com outros nos moldes da Competição de Planificadores do AIPS98 [McD98a]. No capítulo 5 são apresentadas as conclusões e propostas de trabalhos futuros.

Capítulo 2

Algoritmos para Planificação

Neste capítulo descrevemos o estado da arte sobre o problema de planificação baseado em *STRIPS*. Iniciamos com a abordagem clássica e em seguida apresentamos os algoritmos recentes propostos nos últimos cinco anos para o problema. Descreveremos apenas os aspectos teóricos destas abordagens, deixando para o capítulo 4 as considerações de implementação e desempenho.

2.1 A representação *STRIPS*

Um dos primeiros sistemas desenvolvidos para o problema de planificação foi o *STRIPS* (*STanford Research Institute Problem Solver*) em 1971 [FN71]. Este sistema usa uma linguagem baseada em cálculo proposicional que fornecia mecanismos para se definir ações e especificar estados do mundo.

Na representação *STRIPS*, ou simplesmente *STRIPS*, os estados do mundo são representados por conjunções de literais [1] instanciados, ou seja, predicados aplicados sobre constantes. Por exemplo, o estado inicial para o problema do bolo apresentado no capítulo 1 pode ser descrito como:

$$\mathsf{EstarNa(sala)} \wedge \mathsf{Ter(ingredientes)} \wedge \neg\mathsf{Ter(bolo)} \wedge \cdots$$

Numa descrição completa do estado inicial, todas as fórmulas atômicas não explicitamente listadas são assumidas como falsas (o que é chamado de Hipótese do Mundo Fechado[2] [Rei78], usada para tratar o Problema do Quadro[3] [MH69]). Assim o literal $\neg\mathsf{Ter(bolo)}$ pode ser suprimido da descrição do estado inicial.

Analogamente o estado meta ou estado final, pode ser descrito por:

[1] Neste texto usamos os termos "literal" e "proposição" como sinônimos.
[2] *Closed World Assumption.*
[3] *Frame Problem.*

$$\mathsf{EstarNa(cozinha)} \wedge \mathsf{Ter(bolo)}$$

A representação *STRIPS* restringe o tipo dos estados meta que podem ser especificados para aqueles compostos por conjunções de literais positivos.

Estes dois conceitos, o estado inicial e o final, são dois dos três componentes que definem um problema de planificação. O terceiro é denominado de teoria do domínio, denotada por Δ: é a descrição formal das ações que estão disponíveis para o agente.

Em *STRIPS* as ações são constituídas de três partes:

- A *descrição da ação*, que define o nome e os parâmetros da ação;
- A *pré-condição*, que é uma conjunção de literais positivos que devem ser verdadeiros para que a ação possa ser aplicada; e
- O *efeito*[4], que é uma conjunção que pode incluir literais positivos ou negativos. Eles descrevem as alterações que devem ser realizadas sobre um estado do mundo a fim de obter um outro.

Deste modo, podemos definir a ação *Assar* como:

Descrição	:	$\mathsf{Assar(massa)}$
Pré-Condição	:	$\mathsf{Ligado(forno)} \wedge \mathsf{Dentro(massa, forma)}$
Efeito	:	$\mathsf{Dentro(forma, forno)} \wedge \mathsf{Dentro(bolo, forma)} \wedge$ $\neg\mathsf{Dentro(massa, forma)}$

Ações podem ser executadas apenas quando suas pré-condições são verdadeiras. Quando uma ação é executada ela muda a descrição do mundo (estado corrente) da seguinte forma: todos os literais positivos na conjunção efeito são adicionados ao estado enquanto que todos os literais negativos são removidos. Por exemplo, se o estado corrente do mundo for:

$$\mathsf{EstarNa(cozinha)} \wedge \mathsf{Ligado(forno)} \wedge \mathsf{Dentro(massa, forma)},$$

o estado do mundo após a ação $\mathsf{Assar(massa)}$ é o seguinte:

$$\mathsf{EstarNa(cozinha)} \wedge \mathsf{Ligado(forno)} \wedge \mathsf{Dentro(forma, forno)} \wedge \mathsf{Dentro(bolo, forma)},$$

onde observa-se que os elementos não mencionados na conjunção efeito permanecem inalterados.

[4]Na formulação original de Fikes e Nilsson [FN71] a conjunção efeito é representada por duas listas de literais: a lista de inclusão contendo os literais positivos, e a lista de remoção contendo os literais negativos.

A entrada para um planificador é dada por uma descrição do estado inicial, uma descrição do estado meta e a teoria do domínio Δ. Quando chamado com esta entrada, um planificador deve retornar uma seqüência de ações que transforme o estado inicial no estado meta.

Como dissemos, a seqüência de ações gerada como saída é chamada *plano*. Definimos formalmente um plano como sendo a tupla $< \mathcal{A}, \mathcal{O}, \mathcal{L} >$ onde $\mathcal{A}$ é um conjunto de ações, $\mathcal{O}$ é um conjunto de *restrições de ordenação* sobre o conjunto de ações $\mathcal{A}$ e $\mathcal{L}$ é um conjunto de *ligações causais*.

Por exemplo, considere o plano $P =< \mathcal{A}, \mathcal{O}, \mathcal{L} >$ dado por:

- $\mathcal{A} = \{a_1, a_2, a_3, a_4\}$;
- $\mathcal{O} = \{(a_1 < a_2), (a_1 < a_4), (a_3 < a_2), (a_2 < a_4)\}$;
- $\mathcal{L} = \{(a_2 \stackrel{P}{\rightarrow} a_4), (a_1 \stackrel{Q}{\rightarrow} a_2)\}$.

Podemos observar que o conjunto $\mathcal{O}$ é composto por restrições de ordem sobre os elementos de $\mathcal{A}$. Tais restrições definem a precedência das ações de $\mathcal{A}$ para se partir do estado inicial e chegar ao estado meta.

As ligações causais do conjunto $\mathcal{L}$ do exemplo têm o objetivo de determinar dependências entre as ações de $\mathcal{A}$ da seguinte forma: a ação a_2 tem como efeito a proposição P que é uma pré-condição da ação a_4; da mesma forma, a ação a_1 tem como efeito a proposição Q que é pré-condição de a_2. Com estas informações podemos evitar que ações que tragam inconsistências ao plano sejam inseridas na solução. Por exemplo, uma ação que tenha como efeito $\neg Q$, executada antes de a_2, trará uma inconsistência para esta ação.

Mapeando esta representação para o problema do bolo temos o seguinte plano $B =< \mathcal{A}_B, \mathcal{O}_B, \mathcal{L}_B >$ como a solução para o problema:

$$
\begin{aligned}
\mathcal{A}_B &= \{ \ \mathsf{IrPara(cozinha)}, \mathsf{Assar(massa)}, \mathsf{Colocar(massa, forma)}, \\
&\quad \mathsf{Ligar(forno)}, \mathsf{Preparar(massa, ingredientes)}, \\
&\quad \mathsf{Retirar(forma, forno)}, \mathsf{Desenformar(bolo)} \ \ \}; \\
\mathcal{O}_B &= \{ \ [\mathsf{IrPara(cozinha)} < \mathsf{Ligar(forno)}], \\
&\quad [\mathsf{Ligar(forno)} < \mathsf{Assar(massa)}], \\
&\quad [\mathsf{Preparar(massa, ingredientes)} < \mathsf{Assar(massa)}], \\
&\quad [\mathsf{Colocar(massa, forma)} < \mathsf{Assar(massa)}], \\
&\quad [\mathsf{Preparar(massa, ingredientes)} < \mathsf{Colocar(massa, forma)}], \\
&\quad [\mathsf{Retirar(forma, forno)} < \mathsf{Desenformar(bolo)}] \ \ \}; \\
\mathcal{L}_B &= \{ \ [\mathsf{Assar(massa)} \stackrel{\mathsf{Dentro(forma,forno)}}{\longrightarrow} \mathsf{Retirar(forma, forno)}] \ \ \}.
\end{aligned}
$$

Se considerarmos as restrições de ordenação das ações de um plano, podemos classificá-lo em parcial ou completamente ordenado. Um *plano parcialmente ordenado* é aquele onde apenas parte das ações estão ordenadas, garantindo apenas ordenações essenciais.

Considerando o conjunto de restrições de ordenação $\mathcal{O}$ do exemplo do plano P temos um conjunto de ações parcialmente ordenado, pois estas restrições não definem uma única forma de ordenação total das ações, permitindo obtenção tanto da seqüência $p_1 = (a_1, a_3, a_2, a_4)$ quanto da seqüência $p_2 = (a_3, a_1, a_2, a_4)$.

Um *plano completamente ordenado* é aquele onde todas as ações estão ordenadas, como nas seqüências de ações p_1 e p_2, formando uma seqüência cronológica, partindo do estado inicial e chegando ao estado meta.

Usaremos no decorrer do texto o termo "plano", além da definição anterior, como referência para uma seqüência de ações parcial ou completamente ordenada.

Para o exemplo do plano B, as duas seqüências abaixo são planos completamente ordenados:

$$\begin{aligned}
S_a &= (\ \mathsf{IrPara(cozinha), Ligar(forno), Preparar(massa, ingredientes),} \\
&\quad\ \mathsf{Colocar(massa, forma), Assar(massa),} \\
&\quad\ \mathsf{Retirar(forma, forno), Desenformar(bolo)}\ ). \\
S_b &= (\ \mathsf{IrPara(cozinha), Preparar(massa, ingredientes),} \\
&\quad\ \mathsf{Colocar(massa, forma), Ligar(forno), Assar(massa),} \\
&\quad\ \mathsf{Retirar(forma, forno), Desenformar(bolo)}\ ).
\end{aligned}$$

Podemos ainda classificar os planos em parcial ou completamente especificados. Um *plano parcialmente especificado* é aquele cujas ações não levam ao estado meta mas sim a um estado intermediário do mundo. Já um *plano completamente especificado* leva do estado inicial ao estado meta, ou seja, é uma solução para o problema. Por exemplo, um plano parcialmente especificado para o problema do bolo pode ser:

$$\begin{aligned}
\mathcal{A}_{Bpar} &= \{\ \mathsf{Assar(massa), Ligar(forno), Retirar(forma, forno),}\ \}; \\
\mathcal{O}_{Bpar} &= \{\ [\mathsf{Ligar(forno)} < \mathsf{Assar(massa)}], \\
&\quad\ [\mathsf{Assar(massa)} < \mathsf{Retirar(forma, forno)}]\ \}; \\
\mathcal{L}_{Bpar} &= \{\ [\mathsf{Assar(massa)} \overset{\mathsf{Dentro(forma,forno)}}{\longrightarrow} \mathsf{Retirar(forma, forno)}]\ \}.
\end{aligned}$$

e uma seqüência de ações para este plano é:

$$S_{Bpar} = (\ \mathsf{Ligar(forno), Assar(massa), Retirar(forma, forno)}\ ).$$

Uma maneira óbvia de se construir um planificador usando *STRIPS* é tratá-lo como um problema de busca no espaço de estados possíveis do mundo. Foi desta maneira que

a planificação foi originamente considerada e vários planificadores foram propostos com a mesma base. Este tratamento é detalhado a seguir.

2.1.1 Procedimentos de busca

Podemos abordar o problema de planificação como uma busca num grafo cujos vértices são os estados e as arestas são as ações que transformam um estado em outro. Desta forma podemos aplicar os algoritmos de busca [RN95] e procedimentos heurísticos [Pea84] conhecidos na literatura clássica de IA.

Com o objetivo de generalizar a aplicação de qualquer mecanismo de busca, vamos defini-lo como um algoritmo não-determinístico. Isto é, quando especificarmos um algoritmo de planificação, usaremos a rotina não-determinística Escolhe, que pega um conjunto de opções e magicamente seleciona a opção correta. A vantagem de se utilizar esta rotina é que podemos facilmente implementá-la usando qualquer método de busca que desejarmos. Isto permitirá uma implementação bastante flexível e simplificará o algoritmo de planificação.

Os algoritmos de busca podem ser divididos em duas classes: os que partem do estado inicial em direção ao estado meta, e os que partem do estado meta em direção ao inicial. No primeiro caso, o algoritmo é dado por um laço que a cada passo seleciona uma ação usando a rotina Escolhe e a inclui na solução até que o estado meta seja alcançado. Este tipo de planificador é chamado de *progressivo*. No outro caso, o procedimento escolhe ações que tenham como efeitos os literais do estado meta e a cada passo redefine este estado meta como a conjunção das pré-condições destas ações escolhidas. Este processo é chamado de planificação *regressiva*.

As figuras 2.1 e 2.2 mostram exemplos de planificadores progressivo e regressivo, respectivamente. Os procedimentos Progressivo e Regressivo fazem a busca por ações no *espaço de estados do mundo* que, dependendo do conjunto de ações possíveis, pode se tornar um espaço de busca exponencial. Isto inviabiliza os planificadores em termos de tempo e memória.

Outra abordagem para o espaço de busca é o *espaço de planos*. No espaço de planos os vértices do grafo de busca representam planos parcialmente especificados e as arestas representam operações de refinamento de planos, como a inclusão de uma ação num plano. O estado inicial representa um plano vazio e o estado meta representa o plano completo, ou seja, a solução. Note que os planificadores baseados nos estados do mundo retornam um caminho no grafo de busca, já os baseados no espaço de planos retornam o vértice que contém o estado meta.

Um planificador para o espaço de planos inicia sua busca a partir de um plano nulo, onde existem apenas duas ações, a_0 e a_∞, a primeira é uma ação que não tem pré-condições e seus efeitos são as proposições presentes no estado inicial. A ação a_∞ tem

Progressivo(estado, objetivo, Δ, plano)

1. Se Satisfaz(estado, objetivo) então retorne plano
2. ação $\leftarrow$ Escolhe(Δ, estado);
3. Se nenhuma ação foi encontrada retorne falha;
4. novo-estado $\leftarrow$ Executa(ação, estado) ;
5. Retorne Progressivo(novo-estado, objetivo, Δ, Inclui(ação, plano)) ;

onde: a rotina Satisfaz verifica se o estado satisfaz logicamente o objetivo; a rotina Escolhe escolhe não-deterministicamente uma ação de Δ (conjunto de ações) cuja pré-condição é satisfeita pelo estado; a rotina Executa aplica a ação sobre o estado gerando um novo-estado; e a rotina Inclui retorna um novo plano a partir da inclusão da ação no plano.

Figura 2.1: Algoritmo progressivo para planificação baseado em busca no espaço de estados do mundo.

Regressivo(estado, objetivo, Δ, plano)

1. Se Satisfaz(estado, objetivo) então retorne plano
2. ação $\leftarrow$ Escolhe(Δ, objetivo);
3. Se nenhuma ação foi encontrada retorne falha;
4. novo-objetivo $\leftarrow$ Executa(ação, objetivo) ;
5. Retorne Regressivo(estado, novo-objetivo, Δ, Inclui(ação, plano)) ;

onde: a rotina Satisfaz verifica se o estado satisfaz logicamente o objetivo; a rotina Escolhe escolhe não-deterministicamente uma ação de Δ (conjunto de ações) cujo efeito está contido na conjunção objetivo; a rotina Executa constrói um novo objetivo a partir do objetivo atual, incluindo as pré-condições da ação e removendo seus os efeitos; e a rotina Inclui retorna um novo plano a partir da inclusão da ação no plano.

Figura 2.2: Algoritmo regressivo para planificação baseado em busca no espaço de estados do mundo.

como pré-condição a conjunção das proposições do estado meta e não tem nenhum efeito. Partindo deste plano nulo o planificador faz escolhas não-determinísticas até que todas as pré-condições de todas as ações sejam suportadas por ligações causais. Portanto o primeiro passo é escolher ações que gerem como efeitos as pré-condições da ação a_∞ e assim sucessivamente.

Os sistemas para problemas de planificação clássica baseada em *STRIPS* derivam diretamente do sistema proposto por Newell e Simon [NS63], o *GPS - General Problem Solver*, cuja proposta era um sistema para a solução de problemas genéricos, incluindo problemas de planificação, representados formalmente numa linguagem lógica. Podemos considerá-lo como sendo o primeiro sistema planificador. Partindo desta proposta, o sistema *STRIPS* de Fikes e Nilsson trata mais especificamente problemas de planificação, apresentando uma representação proposicional simples, permitindo tratar o problema como uma busca regressiva no espaço do mundo.

Vários outros planificadores foram desenvolvidos a partir do *STRIPS* de Fikes e Nilsson, buscando melhorar o seu desempenho e agregar maior capacidade de representação à linguagem. A Figura 2.4 apresenta uma hierarquia de alguns planificadores que surgiram de 1971 a 1991.

Em 1973, Sussman apresentou o planificador *HACKER* [Sus75], que foi um descendente direto do *STRIPS* de Fikes e Nilsson. Foi neste artigo que Sussman apresentou a "Anomalia de Sussman" no mundo dos blocos, onde os estado inicial é dado por:

$$\mathsf{sobre}(\mathsf{C},\mathsf{A}) \wedge \mathsf{sobre}(\mathsf{A},\mathsf{Mesa}) \wedge \mathsf{sobre}(\mathsf{B},\mathsf{Mesa});$$

e o estado meta por:

$$\mathsf{sobre}(\mathsf{C},\mathsf{Mesa}) \wedge \mathsf{sobre}(\mathsf{A},\mathsf{B}) \wedge \mathsf{sobre}(\mathsf{B},\mathsf{C}).$$

Nenhum dos planificadores da época, nem o próprio *HACKER* conseguiu resolver este problema simples pois tratavam as sub-metas isoladamente caindo em mínimos locais como:

$$\mathsf{sobre}(\mathsf{C},\mathsf{Mesa}) \wedge \mathsf{sobre}(\mathsf{A},\mathsf{B}).$$

A figura 2.3 mostra estes três estados.

Sistemas subseqüentes como *WARPLAN* [War74] e *INTERPLAN* [Tat74] abordaram diretamente o problema da Anomalia de Sussman através da reordenação das ações do plano.

Em 1975, Sacerdoti [Sac75] desenvolveu o planificador *NOAH*, que foi o primeiro a adotar a busca no espaço de planos. Este planificador tratava de planos parcialmente ordenados mas sem explorar as propriedades intrínsecas deste tipo de plano.

Tate apresentou em [Tat77] um novo planificador, o *NONLIN*, que foi o primeiro sistema a usar ligações causais entre as ações, permitindo detectar e tratar inconsistências formalmente.

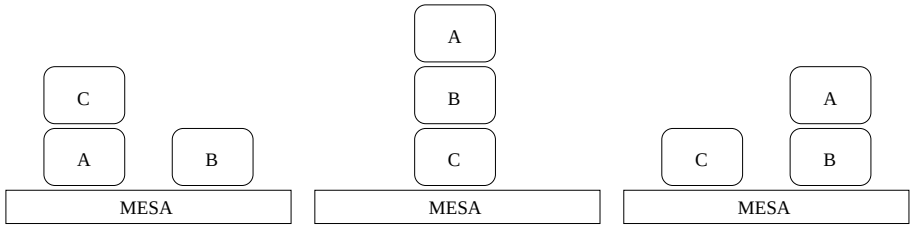

Figura 2.3: Estado inicial, final e mínimo local para a anomalia de Sussman no mundo de blocos.

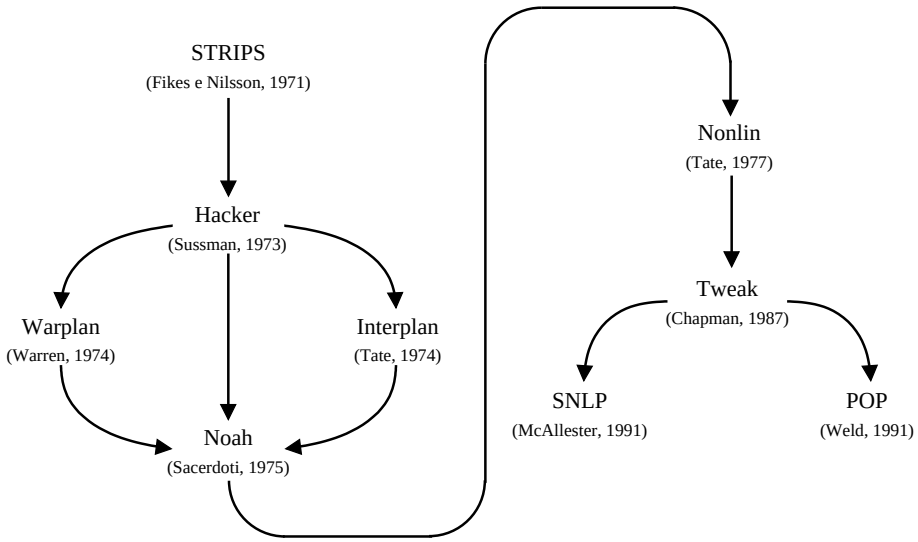

Figura 2.4: Hierarquia com alguns planificadores que utilizam representações derivadas de *STRIPS*.

Em [Cha87], Chapman formalizou as idéias de Sacerdoti sobre planos parcialmente ordenados e construiu o planificador *TWEAK*, provando que o procedimento deste planificador é completo. *TWEAK* foi a base para os algoritmos desenvolvidos por McAllester e Rosenblitt [MR91] (*SNLP*), e por Barrett, Soderland e Weld [BSW91] (*POP*). Estes planificadores apresentavam bons resultados em domínios simples, mas em problemas práticos, onde os domínios são mais complexos, sua aplicação é inviável, resultando invariavelmente em explosões combinatoriais.

2.1.2 Observações

Os sistemas de planificação baseados em *STRIPS* apresentam uma grande limitação quanto à capacidade de representação da linguagem. Entretanto, devido a sua simplicidade, esta representação possibilita o desenvolvimento de estratégias de busca bastante otimizadas, sendo utilizada como estrutura de representação dos planificadores mais rápidos da atualidade.

O grande problema apresentado pelo planificador *STRIPS* de Fikes e Nilsson e pe-

los planificadores derivados diretamente deste é que eles utilizam métodos exaustivos de busca sobre espaços de estados ou de planos que invariavelmente resultam em explosões combinatoriais, impossibilitando sua aplicação em problemas do mundo real.

O problema de planificação baseado na representação proposicional *STRIPS* é em geral PSPACE-Completo [ENV91, Byl94], inviabilizando qualquer abordagem exaustiva. Este fato é o principal motivador de pesquisas que tentam reduzir o espaço de busca ou propor novas estratégias para o tratamento do problema. Alguns destes métodos serão tratados nas próximas seções.

2.2 Planificação heurística

Como vimos, a abordagem do problema de planificação como um problema de busca simplifica a especificação de algoritmos, mas apresenta um problema: "Como implementar a rotina não-determinística que escolhe o próximo passo de forma eficiente?".

O mais simples seria utilizar um procedimento exaustivo de busca, como busca em amplitude. Esta escolha nos dá um procedimento completo, mas em termos práticos acaba se tornando inviável para problemas reais devido ao grande número de possibilidades geradas durante a busca. Outra saída é a utilização de funções heurísticas para o direcionamento do processo de busca, diminuindo as possibilidades a cada passo, como por exemplo o algoritmo clássico $A^\star$.

Uma outra abordagem é a utilização de procedimentos de busca não completos baseados em funções heurísticas [Pea84]. Apesar de não produzir um procedimento completo este tipo de estratégia fornece um ganho de desempenho considerável, gerando planificadores extremamente rápidos, como a família de planificadores *HSP - Heuristic Search Planner* de Bonet e Geffner [BG98, BG99, BG00].

Nas próximas seções apresentamos a função heurística utilizada no planificador *HSP* e a sua utilização no procedimento de busca.

2.2.1 A função heurística

O ponto principal para se determinar uma boa função heurística para direcionar um procedimento de busca é obter uma função capaz de medir de maneira rápida e simples o custo de sair de um determinado estado e chegar ao estado meta, ou seja, uma função que mede a que distância estamos do estado final.

Em problemas de planificação uma função com as características citadas tem que ser capaz de medir a distância (ou custo) entre qualquer estado do mundo e o estado final de maneira ótima. A questão é que tal função tem a mesma complexidade de resolver o problema, pois se sabemos o custo de um caminho ótimo no espaço de estados que leva do estado inicial ao estado final também conhecemos este caminho, que é a solução

para o problema de planificação. Desta forma, trabalhar com uma função de custo ótimo inviabiliza a aplicação desta técnica. Assim, passamos a analisar uma versão reduzida do problema de planificação.

Como vimos anteriormente, uma ação em *STRIPS* altera o estado atual do mundo incluindo os literais positivos de sua conjunção efeito e eliminando deste estado os literais negativos do efeito.

Nesta versão simplificada do problema de planificação apenas incluiremos os literais positivos do efeito no estado do mundo. Ou seja, a lista de remoção das ações representadas em *STRIPS* será ignorada. Vale notar que esta versão simplificada não é equivalente à versão original do problema de planificação, e que uma solução na versão simplificada não corresponde a uma solução do problema original. No entanto, podemos tomar o tamanho (ou custo) desta solução simplificada como uma cota inferior para o tamanho da solução do problema original.

Por exemplo, considerando o problema reduzido P' do problema de planificação P, temos um procedimento que aplica ações apenas incluindo literais no estado s até que todos os literais presentes no estado meta estejam em s. Assim a função de custo ótimo $h'(s)$ para se alcançar o estado meta em P' é uma cota inferior para a função de custo ótimo $h^\star(s)$ em P, pois não há remoção de literais em s para o problema P'.

Poderíamos então usar em nosso planificador a função heurística $h'(s)$ para direcionar o procedimento de busca, sendo esta função uma *heurística admissível* para o problema P, pois não superestima o custo ótimo da solução para P. No entanto, calcular $h'(s)$ é NP-Difícil [ENV91, Byl94], o que inviabiliza o seu uso.

Então partimos para uma aproximação de $h'(s)$. Nesta aproximação, estimamos individualmente o custo de obter cada literal do estado meta a partir do estado atual s, e então tomamos $h'(s)$ como uma combinação destas estimativas.

O custo de um literal é calculado por um procedimento similar àquele utilizado para encontrar caminhos mínimos em grafos. Consideramos os literais como vértices de um grafo e para cada ação ligamos os vértices que representam suas pré-condições aos vértices que representam seus efeitos positivos. Assim o tamanho de um caminho que leva dos vértices que representam o estado inicial até um vértice representando o literal l é o custo de se obter este literal a partir do estado inicial.

Definimos o custo de se obter um literal l a partir de um estado s como $g_s(l)$. Esta estimativa pode ser definida recursivamente por:

$$g_s(l) = \begin{cases} 0 & \text{se} \quad l \in s \\ \min_{a \in A(l)} \left[\, 1 + g_s(Pre(a)) \, \right] & \text{caso contrário,} \end{cases} \tag{2.1}$$

onde $\min[\emptyset] = \infty$, a é uma ação, $A(l)$ é o conjunto de ações que incluem o literal l, e $g_s(Pre(a))$, a ser definida a seguir, é a estimativa de custo de se obter as pré-condições

da ação a a partir de s.

Em termos práticos o cálculo de $g_s(l)$ definido por (2.1) é dado por $g_s(l) = 0$ se $l \in s$, ou $g_s(l) = \infty$ caso contrário. A cada vez que uma ação a é aplicada em s, cada literal positivo do efeito de a é incluido em s e o valor de $g_s(l)$ é atualizado para

$$g_s(l) = \min_{a \in A(l)} [\ g_s(l)\ ,\ 1 + g_s(Pre(a))\ ]. \tag{2.2}$$

Estas atualizações ocorrem até que os valores de $g_s(l)$ não se alterem. Podemos verificar que a atualização dada por (2.2) satisfaz a equação 2.1. O procedimento de atualização é polinomial no número de literais e de ações e corresponde a uma versão do algoritmo Bellman-Ford para caminhos mínimos em grafos [CLR90].

A expressão $g_s(Pre(a))$ presente em (2.1) e (2.2) é o custo estimado do conjunto de literais formado pelas pré-condições da ação a. Assim podemos definir o custo de um conjunto C de literais como a soma dos custos de cada literal $l \in C$:

$$g_s(C) = \sum_{l \in C} g_s(l). \tag{2.3}$$

Finalmente, a partir de (2.3), podemos definir a função heurística $h(s)$ que estima o custo de se obter o conjunto de literais M do estado meta a partir do estado s como:

$$h(s) \stackrel{def}{=} g_s(M). \tag{2.4}$$

A função heurística (2.4) usa a soma dos custos dos literais do estado meta, assumindo que estes literais são independentes. Em geral isso não é verdade pois a obtenção de um literal do estado meta pode facilitar ou dificultar a obtenção de outro, provocando assim uma interdependência entre eles que não é contemplada pela heurística. Assim esta função pode superestimar o custo de se obter todos os literais do estado meta a partir de um determinado estado, sendo portanto não admissível, permitindo que o procedimento de busca caia em mínimos locais. Entretanto, devido à sua simplicidade e à rapidez com que pode ser calculada, apresenta resultados práticos bastante satisfatórios.

2.2.2 O procedimento de busca

O *HSP* usa a função heurística (2.4) para guiar um procedimento de busca *subida de encosta* [RN95]. Este procedimento é bastante simples: a cada passo são calculados os custos dos filhos do nó atual da árvore de busca, um dos filhos de menor custo é então selecionado para a expansão no próximo passo e o mesmo processo é aplicado até que o estado meta seja alcançado.

Alguns incrementos a este procedimento de busca são necessários para um melhor desempenho do planificador, como a memorização dos nós já visitados para evitar repro-

cessamento e uma estratégia para reiniciar a busca em outro estado quando um mínimo local é alcançado [BG00].

2.2.3 Observações

Apesar do uso do procedimento subida de encosta com a função heurística apresentada não fornecer um procedimento completo para resolver o problema de planificação, o *HSP* foi o planificador de melhor desempenho na competição do AIPS-98 [McD98a] tanto em termos de tempo quanto em número de problemas resolvidos. No anexo B apresentamos resultados práticos de desempenho do *HSP* em alguns domínios usados nesta competição. Maiores detalhes sobre o HSP e suas variações podem ser encontradas em [BG00].

Outro ponto a ser considerado é a questão que o procedimento de busca do *HSP* não dá garantias sobre o tamanho do plano resultante, não sendo portanto um algoritmo ótimo em ralação ao tamanho do plano.

Até os trabalhos de Bonet e Geffner [BG98, BG99, BG00] a abordagem de busca heurística para planificação não apresentava resultados práticos relevantes, como por exemplo em [McD96].

2.3 Planificação como satisfabilidade

O uso de mecanismos genéricos para a solução de problemas, como procedimentos clássicos de prova e tradução para instâncias do problema de satisfabilidade (SAT), havia sido abandonado há muito tempo na área de planificação devido ao baixo desempenho destas técnicas. Atualmente, com o desenvolvimento de métodos rápidos para satisfabilidade proposicional [SLM92, SKC94], a afirmação de que a abordagem de prova baseada em SAT não é boa para problemas de planificação passou a ser contestada [Bib97, ETA97, ETA98].

Experimentos [KS96] mostraram que a conversão de problemas de planificação para instâncias SAT podem gerar planificadores com desempenho comparável aos melhores planificadores da época, como por exemplo *GRAPHPLAN* (ver seção 2.4).

Nesta seção serão apresentados o detalhamento da compilação para SAT e a arquitetura de um planificador baseado na transformação de um problema de planificação em um problema de satisfabilidade.

2.3.1 Compilando para SAT

Ao tratarmos planificação como satisfabilidade, um problema de planificação é um conjunto de axiomas com a propriedade de que qualquer valoração que torne este conjunto satisfatível corresponde a um plano válido. Alguns axiomas correspondem ao estado inicial e ao estado meta, e outros descrevem as ações.

A questão principal é como codificar um problema de planificação num problema de satisfabilidade, sendo que o primeiro tem uma componente temporal inerente, pois um plano é uma seqüência cronológica de ações, e o segundo é um problema estático de valoração. Uma alternativa para esta codificação é utilizar uma linguagem baseada em lógica de predicados com igualdade e sem funções ou quantificadores que seja capaz de representar a questão temporal. A seguir definimos uma linguagem com estas características.

As constantes desta linguagem são separadas em um conjunto finito de *tipos*. Cada tipo é um conjunto finito de indivíduos, identificados por letras maiúsculas. Um tipo especial chamado TEMPO tem seus elementos representados por números inteiros de um intervalo finito.

Um conjunto finito de fórmulas pode ser abreviado por um *esquema*. Um esquema pode ser visto como uma fórmula que pode conter variáveis ligadas aos quantificadores $\forall$ ou $\exists$. Ele representa o conjunto de todas as fórmulas que podem ser geradas pela iteração dos quantificadores sobre as constantes dos tipos apropriados. Usaremos letras minúsculas para as variáveis, sendo i e j reservadas para variáveis do tipo TEMPO. Expressões aritméticas do tipo $i + 1$ devem ser interpretadas como uma instanciação de tempo e não como uma função. Qualquer instanciação de um esquema que contenha um inteiro maior que N é descartada, onde N é a maior constante do tipo TEMPO.

Para exemplificar o processo de compilação de planificação para SAT usaremos o cenário do *Mundo de Blocos*. Neste cenário os problemas de planificação são dados por um estado inicial e um estado meta, que são formados por configurações de blocos sobre uma mesa, e uma ação que move blocos dada por:

Descrição	:	$\mathsf{mova}(x, y, z)$
Pré-Condição	:	$\mathsf{sobre}(x, y) \wedge \mathsf{livre}(x) \wedge \mathsf{livre}(z)$
Efeito	:	$\mathsf{sobre}(x, z) \wedge \mathsf{livre}(y) \wedge$
		$\neg\mathsf{livre}(z) \wedge \neg\mathsf{sobre}(x, y)$

Utilizando a linguagem acima temos o mundos dos blocos representado por dois tipos, BLOCOS e TEMPO. Os indivíduos do tipo BLOCO são $\{\mathsf{Mesa}, \mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{D}, \cdots\}$. Usaremos os predicados abaixo com os respectivos significados:

- $\mathsf{sobre}(x, y, i)$: x está sobre y no tempo i;
- $\mathsf{livre}(x, i)$: x está com sua superfície livre no tempo i;
- $\mathsf{mova}(x, y, z, i)$: x é movido de y para z entre os tempos i e $i + 1$.

Introduzimos uma constante especial chamada Mesa que nunca é movida e sempre está livre, ou seja, pode suportar qualquer quantidade de blocos.

Note que as ações devem ser representadas por proposições e não por funções ou termos. Portanto é necessário definirmos um tipo de axioma que represente o fato de que

se as pré-condições de uma ação são satisfeitas então a ação alcança seus efeitos. Por exemplo:

$$\begin{aligned}\forall x, y, z, i \qquad & \mathsf{sobre}(x, y, i) \wedge \mathsf{livre}(x, i) \wedge \mathsf{livre}(z, i) \wedge \\ & \mathsf{mova}(x, y, z, i) \rightarrow \mathsf{sobre}(x, z, i+1) \wedge \mathsf{livre}(y, i+1) \wedge \\ & \neg\mathsf{livre}(z, i+1) \wedge \neg\mathsf{sobre}(x, y, i+1)\end{aligned}$$

Considerando o estado inicial como o bloco A sobre o bloco B e o estado meta como B sobre A podemos representar o conjunto de planos com duas ações que resolvem o problema pelo esquema:

$$\begin{aligned}\exists x_1, y_1, z_1, x_2, y_2, z_2 \qquad & \mathsf{sobre}(\mathsf{A}, \mathsf{B}, 1) \wedge \mathsf{sobre}(\mathsf{B}, \mathsf{Mesa}, 1) \wedge \mathsf{livre}(\mathsf{A}, 1) \wedge \\ & \mathsf{mova}(x_1, y_1, z_1, 1) \wedge \mathsf{mova}(x_2, y_2, z_2, 2) \rightarrow \mathsf{sobre}(\mathsf{B}, \mathsf{A}, 3)\end{aligned}$$

Com esta representação basta encontrarmos uma instância deste esquema que seja consistente com os outros axiomas do problema e teremos um plano. Por exemplo, se a instância abaixo for consistente com o restante do conjunto de axiomas então uma solução para o problema foi encontrada:

$$\begin{gathered}\mathsf{sobre}(\mathsf{A}, \mathsf{B}, 1) \wedge \mathsf{sobre}(\mathsf{B}, \mathsf{Mesa}, 1) \wedge \mathsf{livre}(\mathsf{A}, 1) \wedge \\ \mathsf{mova}(\mathsf{A}, \mathsf{B}, \mathsf{Mesa}, 1) \wedge \mathsf{mova}(\mathsf{B}, \mathsf{Mesa}, \mathsf{A}, 2) \rightarrow \mathsf{sobre}(\mathsf{B}, \mathsf{A}, 3)\end{gathered}$$

Os dois exemplos de esquemas acima definem conjuntos de axiomas que são gerados por um compilador formando uma única fórmula lógica que é passada para um resolvedor. O compilador e o resolvedor serão apresentados na seção 2.3.2. A seguir serão descritos os axiomas necessários para a representação completa de um problema de planificação.

Para representarmos o estado inicial usamos um axioma do tipo:

$$\mathsf{sobre}(\mathsf{A}, \mathsf{B}, 1) \wedge \mathsf{sobre}(\mathsf{B}, \mathsf{Mesa}, 1) \wedge \mathsf{livre}(\mathsf{A}, 1) \wedge \mathsf{sobre}(\mathsf{B}, \mathsf{A}, 3) \wedge \cdots$$

onde $\cdots$ são todas as proposições consideradas falsas no estado inicial (Hipótese do Mundo Fechado).

De modo análogo o estado meta é representado por:

$$\mathsf{sobre}(\mathsf{B}, \mathsf{A}, N)$$

onde N é o tamanho do plano, proposto pelo compilador.

A representação das ações que será utilizada é exemplificada por:

$$\forall x, y, z, i \qquad \mathsf{mova}(x, y, z, i) \rightarrow \mathsf{sobre}(x, y, i) \wedge \mathsf{livre}(x, i) \wedge \mathsf{livre}(z, i) \wedge$$

$$\mathsf{sobre}(x, z, i+1) \wedge \mathsf{livre}(y, i+1) \wedge \neg\mathsf{livre}(z, i+1) \wedge$$
$$\neg\mathsf{sobre}(x, y, i+1)$$

que trata as pré-condições e os efeitos de uma ação de maneira simétrica, onde a ação implica tanto suas pré-condições quanto seus efeitos.

Além do esquema usado para representar a ação, temos também que representar todas as proposições que não são afetadas por esta ação, ou seja, precisamos tratar o *Frame Problem* [MH69]. Esta representação é feita pela inclusão de axiomas de quadro (*frame axioms*). Considerando a ação $\mathsf{mova}(x, y, z, i)$ os esquemas para os axiomas de quadro correspondentes são:

$$\forall x, y, z, w, i \qquad (w \neq x) \wedge (w \neq y) \wedge (w \neq z) \wedge$$
$$\mathsf{livre}(w, i) \wedge \mathsf{mova}(x, y, z, i) \rightarrow \mathsf{livre}(w, i+1)$$

$$\forall x, y, z, w, v, i \qquad (w \neq x) \wedge (w \neq y) \wedge (w \neq z) \wedge$$
$$(v \neq x) \wedge (v \neq y) \wedge (v \neq z) \wedge$$
$$\mathsf{sobre}(w, v, i) \wedge \mathsf{mova}(x, y, z, i) \rightarrow \mathsf{sobre}(w, v, i+1)$$

isto é, mover o bloco A para o bloco B não altera os blocos C, D e E, por exemplo.

Outro fato importante é que apenas uma ação deve ocorrer em um determinado instante de tempo:

$$\forall x_1, y_1, z_1, x_2, y_2, z_2, i \qquad (x_1 \neq x_2) \vee (y_1 \neq y_2) \vee (z_1 \neq z_2) \rightarrow$$
$$\neg\mathsf{mova}(x_1, y_1, z_1, i) \vee \neg\mathsf{mova}(x_2, y_2, z_2, i)$$

Finalmente temos que garantir que pelo menos uma ação ocorre em cada instante de tempo:

$$\forall i < N \quad \exists x, y, z \quad \mathsf{mova}(x, y, z, i)$$

Com os axiomas que definem o estado inicial e o estado meta; o esquema que representa as ações; os esquemas dos axiomas de quadro; e as garantias de que apenas uma e pelo menos uma ação ocorre em um instante de tempo, temos uma representação completa para um problema de planificação.

A conjunção da disjunção das instâncias dos axiomas anteriores constitui a fórmula lógica que representa o problema de planificação como uma instância do problema de satisfabilidade.

A partir da representação gerada pela compilação para SAT, um processo de simplificação é aplicado à fórmula final obtida, com o objetivo de reduzi-la. Após esta redução a

fórmula é passada como entrada para um algoritmo que resolva o problema de satisfabilidade. O planificador *SATPLAN* [KS92] utiliza este mecanismo e é baseado na arquitetura da figura 2.5.

2.3.2 Arquitetura do planificador

A construção de um planificador baseado na tradução do problema de planificação original para um problema SAT pode ser dada pela arquitetura da figura 2.5. O *compilador* recebe um problema de planificação como entrada, propõe um tamanho (número de ações) para o plano e gera como saída uma fórmula lógica proposicional na forma normal conjuntiva (*CNF*), que se satisfeita indica a existência de um plano do tamanho proposto.

Para a verificação da satisfabilidade usam-se dois procedimentos: o *simplificador* e o *resolvedor*. O *simplificador* usa técnicas rápidas (tempo linear) para reduzir o tamanho da fórmula. O *resolvedor* busca, por uma atribuição de valores aos literais, a satisfabilidade da fórmula. Se a fórmula é satisfatível, esta valoração é traduzida pelo *decodificador*, usando a tabela de símbolos do compilador, para um plano solução. Se o resolvedor verifica que a fórmula não é satisfatível o compilador reinicia o processo gerando uma nova codificação refletindo um plano de tamanho maior. Este processo é realizado até que uma solução seja encontrada.

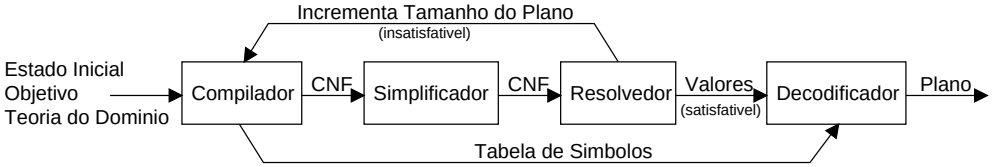

Figura 2.5: Arquitetura típica de um planificador baseado em SAT.

Vale ressaltar que o problema a ser resolvido não é encontrar um plano simplesmente, mas encontrar um plano com tamanho menor ou igual a um determinado valor. Isto implica que a arquitetura apresentada não consegue verificar a existência de um plano qualquer para o problema, ficando em aberta a questão de quando parar o procedimento se um plano não for encontrado.

2.3.3 Observações

A abordagem que transforma um problema de planificação em um dos mais conhecidos problemas da computação, o SAT, permite tirar proveito dos métodos rápidos recentes para resolver este último, resultando em planificadores de desempenho comparável aos melhores planificadores atuais.

A grande questão presente nesta estratégia é o tamanho da instância SAT resultante da compilação, indicando que a simplificação desta instância é essencial para um bom

desempenho do planificador. Esta simplificação pode ser vista como uma fase de pré-processamento para o planificador e é atualmente um ponto bastante explorado por pesquisas na área. Na seção 2.4.4 apresentamos um método de simplificação baseado na estrutura de outro planificador, o *GRAPHPLAN*.

2.4 Planificação usando grafo de planos

De 1971 até 1994 os algoritmos para planificação baseados em *STRIPS* podiam ser divididos basicamente em três classes: os derivados diretamente de *STRIPS*, que usavam procedimentos de busca exaustiva e invariavelmente caindo em explosões combinatoriais (seção 2.1.1); os procedimentos heurísticos, com o objetivo de minimizar o espaço de busca (seção 2.2); e os planificadores baseados em procedimentos de prova ou SAT (seção 2.3).

Depois de duas décadas de otimizações e incrementos sobre a mesma estrutura, em 1995 surge uma nova abordagem para o problema de planificação: uma baseada em um grafo de planos, que será descrita a seguir.

Desenvolvido por Blum e Furst [BF95, BF97], o planificador *GRAPHPLAN* foi um marco na área, pois com um algoritmo simples baseado num grafo conseguiu-se um planificador extremamente mais rápido que sistemas anteriores.

Sua simplicidade vem da idéia de dividir o processo em duas fases: expansão do grafo de planos e extração da solução. A cada passo o algoritmo executa estas duas fases, sucessivamente, até que uma solução seja alcançada.

Antes de entrarmos em detalhes dos procedimentos envolvidos nas duas fases do algoritmo necessitamos de uma definição precisa do grafo de planos, que é a estrutura base do algoritmo.

2.4.1 Grafo de planos

O *grafo de planos* é composto por dois tipos de nós, nós proposições e nós ações, que são distribuídos em camadas. As camadas de número par (incluindo zero) contém *nós proposições*, sendo que a camada zero contém todos os literais positivos do estado inicial do problema. Nas camadas ímpares ficam os *nós ações*, que são as instâncias de ações cujas pré-condições são satisfeitas pelas proposições do nível anterior (camada par). As arestas do grafo conectam os nós proposições aos nós ações da camada posterior, ligando as pré-condições das ações às proposições correspondentes. Da mesma forma, as arestas ligam efeitos de ações de uma camada às proposições correspondentes da camada seguinte.

A figura 2.6 mostra um grafo de planos onde os círculos representam os nós proposições e os quadrados os nós ações. Além das arestas que ligam nós de tipos diferentes (linhas sólidas) temos também arestas de manutenção (linhas pontilhadas), que conectam dois nós proposições. Estas arestas são chamadas de *ações de manutenção* e têm como

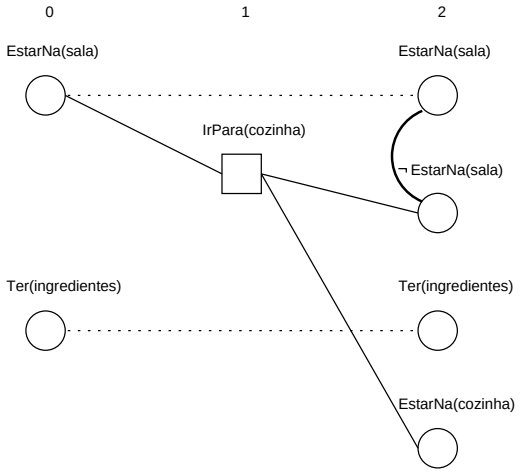

Figura 2.6: Representação de um grafo de planos de três camadas para o problema do bolo.

objetivo manter numa camada x as proposições de uma camada $x-2$. Pode-se imaginar uma ação cujo efeito é a própria pré-condição.

As camadas de proposição do grafo contêm os estados do mundo. Assim a primeira camada do grafo contém o estado inicial e a última o estado meta. As camadas de nós ação representam as ações que transformam um estado do mundo em outro.

2.4.2 Fase 1: expansão do grafo

A fase de expansão do grafo de planos consiste em se adicionar ao grafo uma camada de nós com instâncias de ações que tenham suas pré-condições atendidas na camada anterior, incluindo ações de manutenção. Portanto, quando o procedimento inicia, o grafo inicial a ser expandido consiste de apenas uma camada de proposições que representam o estado inicial do problema. Na primeira expansão as ações cujas pré-condições são atendidas pelo estado inicial são incluídas no grafo, que passa a ter duas camadas, uma de proposições e outra de ações. Após a camada de ações, uma nova camada de proposições é incluída, contendo os literais efeito das ações da camada anterior.

Durante o processo de expansão podem aparecer inconsistências e conflitos, pois nem todas as ações numa camada são viáveis devido a contradições entre suas pré-condições e efeitos. Por exemplo, duas ações podem ter efeitos contraditórios como P e $\neg P$. Com o objetivo de controlar estes conflitos definimos a seguinte relação de exclusão mútua (*mutex*) entre ações (representado graficamente por uma linha grossa):

Definição: Duas instâncias de ações numa camada i são mutuamente exclusivas se:

- um dos efeito de uma ação é a negação de um dos efeito da outra, como por exemplo na Figura 2.7. Neste caso dizemos que os efeitos são *efeitos inconsistentes*;

- um efeito de uma ação é a negação de uma pré-condição de outra, como na Figura 2.8. Dizemos que há *interferência*;
- as ações têm pré-condições que são mutuamente exclusivas na camada $i-1$, como na Figura 2.9. Este caso é chamado de *competição de necessidades*.

Além da relação de mutex para ações precisamos defini-la para proposições:

Definição: Duas proposições numa camada i são mutuamente exclusivas se:

- uma é a negação da outra (Q e $\neg\mathsf{Q}$ na Figura 2.9), ou
- todas as maneiras de obter as proposições são mutuamente exclusivas entre si, chamado *suporte inconsistente*. Ou seja, as ações da camada $i-1$ que têm como efeito estas proposições são duas a duas mutuamente exclusivas (Figura 2.10).

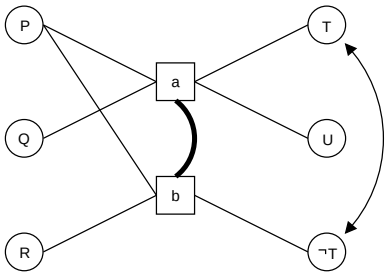

Figura 2.7: A ação b tem como efeito a proposição $\neg T$ que é a negação de um efeito da ação a, portanto as ações a e b são mutuamente exclusivas, representado pelo arco escuro.

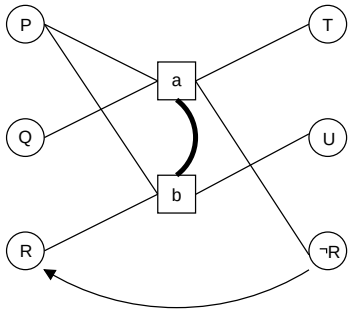

Figura 2.8: A ação a tem como efeito a proposição $\neg R$ que é a negação de uma pré-condição da ação b, portanto as ações a e b são mutuamente exclusivas.

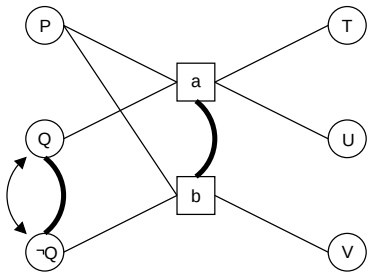

Figura 2.9: As ações a e b tem pré-condições que são mutuamente exclusivas na camada anterior Q e $\neg Q$, portanto também são mutuamente exclusivas.

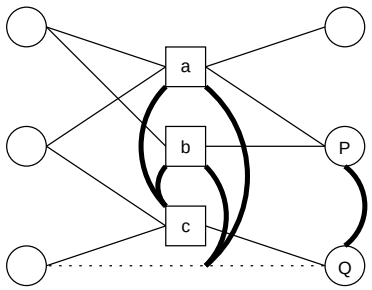

Figura 2.10: As proposições P e Q são mutuamente exclusivas devido às exclusões mútuas entre as ações que obtêm P (a e b) e as que obtêm Q (c e a ação de manutenção).

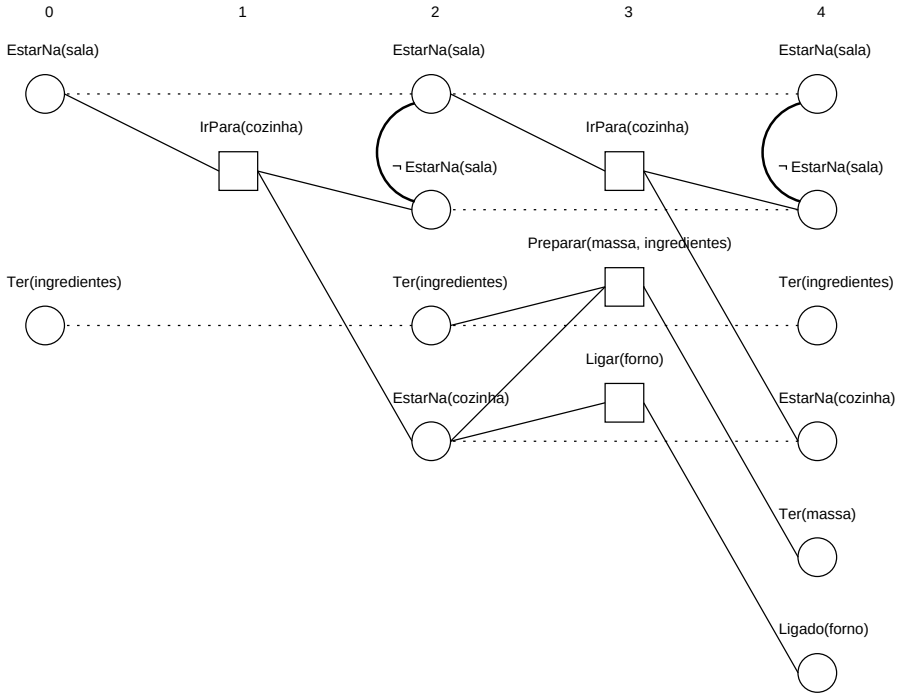

Figura 2.11: Camadas 0 a 4 do grafo de planos para o problema do bolo.

Observação: A definição de relação de exclusão mútua é recursiva, como no terceiro caso de exclusão entre ações: "as ações têm pré-condições que são mutuamente exclusivas na camada $i-1$". Portanto uma exclusão mútua numa determinada camada pode ter sido gerada por ações ou proposições de uma camada anterior.

As figuras 2.11, 2.12, 2.13, 2.14 e 2.15 mostram as expansões do grafo de planos da figura 2.6 para o problema do bolo apresentado suas respectivas exclusões mútuas.

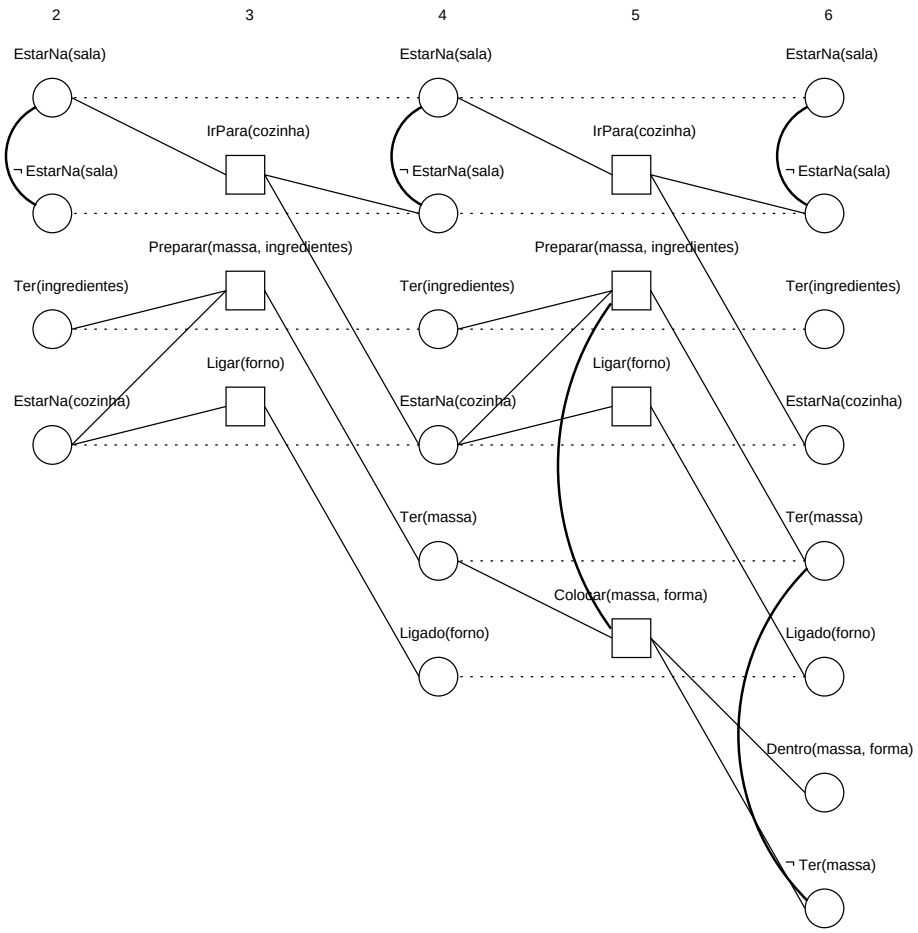

Figura 2.12: Camadas 2 a 6 do grafo de planos para o problema do bolo.

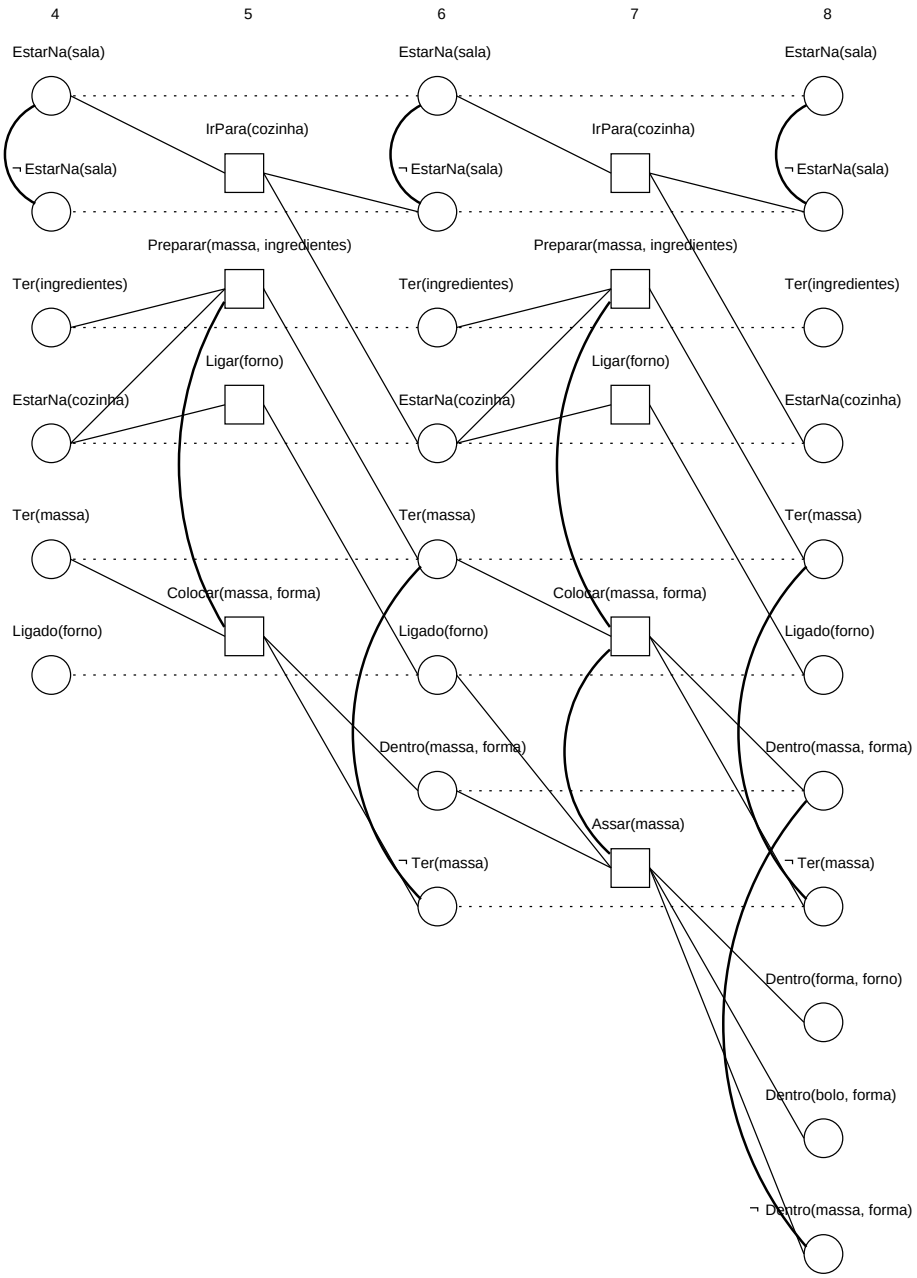

Figura 2.13: Camadas 4 a 8 do grafo de planos para o problema do bolo.

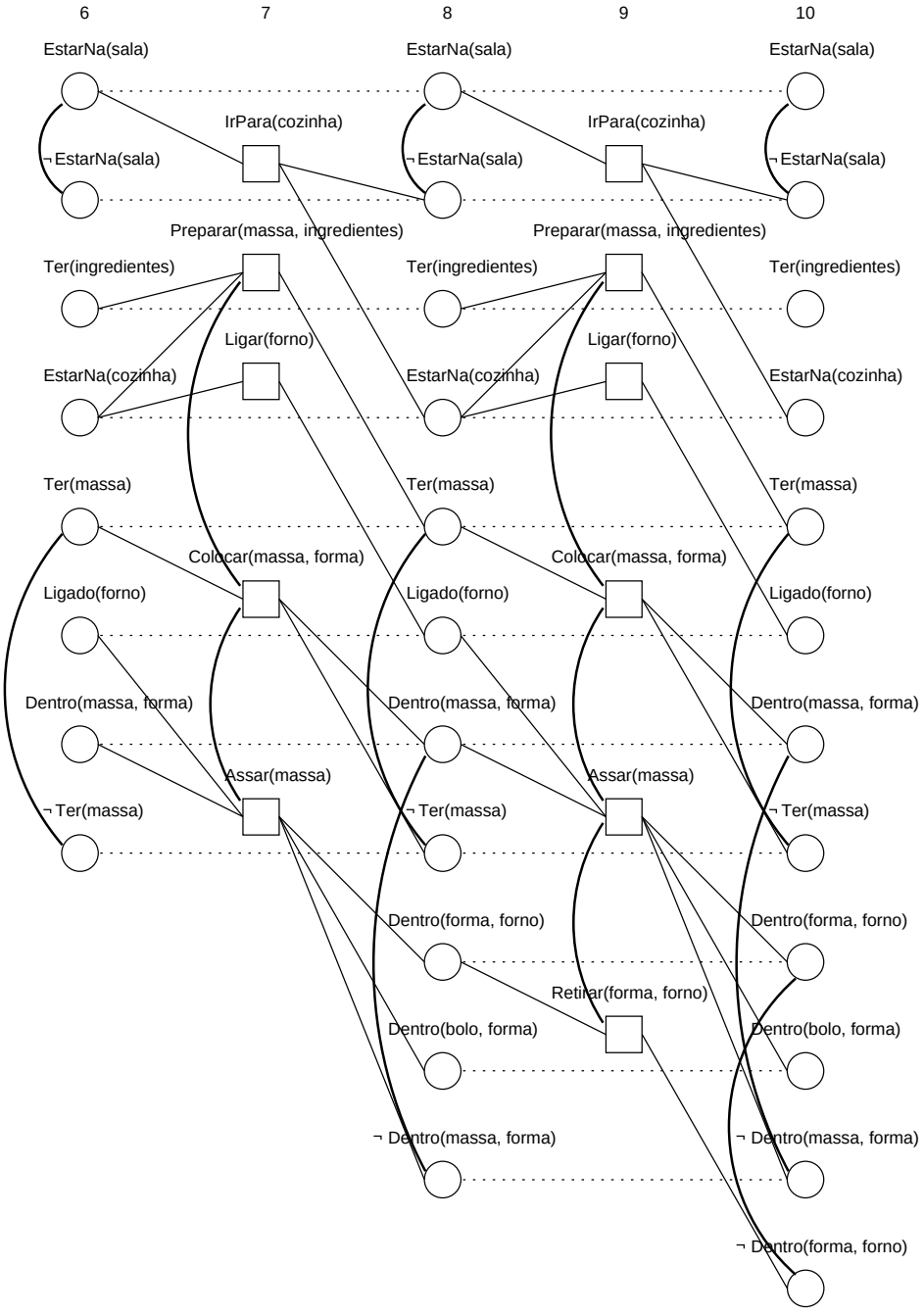

Figura 2.14: Camadas 6 a 10 do grafo de planos para o problema do bolo.

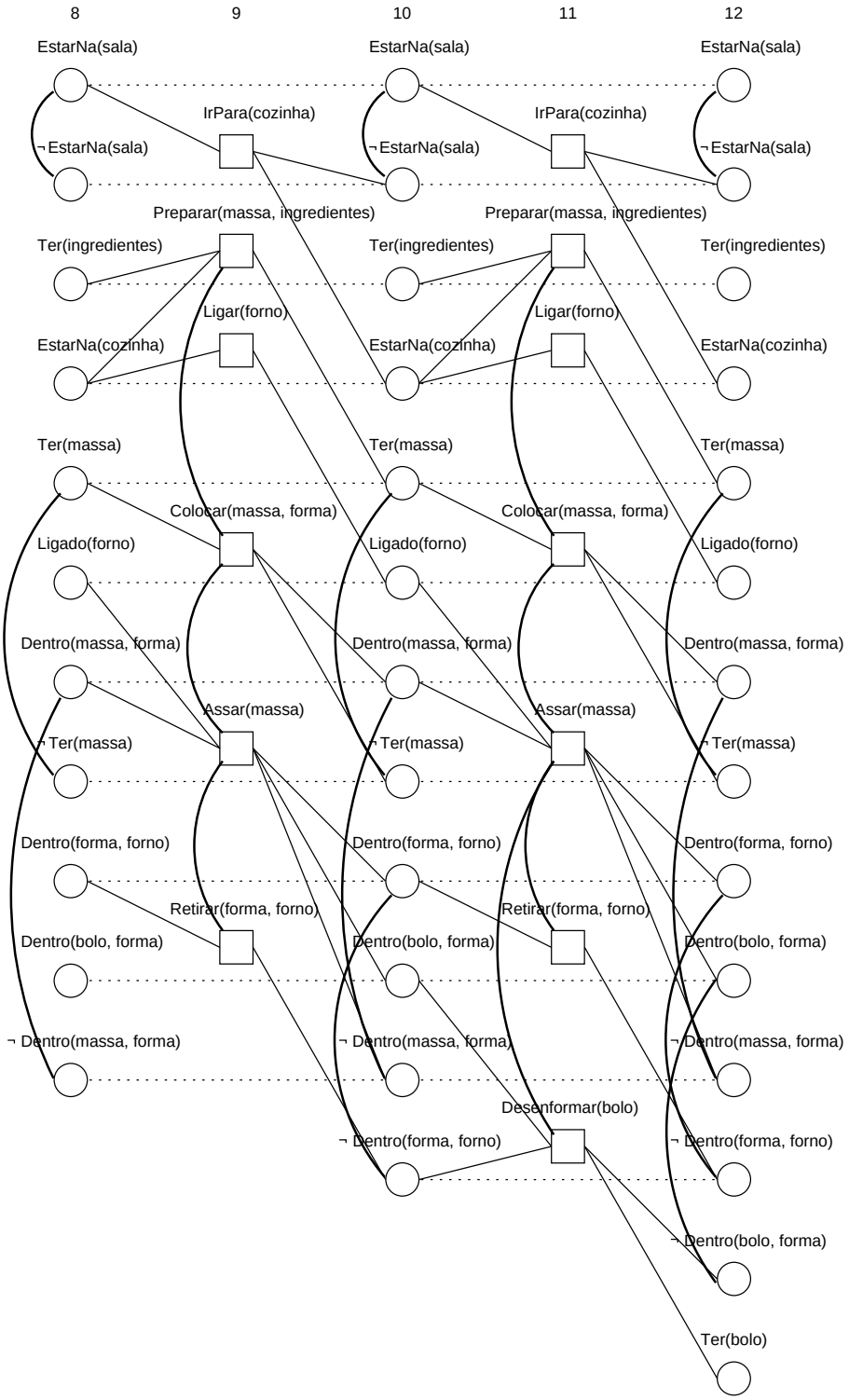

Figura 2.15: Camadas 8 a 12 do grafo de planos para o problema do bolo.

2.4.3 Fase 2: extração da solução

A fase de extração da solução é aplicada quanto, ao final da fase 1, encontramos todos os literais do estado meta (*sub-metas*) na última camada do grafo, como na Figura 2.15.

Esta condição indica que é possível que exista uma solução para o problema. Caso uma solução não seja encontrada, uma nova expansão do grafo é necessária a fim de obtermos todas as sub-metas sem conflitos internos.

O objetivo da fase de extração da solução é verificar se a partir da última camada, que contém o estado meta, é possível obter um plano que seja solução do problema. O processo, usando uma estratégia de *backtracking*, consiste em pegar cada sub-meta na camada i e encontrar uma ação a na camada $i-1$ que a tenha como efeito. Se a é *consistente* (não é *mutex*) com as outras ações escolhidas anteriormente então passamos para a próxima sub-meta. Caso contrário voltamos e escolhemos outra ação (*backtrack*).

Após encontrarmos um conjunto de ações consistente para a camada $i-1$, usamos o mesmo processo, recursivamente, para a camada $i-2$, considerando como sub-metas as pré-condições das ações escolhidas na camada $i-1$. Quando a camada zero é alcançada temos uma solução para o problema, que é a união dos conjuntos de ações escolhidas. Caso o *backtracking* falhe, esgotando as possibilidades, é necessária uma nova expansão do grafo, pois uma solução pode ainda existir.

Caso as duas últimas camadas ímpar do grafo sejam idênticas não existe solução para o problema e nenhum plano poderá ser encontrado para este problema: a última expansão não conseguiu eliminar nenhum *mutex*.

Quando uma solução é encontrada temos um plano parcialmente ordenado. As restrições de ordenação entre as ações são dadas pela ordenação dos conjuntos de ações obtidos de cada camada ímpar do grafo, ou seja, uma ação da camada i deve ser executada depois de todas as ações da camada $i-2$.

Podemos verificar que o algoritmo do *GRAPHPLAN* reduz bastante o espaço de busca em relação aos procedimentos de planificação clássicos; por outro lado utiliza um método exaustivo na fase de extração da solução, que traz os mesmos problemas daqueles procedimentos.

Otimizações no processo de busca do *GRAPHPLAN* e extensões à capacidade de representação da linguagem são temas atuais de pesquisa na área [KNHD97, KS99, LM99].

2.4.4 *GRAPHPLAN* como pré-processamento

Pesquisas recentes apontam a importância do pré-processamento em problemas de planificação [KW99]. Neste sentido, a construção do grafo de planos pode ser vista como uma fase de pré-processamento para um procedimento de busca, pois reduz significativamente o espaço de busca. Assim esta fase pode ser aproveitada por outros planificadores para melhorar o seu desempenho.

O planificador *BLACKBOX* [KS99] baseado na técnica de compilação para instâncias SAT (seção 2.3) utiliza o grafo de planos como um simplificador para a instância SAT. Esta instância não é mais gerada a partir da definição do problema, mas a partir de um grafo de planos. Caso a instância SAT não seja satisfatível uma nova expansão no grafo é realizada e uma nova instância é gerada.

Este planificador agrega o melhor das duas abordagens: o poder de redução do espaço de busca do grafo de planos e o desempenho dos novos métodos para SAT. Ou seja, o *BLACKBOX* substitui o procedimento de busca usado na extração da solução do *GRAPHPLAN* pela tradução do grafo para um instância SAT.

Neste contexto, o algoritmo que propomos no capítulo 3 também usa o *GRAPHPLAN* como uma fase de pré-processamento, trocando o procedimento de busca original por uma tradução do grafo de planos para um problema de alcançabilidade em redes de Petri.

2.4.5 Observações

O *GRAPHPLAN* foi um grande marco na área de planificação, pois apresenta uma abordagem alternativa às abordagens tradicionais para o problema. Sua grande contribuição é fornecer uma estrutura relativamente simples para reduzir o espaço de busca.

O planificadores baseados em sua estrutura, como IPP [KNHD97] e STAN [LM99], estão entre os mais rápidos atualmente. O seu uso como pré-processamento abre espaço para elaboração de novas abordagens para o problema de planificação.

2.5 Planificação por satisfação de restrições

Nesta seção apresentamos a abordagem apresentada por Beek e Chen [BC99] para o tratamento de problemas de planificação como problemas de satisfação de restrições. O planificador resultante desta abordagem, o *CPLAN*, apresenta um desempenho melhor que os atuais planificadores em alguns domínios específicos de problemas. Resultados práticos e maiores detalhes sobre a abordagem podem ser encontradas em [BC99].

2.5.1 Programação por restrições

Programação por restrições é uma metodologia para resolver problemas combinatoriais comumente usada em IA. Nesta abordagem os problemas são representados por conjuntos de restrições sobre seus componentes, como por exemplo restrições sobre tempo e estados do mundo. Resolver um problema usando esta técnica consiste em encontrar um conjunto de valores ou instâncias para os componentes do problema de modo que as restrições que o representam sejam atendidas.

Representar os problemas de planificação como problemas de satisfação de restrições (PSR[5]) é o ponto central da aplicação de programação por restrições na planificação.

Um PSR consiste de:

- um conjunto de n variáveis, $\{x_1, ..., x_n\}$;
- um domínio D_i de valores possíveis para cada variável x_i, $1 \leq i \leq n$;
- e uma coleção de m restrições, $\{C_1, ..., C_m\}$.

Cada restrição C_i, $1 \leq i \leq m$, é uma restrição sobre algum conjunto de variáveis, chamado de *esquema* da restrição. Uma solução para um PSR é uma atribuição de valores $a_i \in D_i$ para x_i, $1 \leq i \leq n$, que satisfaz todas as restrições C_j, $1 \leq j \leq m$.

Considere como exemplo o PSR com as variáveis x, y e z, todas sobre o domínio $\{1, 2, 3, 4\}$ e com as seguintes restrições:

$$\begin{aligned} C_1 : &\quad (x \leq 3) \rightarrow (x \geq 3), \\ C_2 : &\quad y + z \leq 6, \\ C_3 : &\quad \mathsf{diferentes}(x, y, z). \end{aligned}$$

Podemos verificar que as restrições contém operadores lógicos, aritméticos e abstratos como o caso da restrição C_3, isso nos dá um grande poder de representação para a modelagem de problemas. Cada restrição pode ser vista como uma função que retorna verdadeiro ou falso. O problema é que este poder de representação torna complexo o procedimento que encontra uma solução para o problema, caindo invariavelmente num algoritmo de busca por *backtracking* sobre a combinação dos valores possíveis para as variáveis. O número de variáveis e seus possíveis valores definem o tamanho do espaço de busca e as restrições definem como este espaço de busca pode ser reduzido. Para o exemplo apresentado, após uma busca no espaço de valorações possíveis, temos como solução $x = 1$, $y = 4$ e $z = 2$.

2.5.2 Modelagem

Considerando a busca por um plano no espaço de estados do mundo, a modelagem do problema de planificação como um PSR é dada por: cada estado do mundo é representado por um conjunto de variáveis e as restrições do PSR asseguram as transições válidas entre estes estados.

Para detalhar esta modelagem vamos utilizar como exemplo o domínio *logistics world*, que trata de problemas de logística envolvendo o transporte de pacotes entre localidades em cidades, utilizando caminhões e aviões. Desta forma cada estado S_t é dado pelas

[5] *Constraint Satisfaction Problem (CSP).*

variáveis $P_{i,t}$, $C_{j,t}$ e $A_{k,t}$, onde i, j e k identificam os pacotes P, os caminhões C e os aviões A respectivamente, e t é um índice sobre o número de passos do plano.

O domínio das variáveis $P_{i,t}$ é o conjunto que contém as localidades, os caminhões e os aviões, pois um pacote pode estar em uma localidade, ou em um caminhão, ou ainda em um avião. O domínio das variáveis $C_{j,t}$ e $A_{k,t}$ é o conjunto que contém as localidades.

Uma questão importante sobre as variáveis é especificar quais delas serão valoradas prioritariamente durante o processo de busca. Considerando o exemplo, as variáveis C e A são secundárias nos problemas de planificação para este domínio. Assim a busca pela solução será orientada pelas variáveis P.

As restrições do PSR são obtidas a partir do relacionamento das variáveis com as definições das ações do domínio, e definem como ocorrem as mudanças sobre as variáveis de um estado S_t para um estado S_{t+1}. Podemos classificar as restrições para um domínio em sete categorias:

Restrições de Ações modelam os efeitos das ações, por exemplo, uma variável de pacote só pode mudar de uma localidade no estado S_t para um caminhão ou avião no estado S_{t+1} se este avião ou caminhão estiver na mesma localidade que os pacotes nestes estados.

Restrições de Estado asseguram a consistência entre as variáveis de um estado.

Restrições de Distância definem os limites superior e inferior para número de passos necessários para a mudança de uma variável.

Restrições de Valores Simétricos permitem a troca de valores entre variáveis compatíveis, por exemplo, podemos trocar aviões que transportam pacotes entre duas localidades sem afetar o transporte destes pacotes.

Restrições de Escolha de Ações definem quais ações podem ser aplicadas em cada estado. Se um pacote está em uma localidade que é um aeroporto então um avião pode transportá-lo para outro aeroporto.

Restrições de Capacidade definem limites para os recursos disponíveis, este tipo de restrição não se aplica ao exemplo pois não existe um limite de carga para caminhões e aviões.

Restrições de Domínio definem restrições sobre os domínios das variáveis, por exemplo, um pacote que deve ser levado de uma localidade para outro em uma mesma cidade terá seu domínio restrito a localidades e caminhões, excluindo aviões.

As duas primeiras categorias são essenciais pois modelam como as ações alteram os estados e a consistência destes estados, as outras cinco categorias tem o objetivo de reduzir o espaço de busca melhorando o desempenho do processo de busca pelo plano.

Uma instância de um problema de planificação usando esta modelagem é data por um estado inicial, um estado final e limites inferior e superior para o tamanho do plano. Uma solução (um plano) para esta instância pode ser encontrada usando a modelagem do problema num PSR com um número de estados igual ao limite inferior fornecido, instanciando as variáveis dos estados inicial e final de acordo com o problema e, em seguida, por uma busca no espaço de estados. Caso nenhuma solução seja encontrada o número de estados do PSR é aumentado de um e uma nova busca é realizada, até que o limite superior seja alcançado ou uma solução seja encontrada.

2.5.3 Observações

O tratamento do problema de planificação como um problema clássico de satisfação de restrições está diretamente ligado a abordagem baseada em SAT apresentada na seção 2.3, pois podemos tratar cada axioma presente na instância SAT como uma restrição binária, e portanto a instância SAT como uma instância de um problema de satisfação de restrições.

A relevância do trabalho apresentado nesta seção é propor uma modelagem alternativa para o problema de planificação, classificando as restrições usadas na modelagem. O grande problema se resume em como obter as restrições a partir da definição do domínio do problema, pois nem todas podem ser obtidas facilmente ou de forma automática. Os autores não apresentam um método automático para este processo, dependendo de intervenção humana na especificação das restrições que representam as características do domínio.

Esta abordagem está diretamente ligada à apresentada na seção 2.6, onde as restrições são sobre variáveis inteiras.

2.6 Planificação por programação inteira

Em 1999 surge uma nova abordagem para o problema de planificação que liga a Planificação em Inteligência Artificial à área de Pesquisa Operacional, tirando proveito das técnicas desta na solução de problemas de otimização baseados em Programação Inteira (PI) e Programação Inteira Mista (PIM). Os trabalhos de Kautz e Walser [KW99] e de Vossen e colegas [VBLN99, VBLN00] apresentam traduções de problemas de planificação para problemas de programação inteira mista e resultados práticos relevantes quando comparados aos planificadores atuais.

Nas próximas seções apresentamos uma breve introdução a PIM e um resumo do processo de conversão dos problemas de planificação para problemas de PIM.

2.6.1 Programação inteira mista

Um *problema de programação inteira mista* pode ser dado por uma função sobre um conjunto de variáveis

$$M(x) = cx, \tag{2.5}$$

sujeita à restrição sobre o mesmo conjunto de variáveis

$$Ax \leq b, \tag{2.6}$$

com as variáveis divididas em dois grupos

$$x_1, ..., x_p \in Z^+ \tag{2.7}$$

$$x_{p+1}, ..., x_n \in R^+ \tag{2.8}$$

onde M é a função objetivo a ser minimizada no problema de otimização correspondente, c é um vetor linha de dimensão n, x é um vetor coluna de dimensão n, A é uma matriz $(m \times n)$, b é um vetor coluna de dimensão m, e p é o número de variáveis inteiras.

Um problema de programação inteira mista é o problema de se encontrar um vetor x^* que obedece as restrições impostas por (2.6), (2.7) e (2.8), e minimiza a função (2.5). Quando todas as variáveis são inteiras, ou seja, quando $p = n$ dizemos que o problema é um *problema de programação inteira*. Se $x_1, ..., x_p \in \{0, 1\}$ temos um *problema de programação inteira mista 0/1* (PIM-0/1). Estes problemas são resolvidos utilizando algoritmos de "ramificar e limitar[6]" partindo de soluções parciais obtidas através de técnicas de programação linear. A descrição de métodos para resolver problemas de programação inteira, mista ou 0-1 estão fora do escopo deste trabalho, informações detalhadas podem ser encontradas em [Wol98].

2.6.2 Formulando planificação como PI

A formulação do problema de planificação como um problema de PI apresentada nesta seção é derivada diretamente da tradução para SAT apresentada na seção 2.3.1, seguindo aquela apresentada em [VBLN00].

Considerando $\mathcal{A}$ como o conjunto de ações e $\mathcal{P}$ como o conjunto de proposições presentes no problema de planificação, definimos os seguintes conjuntos:

- $pre_p \subseteq \mathcal{A}$, para todo $p \in \mathcal{P}$, onde pre_p representa o conjunto de ações que tem a proposição p como uma pré-condição;
- $pos_p \subseteq \mathcal{A}$, para todo $p \in \mathcal{P}$, onde pos_p representa o conjunto de ações que tem a proposição p como um efeito positivo;

[6] *branch and bound*.

- $neg_p \subseteq \mathcal{A}$, para todo $p \in \mathcal{P}$, onde neg_p representa o conjunto de ações que tem a proposição p como um efeito negativo.

A modelagem em PI é dada em três partes: a definição das variáveis, a definição das restrições e a função objetivo.

As variáveis do problema são dadas por:

- Para todo $p \in \mathcal{P}$ e $i \in \{1, ..., t+1\}$, temos as variáveis de proposições definidas como
$$x_{p,i} = \begin{cases} 1 & \text{se a proposição } p \text{ é verdadeira no tempo } i, \\ 0 & \text{caso contrario.} \end{cases}$$

- Para todo $a \in \mathcal{A}$ e $i \in \{1, ..., t\}$, temos as variáveis de ações definidas como
$$y_{a,i} = \begin{cases} 1 & \text{se a ação } a \text{ é executada no tempo } i, \\ 0 & \text{caso contrario.} \end{cases}$$

As restrições são separadas em quatro classe diferentes:

- *Restrições dos estados inicial e final* - considerando $\mathcal{I}$ o conjunto de proposições do estado inicial e $\mathcal{F}$ o conjunto de restrições do estado final:
$$x_{p,1} = \begin{cases} 1 & \text{se } p \in \mathcal{I}, \\ 0 & \text{se } p \notin \mathcal{I}. \end{cases}$$
$$x_{p,t+1} = 1 \quad \text{se } p \in \mathcal{F}.$$

- *Restrições de pré-condições* - ações devem implicar em suas pré-condições:
$$y_{a,i} \leq x_{p,i} \quad \forall a \in pre_p, \quad i \in \{1, ..., t\}.$$

- *Restrições de "quadro"* - restrições que representam os "*frame axioms*", vinculando as proposições com ações que as têm como efeitos:
$$x_{p,i+1} \leq \sum_{a \in pos_p} y_{a,i} \quad \forall i \in \{1, ..., t\} \quad p \in \mathcal{F}.$$

- *Restrições de exclusão de conflito* - se uma ação tem como efeito a negação de uma pré-condição de outra ação elas estão em conflito:
$$y_{a,i} + y_{a',i} \leq 1$$
para todo $i \in \{1, ..., t\}$ e todo a, a' para os quais existe $p \in \mathcal{F}$ tal que $a \in neg_p$ e $a' \in (pre_p \cup pos_p)$.

A última parte da formulação do problema de PI é a definição da função objetivo a ser minimizada. Para qualquer função que se escolha, a solução para o problema de PI, ou seja a valoração das variáveis, será uma solução para o problema de planificação pois esta valoração atende as restrições definidas. Se considerarmos a função como a soma das variáveis de ações a solução do problema de PI resultará no plano com o menor número de ações, ou seja, o plano ótimo.

A formulação acima pode ser transformada em um problema de PIM 0-1 se relaxarmos a restrição de integralidade das variáveis de proposições, ou seja, trocar $x_{p,i} \in \{0, 1\}$ por $0 \leq x_{p,i} \leq 1$. Esta integralidade é garantida pela integralidade das variáveis de ações. Esta mudança torna a solução do problema mais simples e rápida pois as variáveis de proposições poderão ser ignoradas durante o processo de ramificar e limitar que busca valores inteiros para as variáveis.

2.6.3 Observações

Podemos observar que a formulação apresentada é uma aplicação direta da compilação para SAT descrita na seção 2.3.1. Percebe-se um mapeamento praticamente direto dos esquemas de axiomas em restrições de um problema de PI. Outras formulações equivalentes podem ser encontradas em [BD98, KW99, VBLN00]. No capítulo 3 apresentamos uma formulação alternativa para o problema de planificação como programação inteira.

A aplicação de técnicas de programação inteira em planificação em IA estabelece uma ligação entre duas áreas da computação, a de pesquisa operacional e a de IA, abrindo espaço para a integração de técnicas destas duas na solução de problemas de planificação.

2.7 Considerações finais

Neste capítulo foram analisados vários algoritmos para o problema de planificação baseados na representação proposicional *STRIPS*.

A primeira abordagem, que chamamos de clássica, é aquela que trata o problema como uma busca no espaço de estados do mundo ou no espaço de planos. Verificamos que neste caso a construção do plano pode ser realizada a partir do estado inicial do mundo, progressivamente, até que o estado meta seja alcançado, ou regressivamente, partindo do estado meta em direção ao estado inicial.

Os sistemas para problemas de planificação clássica baseada em *STRIPS* utilizam uma linguagem de representação bastante simples, permitindo tratar o problema como uma busca regressiva no espaço de estados. Outra característica relevante da linguagem *STRIPS* é a utilização da "Hipótese do Mundo Fechado" (*Closed World Assumption* [Rei78]) como uma alternativa para tratar o problema do quadro (*Frame Problem* [MH69]).

Os sistemas de planificação baseados em *STRIPS* apresentam uma grande limitação

quanto à capacidade de representação, mas por outro lado possibilitam o desenvolvimento de estratégias de busca bastante otimizadas. Já os planificadores que utilizam representações mais ricas apresentam um desempenho muito baixo quando comparados aos planificadores mais simples.

A partir da modelagem da planificação como um problema de busca, analisamos uma estratégia baseada em busca heurística, com o objetivo de melhorar o desempenho do planificador. Apesar de não oferecer um procedimento completo, este método se mostra bastante eficiente em termos práticos.

A utilização de funções heurísticas para o direcionamento do processo de busca possibilita o desenvolvimento de planificadores mais rápidos que aqueles baseados em busca exaustiva. O preço pago por este ganho de desempenho é relativamente caro pois estes procedimentos normalmente não são completos. Em casos práticos estes planificadores são muito eficientes, pois evitam a explosão combinatória dos métodos tradicionais de busca. Podemos citar o desempenho do *HSP* na competição do AIPS-98 [McD98a]. Outro problema dos planificadores heurísticos é que o plano gerado normalmente tem um tamanho maior que o plano ótimo para o problema, o que, dependendo da aplicação, pode ser altamente relevante.

Descrevemos uma abordagem que visa converter o problema de planificação em um dos problemas mais conhecidos da computação em geral, SAT, aproveitando assim inovações em termos de algoritmos nesta área.

Outra modelagem tratada é a do grafo de planos, que de maneira simples consegue reduzir o espaço de busca, tornando o algoritmo extremamente eficiente. Esta abordagem baseada em grafos é a maior inovação na área de planificação desde o sistema *STRIPS*.

O planificador *BLACKBOX* [KS99] utiliza o processo de expansão do grafo de planos do *GRAPHPLAN*, mas durante a extração da solução faz uma compilação do grafo para SAT. Este planificador usa o melhor das duas estratégias, pois o processo de expansão do grafo de planos é mais eficiente que o processo de simplificação, e a utilização dos novos métodos para SAT é mais eficiente que o processo de busca usado na extração da solução do *GRAPHPLAN*. Esta fusão nos dá um procedimento mais promissor para o tratamento de problemas reais de planificação, pois o grafo de planos é atualmente o melhor método de redução do espaço de busca, e a compilação para SAT possui melhor desempenho que os procedimentos clássicos de busca.

As duas abordagens baseadas em restrições, PSR e PI, são bastante parecidas na questão da modelagem do problema, sendo que a última apresenta uma vantagem sobre a anterior, fornecendo um método automático para a formulação de problemas de planificação como conjuntos de restrições.

A questão da intervenção humana para a formulação das restrições do PSR inviabiliza sua aplicação em soluções genéricas para o problema, mas é preciso reconhecer que o método PSR de [BC99] é mais rápido que qualquer outro planificador atual.

O grande problema da formulação por PI é estar baseada no mesmo esquema de tradução usado para SAT, que devido aos axiomas de quadro e outros, tornam o número de restrições muito grande, dificultando a busca pela solução.

No próxima capítulo apresentamos uma nova abordagem para o problema de planificação, traduzindo-o para um problema de alcançabilidade em uma Rede de Petri e usando PI para resolver este último. Caímos, assim, em um problema de PI que não tem a representação de axiomas do quadro. Isto, ao menos teoricamente, é uma vantagem [Sha97]. Nossa abordagem está diretamente ligada às formulações por restrição apresentadas.

Capítulo 3

Petriplan

Apresentamos neste capítulo o algoritmo Petriplan, que é uma nova abordagem para o problema de planificação usando redes de Petri como ferramenta de modelagem. Antes de entrarmos propriamente na descrição do algoritmo faremos uma breve introdução dos conceitos necessários ao seu entendimento.

3.1 Redes de Petri e alcançabilidade

Rede de Petri é uma ferramenta formal que é particularmente utilizada para representar paralelismo, concorrência, conflito e relações causais em sistemas dinâmicos de eventos discretos.

Nesta seção definimos uma rede de Petri e apresentamos a notação que será usada posteriormente. Seguindo [VPCG99], apresentaremos uma definição formal de redes de Petri como uma coleção de matrizes e vetores, cujos componentes são números naturais, possibilitando sua caracterização como programação inteira.

Uma rede de Petri é uma quádrupla:

$$N = (P, T, Pre, Pos),$$

onde:

$$\begin{aligned} P = \{p_1, p_2, \ldots, p_n\} &\quad \text{é um conjunto finito de lugares;} \\ T = \{t_1, t_2, \ldots, t_m\} &\quad \text{é um conjunto finito de transições;} \\ Pre : P \times T \to \mathbb{N} &\quad \text{é a função de incidência de entrada;} \\ Pos : P \times T \to \mathbb{N} &\quad \text{é a função de incidência de saída.} \end{aligned}$$

Uma rede de Petri com uma dada marcação inicial é denotada por (N, M_0), onde:

$$M_0 : P \to \mathbb{N} \quad \text{é a marcação inicial.}$$

Podemos visualizar uma rede de Petri usando uma representação gráfica, onde círculos representam lugares e barras verticais representam transições. Os pequenos círculos pretos no interior dos lugares representam a marcação da rede, chamados de marcas. Os arcos orientados representam as funções de incidência de entrada e saída de uma transição e os números relacionados representam os pesos destes arcos. Assim, na figura 3.1(a) temos três lugares (a, b e c), duas transições (x e y), cinco arcos, uma marca no lugar a e duas no lugar b.

Lugares são os nós descrevendo os estados (um lugar é uma representação parcial de um estado) e as transições são responsáveis pelas mudanças de estado, elas removem e inserem marcas dos lugares. Por exemplo, ao dispararmos a transição x da rede representada pela figura 3.1(a), uma marca é removida do lugar a e outra do lugar b, e três são inseridas no lugar c. Estas quantidades são dadas pelo valor dos arcos associados à transição x. A marcação resultante é dada pela figura 3.1(b). A definição formal do disparo de uma transição e das transformações ocorridas na marcação da rede serão apresentadas a seguir.

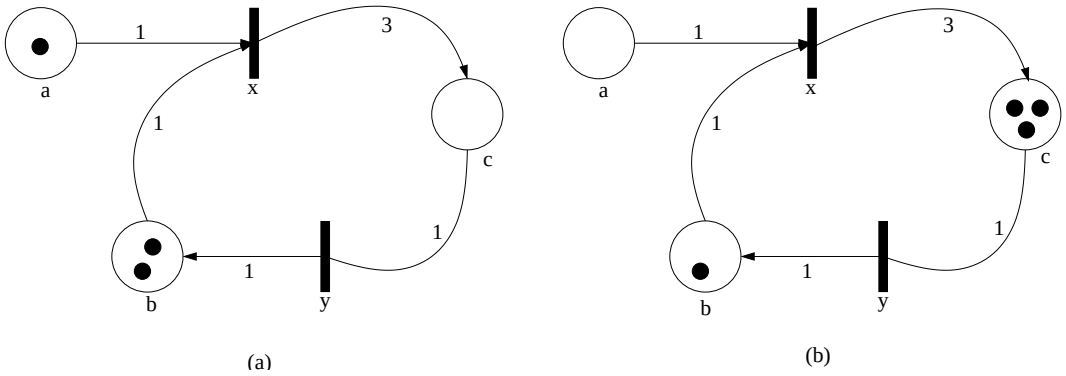

Figura 3.1: Representação gráfica de uma rede de Petri.

A função de incidência *Pre* descreve os arcos orientados que ligam lugares a transições. Ela representa, para cada transição t, o fragmento do estado em que o sistema deve estar antes da mudança de estado correspondente à ocorrência de t. $Pre(p,t)$ é o peso do arco (p,t). $Pre(p,t) = 0$ representa a ausência de um arco entre o lugar p e a transição t.

A função de incidência *Pos* descreve os arcos orientados que ligam transições a lugares. Ela representa, para cada transição t, o fragmento do estado em que o sistema deve estar após a a mudança de estado correspondente à ocorrência de t. $Pos(p,t)$ é o peso do arco (t,p). $Pos(p,t) = 0$ representa a ausência de um arco entre a transição t e o lugar p.

O vetor $Pre(.,t)$ denota os arcos de entrada da transição t com seus pesos. O vetor $Pos(.,t)$ denota os arcos de saída da transição t com seus pesos.

Na representação matricial de uma rede de Petri, *Pre* e *Pos* são matrizes de n linhas (os lugares) e m colunas (as transições) e seus elementos pertencem a $\mathbb{N}$.

A dinâmica da rede de Petri é dada pelo *disparo* das transições habilitadas, cuja ocorrência corresponde a uma mudança de estado do sistema modelado pela rede. Uma

transição t de uma rede de Petri N está *habilitada* para uma marcação M se e somente se:

$$M \geq Pre(., t).$$

Esta condição de habilitação, expressa sob a forma de uma desigualdade entre dois vetores, é equivalente a:

$$\forall\ p \in P,\ M(p) \geq Pre(p, t).$$

Na marcação da figura 3.1(a) apenas a transição x está habilitada e na figura 3.1(b) apenas a transição y. Apenas transições habilitadas podem ser disparadas. Se M é uma marcação de N habilitando uma transição t, e M' a marcação derivada pelo disparo de t a partir de M, então:

$$M' = M + Pos(., t) - Pre(., t).$$

Note que o disparo da transição t a partir da marcação M deriva a marcação M', representado por:

$$M' : M \xrightarrow{t} M'.$$

Podemos generalizar esta fórmula para calcular uma nova marcação após o disparo de uma seqüência s de transições. Vamos considerar a matriz C dada por:

$$C = Pos - Pre,$$

chamada *matriz de incidência*, e o vetor $\overline{s}$, chamado *vetor característico* do disparo de uma seqüência s, dado por $\overline{s} : T \rightarrow \mathbb{N}$, tal que $\overline{s}(t)$ é o número de vezes que a transição t aparece na seqüência s. O número de transições em T define a dimensão do vetor $\overline{s}$. Então, uma nova marcação M_g de uma marcação M, após o disparo da seqüência s de transições, é dada por:

$$M_g = M + C.\overline{s}. \tag{3.1}$$

Esta equação é chamada *equação fundamental* de N.

Podemos usar a equação fundamental para determinar um vetor $\overline{s}$ para uma dada rede N e duas marcações M e M_g. A solução que satisfaz a equação deve ser um vetor inteiro não-negativo e sua existência é apenas uma condição necessária para que M_g seja alcançada a partir de M. Esta condição se torna necessária e suficiente para *redes de Petri acíclicas*, uma sub-classe de redes de Petri que não possui ciclos dirigidos [Mur89].

A relação de alcançabilidade entre marcações de uma transição disparável pode ser estendida, por transitividade, para a alcançabilidade do disparo de uma seqüência de transições. Assim, em uma rede de Petri N, é dito que a marcação M_g é alcançável a partir da marcação M se e somente se existe uma seqüência de transições s tal que:

$$M \xrightarrow{s} M_g.$$

O *conjunto alcançabilidade* de uma rede de Petri marcada (N, M_0) é o conjunto $\mathcal{R}(N, M_0)$ tal que

$$(M \in \mathcal{R}(N, M_0)) \Leftrightarrow (\exists s M_0 \xrightarrow{s} M). \tag{3.2}$$

Chamamos de *problema de alcançabilidade* para redes de Petri o problema de encontrar uma seqüência disparável s para alcançar uma dada marcação M_g a partir de M_0. Tal seqüência existe se $M_g \in \mathcal{R}(N, M_0)$. O *problema de alcançabilidade de sub-marcação* consiste em encontrar, caso exista, uma seqüência disparável s para alcançar um subconjunto de lugares marcados $M_s \subseteq M_g$, onde $M_g \in \mathcal{R}(N, M_0)$. É sabido que os problemas de alcançabilidade citados são decidíveis [Mur89].

3.2 O algoritmo Petriplan

O Petriplan é um algoritmo de três fases que usa uma estrutura semelhante à dos algoritmos *GRAPHPLAN* e *BLACKBOX* apresentados nas seções 2.4 e 2.4.4.

A primeira fase consiste da construção do grafo de planos seguindo as mesmas regras apresentadas na seção 2.4. Esta fase pode ser vista como um pré-processamento, buscando simplificar o problema para a próxima fase.

Na segunda fase, o grafo de planos gerado é transformado em uma rede de Petri equivalente, onde o problema de planificação original pode ser visto como um problema de alcançabilidade de sub-marcação, apresentado na seção 3.1.

Na terceira fase do algoritmo o problema de alcançabilidade definido na fase anterior é convertido em um problema de programação inteira (seção 2.6.1) baseado nas equações 3.1 e 3.2, que então é resolvido por sistemas específicos para otimização de problemas de programação inteira.

Se nenhuma solução for encontrada para o problema de programação inteira na terceira fase, o algoritmo volta para a primeira fase e uma nova expansão do grafo é realizada (seção 2.4.2), este laço continua até que uma solução seja encontrada na fase 3 ou a condição de falha do *GRAPHPLAN* (seção 2.4.3) seja atendida na fase 1. Quando uma solução é encontrada na fase 3 ela é convertida para um plano que resolve o problema original de planificação.

Nas próximas seções apresentamos, usando um exemplo, o grafo de planos usado na fase 1, a tradução para redes de Petri da fase 2, a solução do problema de alcançabilidade na fase 3 usando programação inteira e a recuperação do plano a partir da solução do problema de programação inteira.

3.2.1 Grafo de planos

O grafo de planos usado no Petriplan é o mesmo apresentado na seção 2.4, isto é, é o grafo obtido pelo algoritmo *GRAPHPLAN*.

Para detalhar o funcionamento do algoritmo usaremos um exemplo apresentado em [Wel99]. O problema é preparar um jantar para um encontro, os objetivos são: remover o lixo, preparar o jantar e embrulhar um presente. As quatro ações possíveis são: *cook*, *wrap*, *carry* e *dolly*. A ação *cook* requer mãos limpas (*clean*) e obtém o jantar (*dinner*). A ação *wrap* tem que ser realizada em silêncio (*quiet*) e obtém o presente embrulhado (*present*). *Dolly* elimina o lixo (*garbage*) usando um carrinho de mão e produz barulho (negando *quiet*), *Carry* também elimina o lixo mas suja as mãos (negando *clean*).

No estado inicial o agente está com as mãos limpas, tem lixo na casa e está em silêncio, todas as outras proposições são falsas.

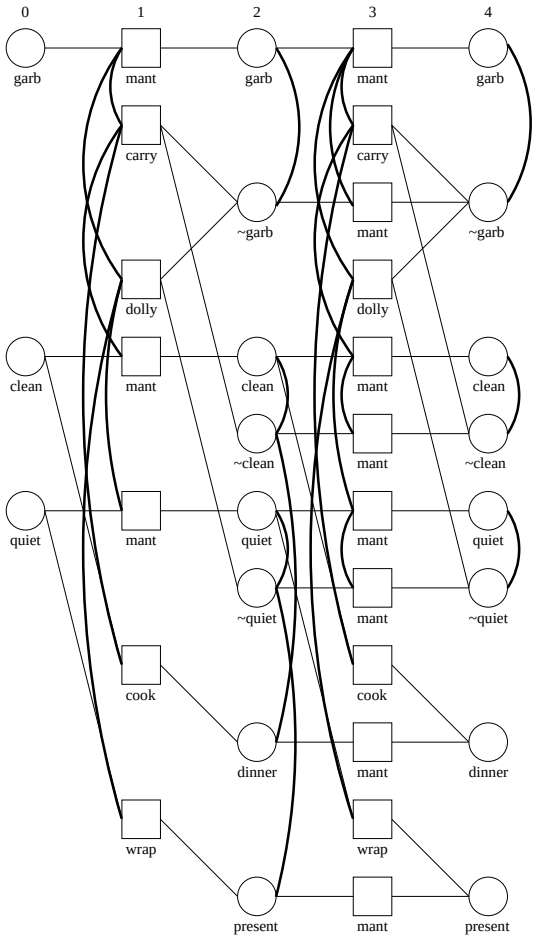

Figura 3.2: O grafo de planos para o problema do jantar.

A figura 3.2 mostra o grafo de planos para este problema. A única diferença na representação da seção 2.4 é que as ações de manutenção, antes representadas por linhas pontilhadas, agora são representadas como ações normais, com o objetivo de simplificar o processo de tradução apresentado na próxima seção.

A figura 3.3 mostra o grafo de planos para o problema do jantar com uma das soluções possíveis (um plano) marcada.

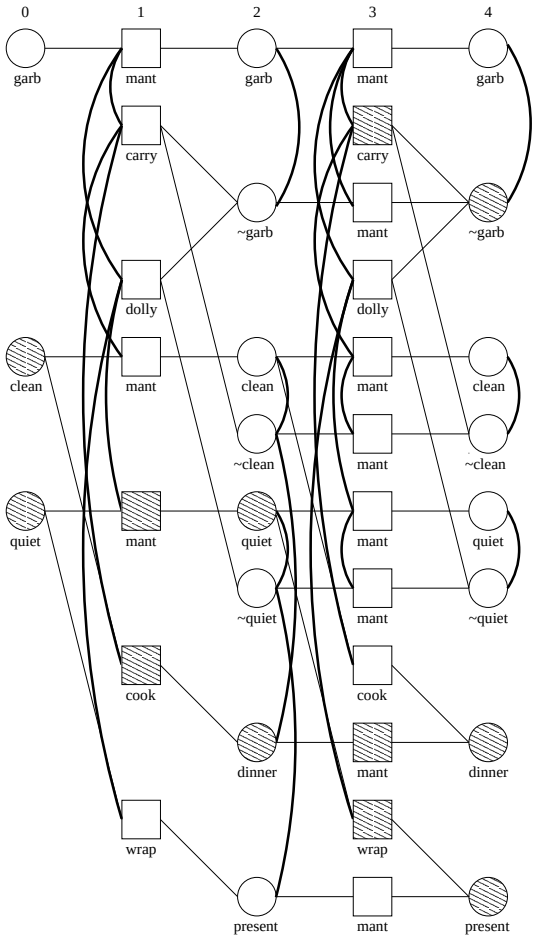

Figura 3.3: O grafo de planos para o problema do jantar com uma de suas soluções.

3.2.2 Tradução para rede de Petri

Nesta seção vamos mostrar como traduzir o grafo de planos obtido pela primeira fase do algoritmo *GRAPHPLAN* (seção 3.2.1) em uma rede de Petri acíclica.

Usando os conceitos apresentados na seção 3.1 vamos mostrar que o problema de encontrar um plano em um grafo de planos é equivalente ao problema de alcançabilidade de sub-marcação nesta rede de Petri.

Um grafo de planos pode ser fragmentado em cinco estruturas: nós ação, nós proposição, arestas de nós ação para nós proposição (*arestas efeito*), arestas de nós proposição para nós ação (*arestas pré-condição*) e relações de exclusão mútua.

Cada uma destas estruturas é traduzida diretamente em uma estrutura de redes de Petri equivalente. A seguir apresentamos estas traduções.

Nós ação: um nó ação do grafo é traduzido em uma única transição na rede de Petri, a figura 3.4 mostra à esquerda o nó do grafo e à direita a transição correspondente.

Os nós ação do grafo são traduzidos diretamente para uma transição da rede de Petri, que só estará habilitada para disparo se todas as suas pré-condições forem verdadeiras, ou

Figura 3.4: Tradução dos nós ação.

seja, se existirem marcas em todos os lugares que representam as arestas pré-condição ligadas à ação representada pela transição. Após o disparo de uma transição que representa uma ação, uma marca é removida de cada lugar representando uma aresta pré-condição e outra é inserida em cada lugar representando um proposição efeito.

Nós proposição: um nó proposição é traduzido em um lugar e uma transição, com um arco do lugar para a transição. A figura 3.5 mostra esta tradução.

Figura 3.5: Tradução dos nós proposição.

Os nós proposição são representados por um lugar ligado a uma transição. Esta transição é necessária para que todos os lugares que representam arestas pré-condição recebam marcas após o seu disparo. O disparo só ocorrerá se existir uma marca no lugar que representa a proposição, ou seja, a proposição é verdadeira.

Arestas efeito: uma aresta efeito é traduzida em um arco que vai da transição que representa o nó ação para o lugar representando o nó proposição. A figura 3.6 mostra esta tradução.

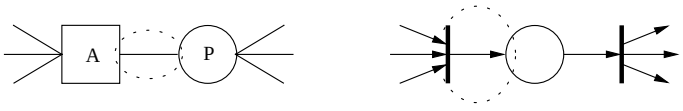

Figura 3.6: Tradução das arestas efeito.

Arestas pré-condição: uma aresta pré-condição é traduzida em um lugar com dois arcos: um vindo da transição que representa o nó proposição e outro que vai para a transição representando o nó ação. A figura 3.7 mostra esta tradução.

As arestas pré-condição são representadas por um lugar devido ao fato de um nó proposição do grafo poder ser pré-condição de mais de uma ação, assim cada transição que representa ações tem um conjunto de arestas pré-condição independente. Caso esta

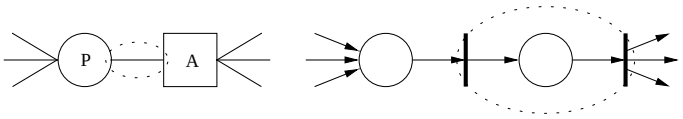

Figura 3.7: Tradução das arestas pré-condição.

estrutura não fosse utilizada, o disparo de uma transição ação desabilitaria outra transição ação que tivesse uma pré-condição em comum, pois o disparo da primeira removeria as marcas de todos os lugares representando suas pré-condições.

Exclusão mútua: a relação binária de exclusão mútua é traduzida em um lugar com dois arcos saindo, um para cada transição que representa cada nó ação da relação, e uma marca neste lugar. A figura 3.8 mostra esta tradução.

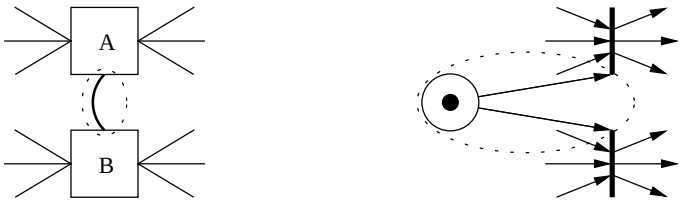

Figura 3.8: Tradução das relações de exclusão mútua.

Note que na figura 3.8 há uma marca no lugar que representa a relação de exclusão mútua porque queremos executar apenas uma das duas ações, e esta marca habilita as duas transições para disparo, mas após o disparo de uma delas a outra deixa de estar habilitada, impossibilitando o seu disparo.

As relações de exclusão mútua entre proposições não são representadas na rede, pois, estas são utilizadas somente durante a construção do grafo de planos para determinar as exclusões mútuas entre as ações do grafo.

Todos os arcos da rede têm peso igual a um, pois não há necessidade de colocarmos mais de uma marca em cada lugar da rede. Quando existe uma marca em um determinado lugar da rede isso pode indicar que uma proposição é válida ou que uma pré-condição é valida ou ainda que existe uma relação de exclusão mútua entre duas ações. Observe que após o disparo da um conjunto qualquer de transições é possível que existam lugares com mais de uma marca na rede, indicando que existe mais de um caminho na rede que valida a estrutura representada.

Para representar o estado inicial do problema de planificação colocamos uma marca em cada lugar que representa a camada zero do grafo de planos.

As marcas nos lugares do estado inicial e as marcas nos lugares que controlam as relações de exclusão mútua definem a marcação inicial da rede. A figura 3.9 mostra a

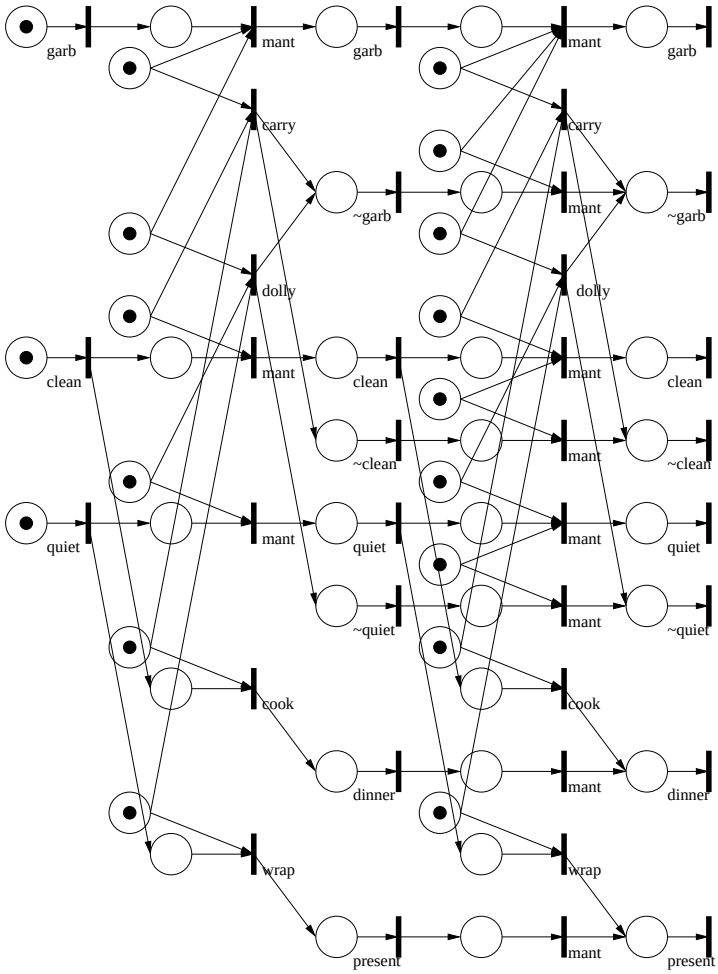

Figura 3.9: A rede de Petri com marcação inicial para o problema do jantar.

rede de Petri obtida pela tradução do problema do jantar apresentado na figura 3.2.

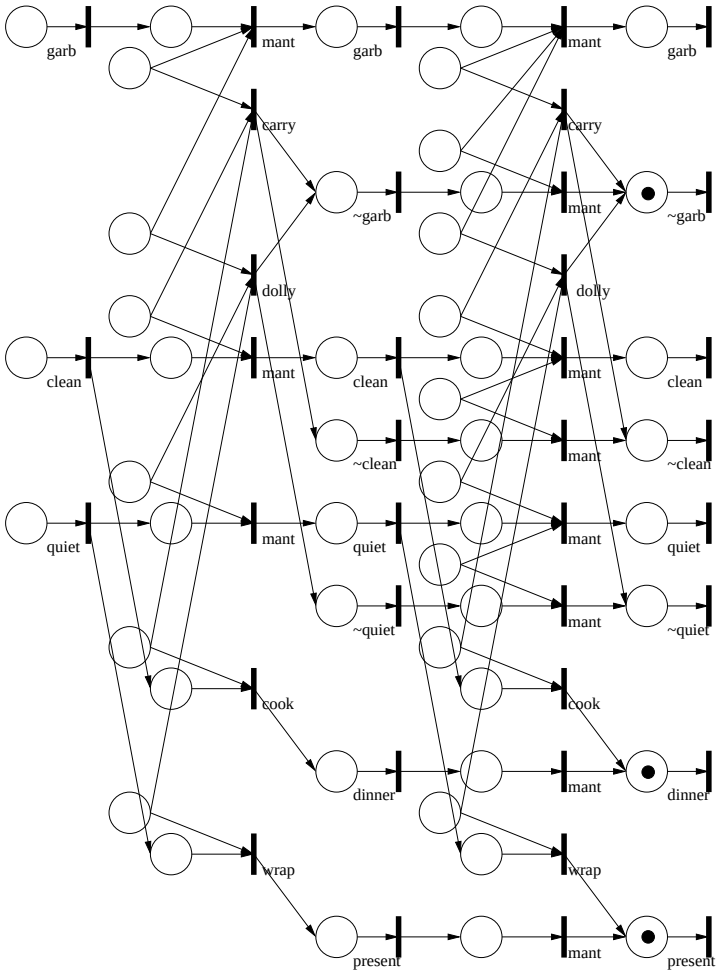

Figura 3.10: A rede de Petri com a marcação final desejada para o problema do jantar.

O estado meta do problema de planificação é representado pela sub-marcação da rede que contém marcas nos lugares que representam os nós proposições do estado meta. A figura 3.10 mostra uma rede de Petri com uma marcação que contém o estado meta para o problema do jantar.

Seja N a rede obtida pela tradução do grafo de planos, M_0 a marcação inicial da rede e M_g a marcação que contém marcas nos lugares que representam as proposições do estado meta. Podemos definir o problema de alcançabilidade de marcação relacionado como: "encontre uma seqüência s de transições de N que quando disparada transforma M_0 em M_g".

O problema aqui é que M_g é uma marcação completa da rede que contém os lugares representando os nós proposições do estado meta e não conhecemos a marcação inteira desejada. Conhecemos apenas o subconjunto $S_g \subseteq M_g$ que contém o estado meta. Portanto, o problema de alcançabilidade da marcação M_g não é bem formado, pois não conhecemos M_g.

A alternativa que temos é usar o problema de encontrar uma seqüência de transições

s que alcance alguma sub-marcação S_g, isto é, queremos resolver o problema de alcançabilidade da sub-marcação S_g a partir de M_0. A solução deste problema será tratada na próxima seção.

A seqüência de transições s define um conjunto de caminhos na rede de Petri que sai dos lugares que representam o estado inicial e chegam aos lugares que representam o estado meta. Considerando apenas as transições que representam ações nesta seqüência, temos uma estrutura semelhante à de um plano do problema original. A recuperação deste plano será tratada na seção 3.2.4.

3.2.3 Alcançabilidade e programação inteira

Usando a rede de Petri obtida na seção anterior vamos mostrar como usar métodos para problemas de programação inteira para resolver o problema de alcançabilidade de sub-marcação nesta rede e, assim, resolver o problema original de planificação.

Para resolver este problema usamos a equação fundamental (3.1) da rede N com a marcação inicial M_0 como um problema de programação inteira definido pela restrição vetorial:

$$C.\overline{s} \leq M_0 - S_g. \tag{3.3}$$

Qualquer vetor $\overline{s}$ que é solução para o problema de programação inteira (3.3) representa quantas vezes cada transição da rede é disparada para obter M_g a partir de M_0, onde $M_g(p) \geq S_g(p)$, para todo lugar p na rede N.

A partir do exemplo do problema do jantar das seções anteriores (figura 3.9), considere o sub-problema[1] de embrulhar o presente e remover o lixo da casa, dado pela rede N_p, apresentada nas figuras 3.11 e 3.12. Esta rede pode ser representada matricialmente pela matriz de incidência C_p dada por[2]:

$$C_p = \begin{bmatrix}
- & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & - & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & - & - & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & - & . & - & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & - & - & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & - & . & - & . & . & . & . & . & . & . & . & . & . & . & . & . \\
+ & . & - & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & + & . & . & . & - & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & + & . & . & . & . & - & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & + & . & . & . & . & - & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & + & + & . & . & . & - & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & + & . & . & . & - & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & + & . & . & . & . & . & - & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & + & . & . & . & . & - & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & - & - & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & - & . & - & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & - & . & . & - & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & - & - & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & - & - & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & - & . & . & - & . \\
. & . & . & . & . & . & . & + & . & . & . & . & - & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & + & . & . & . & . & . & - & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & + & . & . & . & . & . & . & - & . & . & . \\
. & . & . & . & . & . & . & . & . & . & + & . & . & . & . & . & . & - & . & . \\
. & . & . & . & . & . & . & . & . & + & . & . & . & . & . & . & . & . & - & . \\
. & . & . & . & . & . & . & . & . & . & . & + & . & . & . & . & . & . & . & - \\
. & . & . & . & . & . & . & . & . & . & . & . & + & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & + & + & + & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & + & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & + & . & + & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & + & + \\
\end{bmatrix}$$

Podemos interpretar os valores desta matriz como:

- um "+" em uma posição (x, y) da matriz indica que existe um arco entrando no lugar x e saindo da transição y,
- um "−" em uma posição (x, y) da matriz indica que existe um arco saindo do lugar

[1] Usaremos este sub-problema para simplificar a modelagem matricial, resultando em uma matrix de 31 linhas e 20 colunas, caso contrário teríamos uma matriz com 49 linhas e 38 colunas.

[2] Para facilitar a visualização os valores da matriz foram substituídos da seguinte forma, "0" por ".", "1" por "+" e "−1" por "−".

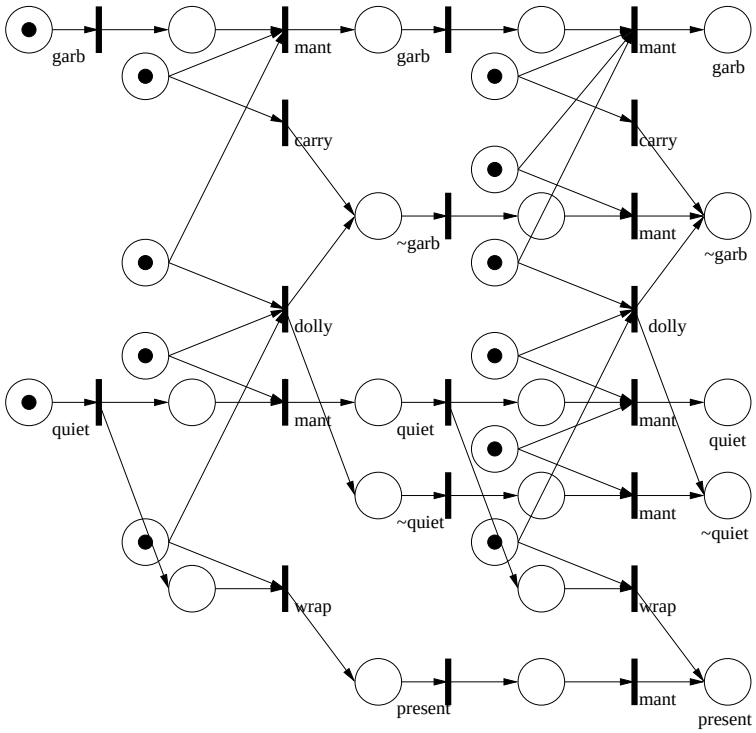

Figura 3.11: A rede de Petri com a marcação inicial o problema do presente e do lixo.

x e entrando na transição y e

- um "." em uma posição (x, y) da matriz indica que não existem arcos entre o lugar x e a transição y,

onde x é uma linha da matriz e y uma coluna. Estes índices, x para os lugares e y para as transições, são representados na figura 3.12 por números acima e a direita de cada estrutura da rede.

Note que as transições que representam os nós proposição da última camada do grafo, ou seja, a última camada da rede, foram removidas da representação, pois nenhuma ação tem como pré-condição estas proposições. Esta remoção não afeta a representação do problema e diminui o número de colunas da matriz C_p e portanto diminui o problema de programação inteira correspondente.

A marcação inicial M_{p_0} apresentada na figura 3.11 é dada por:

$$M_{p_0} = [\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ ].$$

A marcação final M_{p_g} apresentadas na figura 3.12 é dada por:

$$M_{p_g} = [\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 3\ 0\ 1\ 1\ ].$$

A sub-marcação $S_{p_g} \subset M_{p_g}$ que contém o estado meta para o problema do presente e

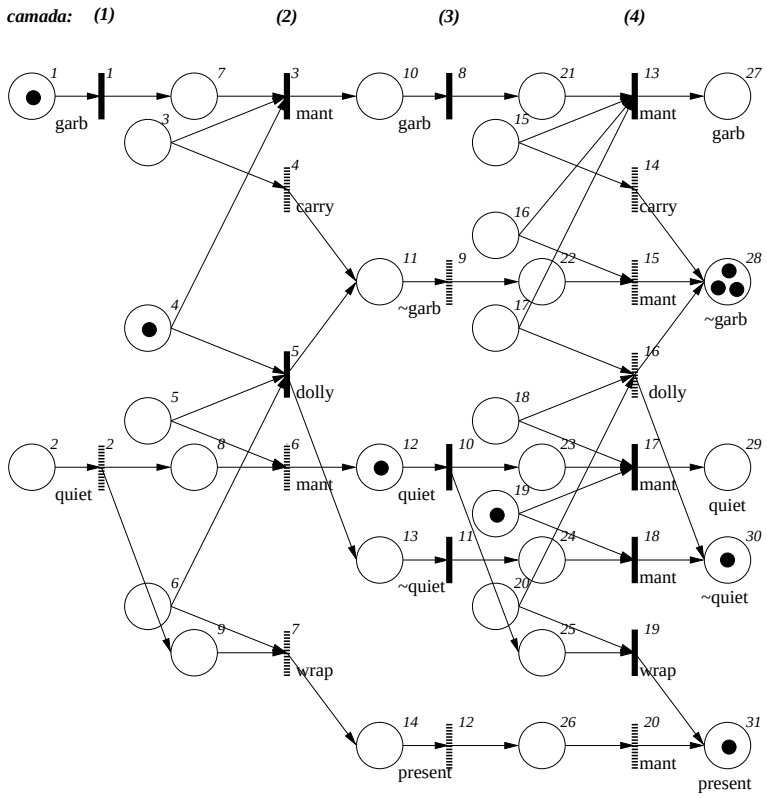

Figura 3.12: A rede de Petri com uma marcação final para o problema do presente e do lixo.

do lixo da figura 3.11 é dada pelo vetor abaixo, onde "·" indica que este lugar não pertence à sub-marcação:

$$S_{p_g} = [\ \cdot\ 1\ \cdot\ \cdot 1\ ].$$

Aplicando a inequação 3.3, que define o problema de programação inteira, sobre a rede de Petri N_p, dada pela matriz C_p, a marcação inicial M_{p_0} e a sub-marcação final S_{p_g}, temos:

$$C_p.\overline{s_p} \leq M_{p_0} - S_{p_g} \quad , \tag{3.4}$$

onde $\overline{s_p}$ é o vetor característico da rede N_p e representa o número de vezes que cada transição desta rede é disparada.

A inequação 3.4 é um problema de programação inteira para o problema do presente e do lixo. Uma solução para o problema de programação inteira, que obtém a marcação M_{p_g} da figura 3.12, é dada pelo vetor:

$$\overline{s_p} = [\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ ]^T.$$

As transições disparadas do vetor $\overline{s_p}$ estão representadas na figura 3.12 por barras verticais pontilhadas.

Observe que apesar das transições t_1 e t_{10} estarem habilitadas elas não foram disparadas em $\overline{s_p}$, pois não influenciam na obtenção da sub-marcação S_{p_g}. Este fato ocorre devido à desigualdade ($\leq$) da inequação 3.4, permitindo que marcas fiquem espalhadas pela rede caso estas não influenciem na obtenção de S_{p_g}.

O problema que temos agora é que o vetor $\overline{s_p}$ não é uma seqüência ordenada de transições que transforma a marcação M_{p0} na marcação M_{p_g}, ele apenas nos diz quantas vezes cada ação desta seqüência foi disparada. Resolvemos este problema tomando as transições da esquerda para a direita de acordo com as camadas verticais indicadas na figura 3.12. Assim temos a seqüência parcialmente ordenada s_p dada por:

$$s_p = (T_1, T_2, T_3, T_4),$$

onde T_i é o conjunto das transições t_j da camada i da figura 3.12, tal que $\overline{s_p}(t_j) \geq 1$, dado por:

$$\begin{aligned} T_1 &= \{t_2\}, \\ T_2 &= \{t_4, t_6, t_7\}, \\ T_3 &= \{t_9, t_{12}\}, \\ T_4 &= \{t_{14}, t_{15}, t_{16}, t_{20}\}. \end{aligned}$$

A partir da seqüência parcialmente ordenada s_p podemos obter seqüências totalmente ordenadas escolhendo uma ordem para os elementos dos conjuntos T_i, como por exemplo as seqüências abaixo:

$$\begin{aligned} s_{p1} &= (t_2, t_4, t_6, t_7, t_9, t_{12}, t_{14}, t_{15}, t_{16}, t_{20}), \\ s_{p2} &= (t_2, t_7, t_4, t_6, t_{12}, t_9, t_{16}, t_{20}, t_{14}, t_{15}). \end{aligned}$$

Portanto as seqüências de transições s_{p1} e s_{p2}, derivadas da solução $\overline{s_p}$ do problema de programação inteira (3.4), são soluções para o problema de alcançabilidade da sub-marcação S_{p_g} da rede de Petri N_p.

3.2.4 Encontrando o plano

A solução do problema de alcançabilidade de sub-marcação, apresentada na seção anterior, contém a solução para o problema de planificação original. A questão é: como recuperar este plano a partir da seqüência de transições parcialmente ordenada que resolve o problema de alcançabilidade.

Para que possamos encontrar um plano que resolve o problema de planificação, inicialmente usamos o seguinte procedimento para simplificar a rede de Petri com a marcação final para o problema de alcançabilidade:

1. remova da rede todos os lugares que representam relações de exclusão mútua, e os seus respectivos arcos;

2. remova da rede todas as transições t que tem $\overline{s}(t) = 0$, e os seus respectivos arcos;

3. a partir dos lugares l, tal que $l \in S_g$, percorra a rede no sentido contrário ao dos arcos marcando os lugares e as transições por onde passar;

4. remova todos os lugares e transições que não foram marcados no passo anterior, e os seus respectivos arcos.

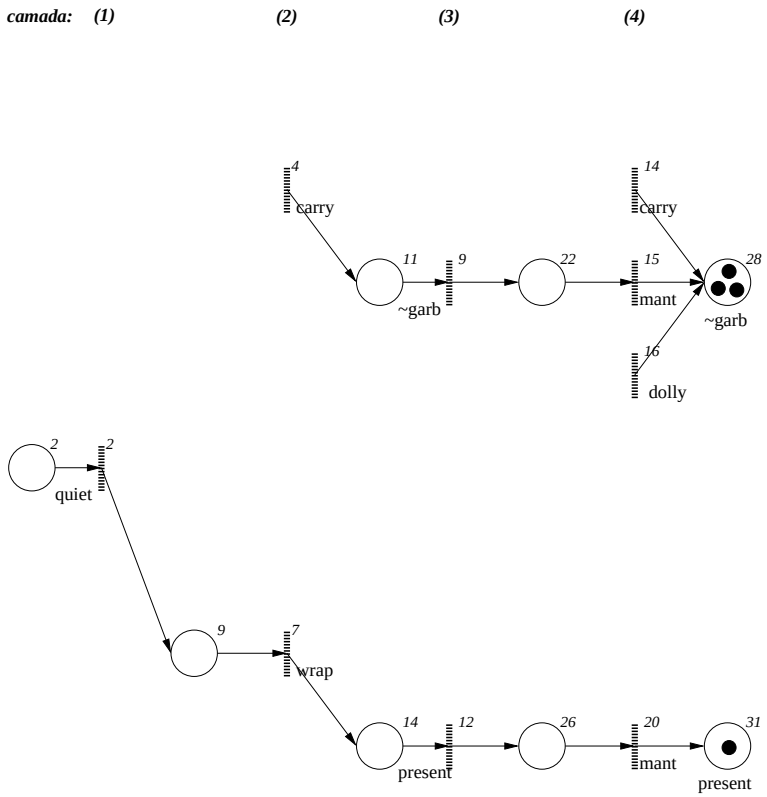

Figura 3.13: A parte relevante da rede de Petri para o problema do presente e do lixo.

A rede resultante contém apenas os caminhos que representam planos que são solução para o problema original. Considerando o problema do presente e do lixo da seção anterior (figura 3.12), após aplicarmos o procedimento acima temos a rede da figura 3.13 como resultado.

Para obtermos um plano que seja solução para o problema de planificação original basta escolhermos na rede um caminho para cada lugar de S_g, percorrendo os arcos em sentido contrário. As seqüências de transições que representam ações nestes caminhos constituem o plano.

Por exemplo, para o problema do presente e do lixo, podemos escolher as seguintes seqüencias $garb_i$ de transições para obter o lugar 28 da figura 3.13, que representa a

proposição $\sim garb$:

$$garb_1 = (t_{14}),$$
$$garb_2 = (t_4, t_9, t_{15}),$$
$$garb_3 = (t_{16}).$$

Para o lugar 31 que representa a proposição *present* temos apenas uma seqüência $pres_i$:

$$pres_1 = (t_2, t_7, t_{12}, t_{20}).$$

Intercalando estas seqüências, duas a duas, de acordo com as camadas onde as transições aparecem, temos as seguintes seqüências s_i parcialmente ordenadas:

$$s_1 = (\ t_2,\ t_7,\ t_{12},\ \{\ t_{14},\ t_{20}\ \}\ ),$$
$$s_2 = (\ t_2,\ \{\ t_4,\ t_7\ \},\ \{\ t_9,\ t_{12}\ \},\ \{\ t_{15},\ t_{20}\ \}\ ),$$
$$s_3 = (\ t_2,\ t_7,\ t_{12},\ \{\ t_{16},\ t_{20}\ \}\ ),$$

onde s_1 foi obtida a partir de $garb_1$ e $pres_1$, s_2 a partir de $garb_2$ e $pres_1$, e s_3 a partir de $garb_3$ e $pres_1$. Retirando destas seqüências as transições que representam proposições e as que representam ações de manutenção temos:

$$s'_1 = (\ t_7,\ t_{14}\ ),$$
$$s'_2 = (\ \{\ t_4,\ t_7\ \}\ ),$$
$$s'_3 = (\ t_7,\ t_{16}\ ).$$

A partir das seqüências s'_1, s'_2 e s'_3 obtemos os seguintes planos para o problema de planificação:

$$plano_1 = (\ wrap,\ carry\ ),$$
$$plano_2 = (\ \{\ carry,\ wrap\ \}\ ),$$
$$plano_3 = (\ wrap,\ dolly\ ),$$

onde os planos $plano_1$ e $plano_3$ são completamente ordenados, onde uma ação deve ser executada após a outra. O plano $plano_2$ é um plano paralelo, ou parcialmente ordenado, indicando que as ações podem ser executadas em qualquer ordem, ou ainda, ao mesmo tempo.

Os três planos obtidos a partir da seqüência parcialmente ordenada s_p e da simplificação da rede N_p são soluções para o problema de planificação: "embrulhar o presente e remover o lixo da casa", apresentado anteriormente.

3.3 Considerações finais

Neste capítulo apresentamos uma nova abordagem para o problema de planificação baseado em *STRIPS*, unindo o formalismo de redes de Petri a problemas de programação inteira.

Usando a teoria de redes de Petri mostramos como podemos definir o problema de planificação como um problema da alcançabilidade de sub-marcação em uma rede de Petri. Usamos técnicas de programação inteira para resolver o problema de alcançabilidade, obtendo a partir desta solução o plano que resolve o problema original de planificação.

O algoritmo proposto é baseado no *GRAPHPLAN* e possui três fases. A primeira é uma fase de pré-processamento que usa o grafo de planos para simplificar o espaço de busca. Na segunda fase o grafo de planos é transformado em uma rede de Petri e o problema de planificação em um problema de alcançabilidade. Este último é resolvido na terceira fase do algoritmo usando técnicas de programação inteira. A partir da solução encontrada para o problema de programação inteira e da simplificação da rede de Petri o plano para o problema original é recuperado.

Os procedimentos de tradução e recuperação da solução apresentados são bastante simples, ficando a complexidade do algoritmo diretamente ligada à criação do grafo de planos e à solução do problema de programação inteira.

O tamanho do problema de programação inteira obtido a partir da rede Petri é dado pelo número de restrições e variáveis deste problema. Baseado no processo de tradução do grafo de planos para rede Petri apresentado, o núnero de restrições do problema de programação inteira r é igual ao número de lugares da rede, dado por:

$$r = n_p + a_p + m,$$

onde n_p é o número de nós proposição do grafo, a_p é o número de arestas pré-condição do grafo e m o número de relações de exclusão mútua do grafo. O número de variáveis do problema de programação inteira v é igual ao número de transições da rede, dado por:

$$v = n_p + n_a,$$

onde n_p é o número de nós proposição do grafo e n_a é o número de nós ação do grafo. Portanto o tamanho do problema de programação inteira é linearmente proporcional ao tamanho do grafo de planos do problema original de planificação.

Para a implementação do algoritmo usamos o código fonte original do *GRAPHPLAN*, referente a expansão do grafo de planos, para a primeira fase. Na segunda fase o grafo foi traduzido para uma representação matricial da rede de Petri, armazenada em uma matriz esparsa, que foi usada na definição do problema de programação inteira da terceira fase. O problema de programação inteira foi resolvido usando-se uma biblioteca específica para

programação inteira. A recuperação da solução foi implementada a partir de um simples caminhamento na matriz de incidência da rede e no grafo de planos.

O algoritmo proposto faz a ligação entre duas abordagens já existentes para o problema de planificação usando as redes de Petri como ferramenta de ligação. A figura 1.1 mostra onde nossa abordagem se insere no atual contexto do problema de planificação baseado em *STRIPS*.

No capítulo 4 apresentamos alguns resultados comparativos obtidos com o Petriplan e outros planificadores atuais.

Capítulo 4

Resultados Comparativos

Neste capítulo apresentaremos uma reprodução parcial do experimento realizado na Competição de Planificadores do Congresso AIPS-98 [McD98a], adicionando o planificador Petriplan [SCK00], proposto no capítulo 3. O objetivo destes experimentos é comparar os planificadores entre si num cenário com recursos computacionais limitados, avaliando seu desempenho.

Nas próximas seções apresentamos uma breve descrição da competição de 1998, o experimento realizado e uma análise dos resultados obtidos.

4.1 A competição do AIPS-98

A competição realizada no AIPS-98 foi dividida em duas categorias: uma para planificadores que suportam a linguagem *STRIPS* [FN71] e outra para os que suportam a linguagem *ADL* [Ped89], que estende a primeira para efeitos condicionais e quantificadores.

Na categoria *STRIPS* participaram os seguintes planificadores: *BLACKBOX* [KS99], *HSP* [BG00], *IPP* [KNHD97] e *STAN* [LM99]. Esta categoria foi dividida em duas baterias: a primeira composta por 140 problemas em 5 domínios diferentes e a segunda com 15 problemas em 3 domínios diferentes, com o objetivo de refinar os resultados da primeira. Na categoria *ADL* os planificadores foram *IPP* e *SGP* [ASW98] com um conjunto equivalente de problemas.

Os problemas propostos nesta competição foram especificados na linguagem *PDDL*[1] [McD98b] e buscam explorar de forma bastante abrangente as características dos provadores, indo deste problemas simples, resolvidos rapidamente, até problemas complexos resultando na maioria das vezes em explosões combinatoriais.

Os planificadores foram avaliados com relação a tempo de processamento, tamanho da solução e número de problemas resolvidos. Os resultados detalhados desta competição

[1] O anexo A apresenta uma breve descrição da linguagem *PDDL*.

podem ser encontrados em [McD98a].

4.2 Experimentos realizados

Os experimentos que realizamos utilizaram o planificador Petriplan e aqueles da categoria *STRIPS* do AIPS-98: *BLACKBOX*, *HSP*, *IPP* e *STAN* em duas baterias: a primeira com 110 dos 140 problemas utilizados na competição e a segunda com os mesmos problemas da competição. Na primeira bateria os problemas de um dos domínios foram retirados devido a problemas de compatibilidade com os planificadores. A categoria *ADL* também foi retirada por tratar de uma linguagem com maior poder de representação, não compatível com o contexto deste trabalho.

Foram realizadas três execuções de cada bateria de experimentos visando avaliar os planificadores quanto ao tempo médio de execução para cada problema. A questão da qualidade das soluções obtidas pelos planificadores foi desconsiderada na avaliação para simplificar os experimentos realizados.

4.2.1 Domínios

Os problemas propostos para os planificadores foram divididos em cinco domínios:

- **Gripper**: Existe um robô com duas garras, podendo carregar uma bola em cada uma delas. O objetivo é levar N bolas de uma sala para outra. Este domínio tem como característica a explosão combinatória gerada pelo tratamento das garras de forma assimétrica por alguns planificadores.

- **Logistics**: Existem várias cidades, cada uma contendo várias localidades, sendo algumas aeroportos. Existem ainda caminhões que podem andar dentro de uma mesma cidade e aviões que podem voar entre aeroportos. O objetivo é levar alguns pacotes de várias localidades para outras.

- **Mystery**: Existe um grafo planar de nós, cada nó contém veículos, objetos e alguma quantidade de combustível. Objetos podem ser carregados em veículos (até a sua capacidade) e veículos podem se mover entre nós desde que exista combustível no nó de partida, diminuindo de uma unidade a quantidade de combustível. O objetivo é levar objetos de vários nós para outros. Para complicar, os nós são chamados de emoções, os objetos são sofrimentos, os veículos são prazeres e as quantidades e capacidades são entidades geográficas.

- **Mprime**: Este é o domínio Mystery com uma nova ação, a capacidade de mover uma unidade de combustível de um nó para outro vizinho, desde que o nó origem possua pelo menos duas unidades.

- **Grid**: Existe um quadriculado de localidades, um robô pode se mover entre os quadrados de um em um com movimentos horizontais ou verticais. Se um quadrado está trancado, o robô pode se mover para ele apenas depois de destravá-lo, para isto ele precisa ter uma chave do mesmo formato que a trava. O objetivo é levar chaves de várias localidades para outras.

4.2.2 Planificadores

Os planificadores avaliados pelo experimento realizado foram:

- **BLACKBOX**: Desenvolvido por Henry Kautz (ATT Corp.) e Bart Selman (ATT Corp. & Cornell University) na linguagem C, este planificador utiliza a junção do grafo de planos usado pelo *GRAPHPLAN* e a compilação para problemas de satisfabilidade (SAT). (Ver seção 2.4.4.)

- **HSP**: Desenvolvido por Blai Bonet e Hector Geffner (Universidad Simon Bolivar) na linguagem C, este planificador utiliza um algoritmo de busca heurística. (Ver seção 2.2.)

- **IPP**: Desenvolvido por Jana Koehler (Freiburg University) nas linguagens C e C++, este planificador é um descendente direto do *GRAPHPLAN* que utiliza verificação de tipos para otimizar o processo de busca pela solução. (Ver seção 2.4.)

- **STAN**: Desenvolvido por Derek Long e Maria Fox (Durham University) nas linguagens C e C++, este planificador também é uma derivação do *GRAPHPLAN* que utiliza várias técnicas para análise de estados com o objetivo de melhorar o desempenho do planificador. (Ver seção 2.4.)

- **Petriplan**: Planificador proposto no capítulo 3, escrito na linguagem C, utilizando como base o código fonte do *GRAPHPLAN*. (Ver capítulo 3.)

4.2.3 Metodologia

Os experimentos realizados foram divididos em duas baterias, a primeira contendo 110 problemas em 4 domínios diferentes sendo:

- 20 problemas do domínio Gripper, com a complexidade variando em relação ao número de bolas a serem transportadas de uma sala para outra;

- 30 problemas do domínio Logistics, com um número progressivo de cidades, localidades, aeroportos, caminhões e ítens a serem transportados;

- 30 problemas do domínio Mystery, com um número progressivo de elementos ou variando a estrutura de conexão entre os nós do grafo; e

- 30 problemas do domínio Mprime, seguindo a mesma complexidade dos problemas para o domínio Mystery.

A segunda bateria foi composta por 15 problemas sendo 5 novos problemas do domínio Logistics, 5 novos problemas do domínio Mprime e 5 problemas do domínio Grid, onde a complexidade varia de acordo com o tamanho do quadriculado e com o número de chaves disponíveis.

O equipamento utilizado no experimento foi um microcomputador Intel Pentium III de 500 MHz com memória principal de 256 Mbytes, utilizando o sistema operacional Debian GNU/Linux 2.2. Os planificadores foram compilados a partir de seus códigos fonte e configurações originais disponibilizados por seus autores [BLA00, HSP00, IPP00, STA00], usando as ferramentas GNU C Compiler, Flex e Bison.

Nas duas baterias do experimento os planificadores foram executados três vezes com cada problema de cada domínio, limitando o tempo de cada execução em 2 horas e o tamanho da memória em 256 Mbytes. Os tempos de execução apresentados na próxima seção são a média entre os três tempos obtidos.

4.2.4 Resultados

Nesta seção apresentamos os resumos dos resultados obtidos nos experimentos realizados com os cinco planificadores selecionados. As tabelas com os dados detalhados podem ser encontradas no anexo B. Os dados apresentados nas tabelas abaixo são quantidades de tempo medidas em milisegundos e representam o tempo de execução total de um planificador para um determinado problema. Os campos marcados com "x" indicam que o planificador não conseguiu encontrar uma solução para o problema dentro dos limites de tempo e memória estabelecidos.

As tabela 4.1 e 4.2 mostram o número de problemas resolvidos por cada planificador e o tempo médio gasto por problema.

Tabela 4.1: Número de problemas resolvidos por planificador.

Domínio	Petriplan	BLACKBOX	HSP	IPP	STAN	Problemas
Gripper-1	2	2	11	4	4	20
Logistics-1	1	3	4	5	2	30
Mystery-1	4	13	16	13	14	30
Mprime-1	0	15	23	16	8	30
Logistics-2	2	3	5	4	3	5
Mprime-2	0	4	4	4	4	5
Grid-2	0	1	0	3	1	5
Total	9	41	63	49	36	125

Utilizamos a mesma função de pontuação usada no AIPS-98 para classificar os planificadores, os resultados são apresentados nas tabelas 4.3 e 4.4. A função de pontuação

Tabela 4.2: Tempo médio por problema resolvido por planificador.

Domínio	Petriplan	BLACKBOX	HSP	IPP	STAN
Gripper-1	1353707,00	4140,50	2126,36	37675,00	789313,25
Logistics-1	4189902,00	4573,67	12171,00	716734,00	25272,50
Mystery-1	3835687,00	2084,61	12078,19	8077,69	762,93
Mprime-1	x	2538,40	118732,87	22396,87	1398,50
Logistics-2	3480784,00	529,67	10036,60	17438,25	9037,33
Mprime-2	x	1263,25	1767,50	6532,25	5205,50
Grid-2	x	11035,00	x	32064,33	3666,00
Total	2392250,00	2557,44	48467,21	66964,63	121939,03

P_{ij}, para um planificador i num problema j, é dada por:

$$P_{ij} = (N_j - R_{ij})W_i$$

onde N_j é o número de planificadores competindo no problema j, R_{ij} é a classificação em relação ao tempo do planificador i no problema j (0 para o mais rápido e $N_j - 1$ para o mais lento), e W_j é a dificuldade do problema j dada por:

$$W_j = \frac{\mathsf{mediana}_i[\log(T_{ij})]}{\sum_n \mathsf{mediana}_m[\log(T_{mn})]}$$

onde i e m são planificadores, n é um problema, T_{kl} é o tempo que o planificador k gastou para resolver o problema l e mediana é o valor mediano de um conjunto de valores.

Tabela 4.3: Resultados do experimento - bateria I.

	Petriplan	BLACKBOX	HSP	IPP	STAN
Resolvidos	7	63	82	63	64
Pontuação	0,03919	1,27485	2,47166	1,04484	1,14278

Tabela 4.4: Resultados do experimento - bateria II.

	Petriplan	BLACKBOX	HSP	IPP	STAN
Resolvidos	2	8	9	11	7
Pontuação	0,00990	0,27825	0,31319	0,40177	0,27081

Tabela 4.5: Totalização das pontuações dos planificadores.

Problema	Petriplan	BLACKBOX	HSP	IPP	STAN
Total	0,04909	1,55320	2,78485	1,44661	1,41359

4.3 Análise do resultados

Os resultados obtidos apresentados na seção 4.2.4 e no anexo B mostram o desempenho dos cinco planificadores utilizados nos experimentos.

Em relação aos quatro planificadores da competição do AIPS-98 os resultados obtidos são semelhantes àqueles obtidos na competição e mostram que os planificadores de melhor desempenho são o *HSP* e o *BLACKBOX*.

Podemos observar que a principal diferença entre os resultados obtidos pelo nosso experimento e os obtidos na competição é o número de problemas resolvidos por planificador, sendo o da competição maior. Esta diferença se deve ao fato dos planificadores, na competição, terem sido configurados por seus criadores para as condições e problemas específicos. Nos experimentos que realizamos foram usadas as configurações originais dos planificadores.

Os resultados obtidos pelo planificador Petriplan foram pouco significativos quando comparados aos outros, resolvendo apenas 9 dos 125 problemas propostos, num tempo médio de 39 minutos por problema. O motivo deste desempenho baixo está diretamente relacionado à implementação do planificador, pois a biblioteca *LP-SOLVE*[2] utilizada para resolver o problema de programação inteira resultante apresentou dois problemas. O primeiro é o desempenho da biblioteca, que é extremamente baixo quando comparado a outras bibliotecas comerciais [Mit00], fazendo com que, para grande parte dos problemas, o limite de tempo de duas horas fosse atingido. Outro problema é que a biblioteca resultou em vários casos em erros internos de execução.

Acreditamos que a substituição desta biblioteca por outra de melhor desempenho pode resultar em um planificador de desempenho comparável ao daqueles citados na seção 2.6 que também usam programação inteira.

4.4 Considerações finais

Neste capítulo apresentamos experimentos realizados nos moldes da competição do AIPS-98 com o objetivo de comparar a implementação do algoritmo proposto no capítulo 3 com alguns planificadores atuais derivados das técnicas apresentadas no capítulo 2. Podemos concluir que este objetivo não foi alcançado pelas limitações causadas pela biblioteca para programação inteira utilizada na implementação do Petriplan.

Outro ponto a ser considerado é a quantidade de memória usada pelo Petriplan, pois além do espaço para armazenar o grafo de planos gerado na primeira fase, a biblioteca para progamação inteira usa uma quantidade proporcional de memória para resolver o problema (ver seção 3.2.3). Portanto a quantidade de memória disponível para o sistema é

[2]LP-SOLVE é uma biblioteca de livre distribuição para programação inteira mista disponível em ftp://ftp.ics.ele.tue.nl/pub/lp_solve/

um fator limitante para o tamanho dos problemas de planificação que podem ser tratados.

Apesar do baixo desempenho, os resultados obtidos mostram que o algoritmo proposto resolve corretamente alguns dos problemas propostos, sendo limitado principalmente pelos problemas técnicos da biblioteca utilizada.

Capítulo 5

Conclusões

Neste texto apresentamos o problema de planificação em inteligência artificial baseado em *STRIPS*, uma revisão do estado da arte na área e uma nova abordagem para o problema, resultando em um novo algoritmo para problemas de planificação, o Petriplan. Também foram realizados experimentos com alguns planificadores atuais, reproduzindo parcialmente a competição de planificadores do AIPS-98.

Atualmente a área de planificação em IA encontra-se em plena expansão, contrariando o período de estagnação que sofreu até o início dos anos 90. Várias áreas do conhecimento contribuem hoje para o desenvolvimento de soluções e pesquisas em planificação, como por exemplo: teoria dos grafos, prova de teoremas, heurísticas, satisfabilidade, programação inteira e pesquisa operacional. Esta diversidade nos dá um cenário bastante vasto abrindo cada vez mais alternativas para o desenvolvimento de pesquisas na área.

Nosso trabalho contribui nesta direção mostrando como podemos agregar conhecimentos da área de redes de Petri para resolver problemas de planificação, criando um relacionamento com duas outras abordagens, a utilização de teoria dos grafos e o uso de programação inteira (ver figura 1.1).

Mostramos como transformar um problema de planificação em um problema de alcançabilidade de sub-marcação em redes de Petri e como resolver este último usando programação inteira. O algoritmo resultante desta abordagem possui três fases.

A primeira fase consiste da construção do grafo de planos seguindo as mesmas regras do *GRAPHPLAN*. Esta fase pode ser vista como um pré-processamento, buscando simplificar o problema para a próxima fase.

Na segunda fase, o grafo de planos gerado é transformado em uma rede de Petri equivalente, onde o problema de planificação original pode ser visto como um problema de alcançabilidade de sub-marcação.

Na terceira fase do algoritmo o problema de alcançabilidade definido na fase anterior é convertido em um problema de programação inteira, que então é resolvido por sistemas específicos para otimização de problemas de programação inteira.

Os experimentos realizados validam a abordagem apresentada a partir de um conjunto

de problemas de planificação utilizado pela comunidade da área. Os resultados obtidos mostram um baixo desempenho do planificador implementado. Apesar destes, a abordagem é promissora, pois o desempenho se deve principalmente a problemas técnicos da biblioteca de programação inteira utilizada.

A substituição da biblioteca atual para programação inteira por outra de melhor desempenho pode resultar em um planificador mais rápido. Pretendemos realizar tal alteração num futuro próximo, usando a biblioteca ILOG/CPLEX.

Outra questão ligada ao desempenho é o tamanho do problema de programação inteira resultante do processo de tradução. Como trabalho futuro podemos explorar variações da tradução para redes de Petri apresentada buscando reduzir o tamanho do problema de programação inteira.

Uma alternativa ao uso de programação inteira para resolver o problema de alcançabilidade pode ser tratado futuramente. Por exemplo, o uso de procedimentos de busca heurística diretamente sobre a estrutura da rede de Petri, ou a aplicação de outras técnicas de análise usadas na área de redes de Petri [Mur89].

Os relacionamentos apresentados na figura 1.1 e os trabalhos citados nesta, indicam uma forte ligação entre a abordagem proposta e aquelas baseadas em restrições. Explicitar o relacionamento entre estas abordagens também é uma extensão futura. Outra relação interessante nesta figura que pode ser explorada futuramente é a adaptação das heurísticas propostas em [BG00] para a abordagem que propomos.

Referências Bibliográficas

[ASW98] C. Anderson, D. E. Smith, and D. Weld. Conditional effects in graphplan. In *Proc. of AIPS98*, 1998.

[BC99] P. Beek and X. Chen. Cplan: A constraint programming approach to planning. In *Proc. of the Sixteenth National Conference on AI (AAAI-99)*, 1999.

[BD98] A. Bockmayr and Y. Dimopoulos. Mixed integer programming models for planning problems. In *Proc. of the Workshop on Constraint Problem Reformulation (CP-98)*, pages 1–6, Pisa, Italy, 1998.

[BF95] A. Blum and M. Furst. Fast planning through planning graph analysis. In *Proceedings of IJCAI-95*, pages 1636–1642, Montreal, August 1995.

[BF97] A. Blum and M. Furst. Fast planning through planning graph analysis. *Journal of Artificial Intelligence*, 90(1-2):281–300, 1997.

[BG98] B. Bonet and H. Geffner. HSP: Planning as heuristic search. In *Entry at the AIPS-98 Planning Competition*, Pittsburgh, June 1998.

[BG99] B. Bonet and H. Geffner. Planning as heuristic search: New results. In *Proc. European Conference on Planning (ECP-99)*, Durham, UK, 1999.

[BG00] B. Bonet and H. Geffner. Planning as heuristic search, February 2000. Submitted.

[Bib97] Wolfgang Bibel. Let's plan it deductively. In *15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, volume 2, pages 1549–1562. Morgan Kauffmann, 1997.

[BLA00] Blackbox, 2000. http://www.research.att.com/~kautz/blackbox/index.html.

[BSW91] A. Barrett, S. Soderland, and D. Weld. The effect of step-order representations on planning. Technical report, Univ. of Washington, Computer Science Dept., 1991.

[Byl94] T. Bylander. The computational complexity of propositional strips planning. *Artificial Intelligence*, 69:165–205, 1994.

[Cha87] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–377, 1987.

[CLR90] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[ENV91] K. Erol, D.S. Nau, and Subrahmanian V.S. Complexity, decidability and undecidability results for domain-independent planning: A detailed analysis. Technical report, University of Maryland, 1991. CS-TR-2797, UMIACS-TR-91-154, SRC-TR-91-96.

[ETA97] Discussion with wolfgang bibel on topics in his invited paper at ijcai 1997. ETAI, Sep. 1997. http://www.ida.liu.se/ext/etai/actions/nj/9709-1/.

[ETA98] Discussion with wolfgang bibel on topics in his invited paper at ijcai 1997. ETAI, Jan. 1998. http://www.ida.liu.se/ext/etai/actions/nj/9801-3/.

[FN71] R. Fikes and N. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Journal of Artificial Intelligence*, 2(3-4), 1971.

[HSP00] Hsp, 2000. http://www.ldc.usb.ve/~hector/.

[IPP00] Ipp, 2000. http://www.informatik.uni-freiburg.de/~koehler/ipp.html.

[KNHD97] J. Koehler, B. Nebel, J. Hoffmann, and Y. Dimopoulos. Extending planning graphs to an adl subset. Technical report, ICS - University of Freiburg, 1997.

[KS92] H. Kautz and B. Selman. Planning as satisfiability. In *Proc. 10th Eur. Conf. AI*, pages 359–363, Vienna, Austria, 1992. Wiley.

[KS96] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, Portland, OR, 1996.

[KS99] H. Kautz and B. Selman. Unifying sat-based and graph-based planning. In *Proceedings of IJCAI*, 1999.

[KW99] H. Kautz and J.P. Walser. State-space planning by integer optimization. In *Proceedings Fifteenth National Conference on Artificial Intelligence (AAAI-99)*, 1999.

[LM99] D. Long and Fox M. The efficient implementation of the plan-graph. In *JAIR*, 1999.

[McD96] D. McDermott. A heuristic estimator for means-ends analysis in planning. In *Proc. AIPS-96*, 1996.

[McD98a] D. McDermott. Aips-98 planning competition results. Technical report, http://ftp.cs.yale.edu/pub/mcdermott/aipscomp-results.html, 1998.

[McD98b] D. McDermott. *PDDL - The Planning Domain Definition Language*. http://ftp.cs.yale.edu/pub/mcdermott, 1998.

[MH69] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence 4*, 1969.

[Mit00] H. Mittelmann. Benchmarks for optimization software. Technical report, http://plato.la.asu.edu/bench.html, 2000.

[MR91] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. Technical report, Proceedings of the Ninth National Conference on Artificial Intelligence, 1991.

[Mur89] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[NS63] A. Newell and H. Simon. Gps: a program that simulates human thought. In E. Feigenbaum and J. Feldman, editors, *Computers and Thought*, pages 279–293. McGraw Hill, 1963.

[Pea84] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, Massachusetts, 1984.

[Ped89] E. P. D. Pednault. Adl: Exploring the middle qround between strips and the situation calculus. In *Proceedings of Knowledge Representation Conf.*, 1989.

[Rei78] R. Reiter. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*. Plenum Press, 1978.

[RN95] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 1995.

[Sac75] E. Sacerdoti. The nonlinear nature of plans. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, 1975.

[SCK00] F. Silva, M. Castilho, and L. Künzle. Petriplan: a new algorithm for plan generation (preliminary report). In *Proc. of IBERAMIA/SBIA-2000*, 2000. Aceito.

[Sha97] M. Shanahan. *Solving the Frame Problem*. MIT Press, 1997.

[SKC94] B. Selman, H.A. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings of the AAAI-94*, Seattle, WA, July 1994.

[SLM92] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 440–446, San Jose, CA, July 1992.

[STA00] Stan, 2000. http://www.dur.ac.uk/~dcs0www/research/stanstuff/.

[Sus75] G. J. Sussman. *A computer model of skill acquisition*. American Elsevier, New York, 1975.

[Tat74] A. Tate. Interplan: A plan generation system which can deal with interactions between goals. Technical report, Univ. of Edinburgh, Machine Intelligence Research Unit, 1974.

[Tat77] A. Tate. Generating project networks. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, 1977.

[VBLN99] T. Vossen, M. Ball, A. Lotem, and D. Nau. On the use of integer programming models in ai planning. In *Proceedings of the IJCAI 99*, pages 304–309, 1999.

[VBLN00] T. Vossen, M. Ball, A. Lotem, and D. Nau. Applying integer programming to AI planning. *Knowledge Engineering Review*, 2000. To appear.

[VPCG99] R. Valette, B. Pradin-Chézalviel, and F. Girault. *Fuzziness in Petri Nets*, chapter An Introduction to Petri Net Theory. Physica-Verlag, Heidelberg, 1999.

[War74] D. Warren. Warplan: A system for generating plans. Technical report, Computational Logic Dept., School of AI, Univ. of Edinburgh, 1974.

[Wel99] D. Weld. Recent advances in ai planning. *AI Magazine*, 1999.

[Wol98] L. Wolsey. *Integer Programming*. John Wiley, New York, 1998.

Apêndice A

PDDL: Planning Domain Definition Language

A linguagem PDDL - *The Planning Domain Definition Language* é uma linguagem desenvolvida especialmente para a Competição de Planificadores realizada no AIPS-98, com o objetivo de unificar a definição de domínios e problemas para sistemas de planificação, permitindo assim compará-los de forma direta.

A característica principal da linguagem PDDL é o fato de ser uma derivação de outras formalizações para planinficaçãto como STRIPS e ADL e outros formalismos utilizados em planificadores específicos. Esta característica garante uma integração simples com os sistemas planificadores atuais além de fornecer uma capacidade de representação equivalente ao conjunto destes formalismos, como por exemplo:

- Ações representadas no estilo STRIPS;
- Efeitos condicionais;
- Quantificadores universais;
- Universos dinâmicos, permitindo a criação e a destruição de objetos;
- Axiomas de domínio;
- Especificação de ações hierárquicas compostas de sub-ações e sub-metas;
- Gerenciamento de múltiplos problemas em múltiplos domínios.

Abaixo apresentamos a representação em PDDL do domínio Grid citado anteriormente e um problema de planificação para este domínio. Uma maior detalhamento da sintaxe e da semântica da linguagem pode ser encontrado em [McD98b].

A representação de um domínio em PDDL é dada por uma declaração semelhante a LISP contendo os seguintes ítens:

- a definição do nome do domínio;
- o conjunto de características de representação necessárias para a definição deste domínio;
- o conjunto de proposições presentes no domínio e
- as ações que podem ser executadas neste domínio, onde as ações são dadas por:
  - nome da ação;
  - parâmetros passados para a ação;
  - conjunção das pré-condições da ação e
  - conjunção dos efeitos da ação.

Como exemplo temos abaixo o início da definição do domínio Grid citado anteriormente onde: o nome do domínio é *grid*, a única característica de representação necessária para este domínio é a representação STRIPS e o conjunto de proposições é dado por: *conn, key-shape, at, at-robot, place, key, shape, locked, holding, open* e *arm-empty*. As palavras iniciadas por ":" são rótulos reservados da linguagem PDDL e as iniciadas por "?" são variáveis.

```
(define (domain grid)
 (:requirements :strips)
 (:predicates (conn ?x ?y)
              (key-shape ?k ?s)
              (lock-shape ?x ?s)
              (at ?r ?x )
              (at-robot ?x)
              (place ?p)
              (key ?k)
              (shape ?s)
              (locked ?x)
              (holding ?k)
              (open ?x)
              (arm-empty))
```

Abaixo temos as ações possíveis neste domínio: *unlock, move, pickup, pickup-and-loose* e *putdown*, com seus respectivos parâmetros, pré-condições e efeitos.

```
 (:action unlock
  :parameters (?curpos ?lockpos ?key ?shape)
  :precondition (and (place ?curpos)
                     (place ?lockpos)
                     (key ?key)
                     (shape ?shape)
                     (conn ?curpos ?lockpos)
                     (key-shape ?key ?shape)
```

```
                    (lock-shape ?lockpos ?shape)
                    (at-robot ?curpos)
                    (locked ?lockpos)
                    (holding ?key))
 :effect (and (open ?lockpos)
              (not (locked ?lockpos))))

(:action move
 :parameters (?curpos ?nextpos)
 :precondition (and (place ?curpos)
                    (place ?nextpos)
                    (at-robot ?curpos)
                    (conn ?curpos ?nextpos)
                    (open ?nextpos))
 :effect (and (at-robot ?nextpos)
              (not (at-robot ?curpos))))

(:action pickup
 :parameters (?curpos ?key)
 :precondition (and (place ?curpos)
                    (key ?key)
                    (at-robot ?curpos)
                    (at ?key ?curpos)
                    (arm-empty ))
 :effect (and (holding ?key)
              (not (at ?key ?curpos))
              (not (arm-empty))))

(:action pickup-and-loose
 :parameters (?curpos ?newkey ?oldkey)
 :precondition (and (place ?curpos)
                    (key ?newkey)
                    (key ?oldkey)
                    (at-robot ?curpos)
                    (holding ?oldkey)
                    (at ?newkey ?curpos))
 :effect (and (holding ?newkey)
              (at ?oldkey ?curpos)
              (not (holding ?oldkey))
              (not (at ?newkey ?curpos))))

(:action putdown
 :parameters (?curpos ?key)
 :precondition (and (place ?curpos)
                    (key ?key)
                    (at-robot ?curpos)
                    (holding ?key))
```

```
  :effect (and (arm-empty)
               (at ?key ?curpos)
               (not (holding ?key)))))
```

Abaixo temos o problema de planificação grid-3x3 para este domínio, definido por um quadriculado de 3x3, três formatos de chaves, quatro chaves diferentes, o estado inicial e o estado meta.

```
(define (problem grid-3x3)
   (:domain grid)
   (:objects node0-0 node0-1 node0-2
             node1-0 node1-1 node1-2
             node2-0 node2-1 node2-2
             triangle square circle
             key0 key1 key2 key3)
   (:init (arm-empty)
          (place node0-0)
          (place node0-1)
          (place node0-2)
          (place node1-0)
          (place node1-1)
          (place node1-2)
          (place node2-0)
          (place node2-1)
          (place node2-2)
          (shape triangle)
          (shape square)
          (shape circle)
          (conn node0-0 node1-0)
          (conn node0-0 node0-1)
          (conn node0-1 node1-1)
          (conn node0-1 node0-2)
          (conn node0-1 node0-0)
          (conn node0-2 node1-2)
          (conn node0-2 node0-1)
          (conn node1-0 node2-0)
          (conn node1-0 node0-0)
          (conn node1-0 node1-1)
          (conn node1-1 node2-1)
          (conn node1-1 node0-1)
          (conn node1-1 node1-2)
          (conn node1-1 node1-0)
          (conn node1-2 node2-2)
          (conn node1-2 node0-2)
          (conn node1-2 node1-1)
          (conn node2-0 node1-0)
          (conn node2-0 node2-1)
          (conn node2-1 node1-1)
```

```
        (conn node2-1 node2-2)
        (conn node2-1 node2-0)
        (conn node2-2 node1-2)
        (conn node2-2 node2-1)
        (locked node2-2)
        (lock-shape node2-2 square)
        (locked node0-2)
        (lock-shape node0-2 triangle)
        (locked node1-1)
        (lock-shape node1-1 circle)
        (locked node2-0)
        (lock-shape node2-0 circle)
        (open node0-0)
        (open node0-1)
        (open node1-0)
        (open node1-2)
        (open node2-1)
        (key key0)
        (key-shape key0 triangle)
        (at key0 node0-2)
        (key key1)
        (key-shape key1 triangle)
        (at key1 node2-0)
        (key key2)
        (key-shape key2 circle)
        (at key2 node0-0)
        (key key3)
        (key-shape key3 square)
        (at key3 node1-0)
        (at-robot node0-0))
  (:goal (and (at key0 node2-2))))
```

Apêndice B

Resultados dos Experimentos

As tabelas B.1, B.2, B.3 e B.4 apresentam os tempos obtidos na primeira fase dos experimentos realizados, para os problemas dos domínios Gripper, Logistics, Mystery e Mprime respectivamente.

Tabela B.1: Tempos de execução para os 20 problemas do domínio Gripper.

Problema	Petriplan	BLACKBOX	HSP	IPP	STAN
01	357003	139	166	40	143
02	2350411	8142	315	540	1131
03	x	x	523	10810	49791
04	x	x	893	139310	3106188
05	x	x	1137	x	x
06	x	x	1548	x	x
07	x	x	2039	x	x
08	x	x	2823	x	x
09	x	x	3627	x	x
10	x	x	4930	x	x
11	x	x	5394	x	x
12	x	x	x	x	x
13	x	x	x	x	x
14	x	x	x	x	x
15	x	x	x	x	x
16	x	x	x	x	x
17	x	x	x	x	x
18	x	x	x	x	x
19	x	x	x	x	x
20	x	x	x	x	x

Tabela B.2: Tempos de execução para os 30 problemas do domínio Logistics.

Problema	Petriplan	BLACKBOX	HSP	IPP	STAN
01	4189902	1936	4181	860	1063
02	x	5747	27358	777880	x
03	x	x	x	x	x
04	x	x	x	x	x
05	x	x	2420	3500	49482
06	x	x	x	x	x
07	x	x	x	x	x
08	x	x	x	x	x
09	x	x	x	x	x
10	x	x	x	x	x
11	x	6038	14725	8820	x
12	x	x	x	x	x
13	x	x	x	x	x
14	x	x	x	x	x
15	x	x	x	x	x
16	x	x	x	x	x
17	x	x	x	2792610	x
18	x	x	x	x	x
19	x	x	x	x	x
20	x	x	x	x	x
21	x	x	x	x	x
22	x	x	x	x	x
23	x	x	x	x	x
24	x	x	x	x	x
25	x	x	x	x	x
26	x	x	x	x	x
27	x	x	x	x	x
28	x	x	x	x	x
29	x	x	x	x	x
30	x	x	x	x	x

Tabela B.3: Tempos de execução para os 30 problemas do domínio Mystery.

Problema	Petriplan	BLACKBOX	HSP	IPP	STAN
01	499184	112	176	100	52
02	x	5692	5829	16010	516
03	2500312	580	930	1190	261
04	x	x	x	x	x
05	x	x	x	x	x
06	x	x	x	x	x
07	x	x	x	x	x
08	x	x	x	x	x
09	x	1350	1692	1830	212
10	x	x	97417	x	x
11	x	707	292	370	146
12	x	x	x	x	x
13	x	x	x	x	2283
14	x	x	x	x	x
15	x	x	31548	x	x
16	x	x	x	x	x
17	x	3199	22833	40890	1892
18	x	x	x	x	x
19	x	6966	8443	27410	1251
20	x	x	12609	x	x
21	x	x	x	x	x
22	x	x	x	x	x
23	x	x	x	x	x
24	x	x	x	x	x
25	2124991	115	183	110	52
26	x	1599	1094	2000	903
27	2546887	564	1172	960	354
28	x	515	434	250	75
29	x	484	841	840	129
30	x	5217	7758	13050	2555

Tabela B.4: Tempos de execução para os 30 problemas do domínio Mprime.

Problema	Petriplan	BLACKBOX	HSP	IPP	STAN
01	x	776	423	3360	1555
02	x	5950	9648	33080	1879
03	x	832	1345	6040	x
04	x	789	374	3520	x
05	x	x	x	10160	1656
06	x	x	524938	x	x
07	x	x	x	17360	1240
08	x	2623	14654	45000	x
09	x	1770	1966	6940	1515
10	x	x	287894	x	x
11	x	1691	626	5910	x
12	x	2170	2540	8240	x
13	x	x	x	x	x
14	x	x	x	x	x
15	x	x	200854	x	x
16	x	4830	11157	x	x
17	x	6611	54366	125050	x
18	x	x	x	x	x
19	x	x	42922	x	x
20	x	x	250779	x	x
21	x	x	x	x	1071
22	x	x	319557	x	x
23	x	x	x	x	x
24	x	x	242954	x	x
25	x	142	215	1350	393
26	x	2973	2724	18160	x
27	x	2966	7832	44890	x
28	x	1824	1725	6670	1879
29	x	2129	2974	22620	x
30	x	x	748389	x	x

As tabela B.5, B.6 e B.7 apresentam os tempos obtidos na segunda fase para os problemas dos domínios Logistics, Mprime e Grid respectivamente.

Tabela B.5: Tempos de execução para os 5 problemas do domínio Logistics.

Problema	Petriplan	BLACKBOX	HSP	IPP	STAN
01	1301147	378	859	190	136
02	5660421	464	1368	252	221
03	x	747	3296	520	26755
04	x	x	18466	x	x
05	x	x	26194	68791	x

Tabela B.6: Tempos de execução para os 5 problemas do domínio Mprime.

Problema	Petriplan	BLACKBOX	HSP	IPP	STAN
01	x	781	1746	9811	978
02	x	2315	2299	2564	18366
03	x	x	x	x	x
04	x	1533	2538	10969	467
05	x	424	487	2785	1011

Tabela B.7: Tempos de Execução para os 5 problemas do domínio Grid.

Problema	Petriplan	BLACKBOX	HSP	IPP	STAN
01	x	11035	x	4172	3666
02	x	x	x	12936	x
03	x	x	x	x	x
04	x	x	x	79085	x
05	x	x	x	x	x

Os resultados da aplicação desta função sobre os dados apresentados estão nas tabela B.8, B.9, B.10, B.11, B.12, B.13 e B.14. A totalização das pontuações é apresentada na tabula 4.5.

Tabela B.8: Pontuação dos planificadores por problema para o domínio Gripper.

Problema	Petriplan	BLACKBOX	HSP	IPP	STAN
01	0,00445	0,02427	0,00809	0,03236	0,01618
02	0,00577	0,01153	0,04611	0,03458	0,02305
03	0,00000	0,00000	0,06091	0,04568	0,03046
04	0,00000	0,00000	0,07768	0,05826	0,03884
05	0,00000	0,00000	0,04614	0,00000	0,00000
06	0,00000	0,00000	0,04817	0,00000	0,00000
07	0,00000	0,00000	0,04997	0,00000	0,00000
08	0,00000	0,00000	0,05211	0,00000	0,00000
09	0,00000	0,00000	0,05375	0,00000	0,00000
10	0,00000	0,00000	0,05576	0,00000	0,00000
11	0,00000	0,00000	0,05635	0,00000	0,00000
12	0,00000	0,00000	0,00000	0,00000	0,00000
13	0,00000	0,00000	0,00000	0,00000	0,00000
14	0,00000	0,00000	0,00000	0,00000	0,00000
15	0,00000	0,00000	0,00000	0,00000	0,00000
16	0,00000	0,00000	0,00000	0,00000	0,00000
17	0,00000	0,00000	0,00000	0,00000	0,00000
18	0,00000	0,00000	0,00000	0,00000	0,00000
19	0,00000	0,00000	0,00000	0,00000	0,00000
20	0,00000	0,00000	0,00000	0,00000	0,00000
Total	0,01022	0,03580	0,55504	0,17088	0,10853

Tabela B.9: Pontuação dos planificadores por problema para o domínio Logistics.

Problema	Petriplan	BLACKBOX	HSP	IPP	STAN
01	0,00620	0,02482	0,01241	0,04963	0,03722
02	0,00000	0,06700	0,05025	0,03350	0,00000
03	0,00000	0,00000	0,00000	0,00000	0,00000
04	0,00000	0,00000	0,00000	0,00000	0,00000
05	0,00000	0,00000	0,05352	0,04014	0,02676
06	0,00000	0,00000	0,00000	0,00000	0,00000
07	0,00000	0,00000	0,00000	0,00000	0,00000
08	0,00000	0,00000	0,00000	0,00000	0,00000
09	0,00000	0,00000	0,00000	0,00000	0,00000
10	0,00000	0,00000	0,00000	0,00000	0,00000
11	0,00000	0,05958	0,02979	0,04468	0,00000
12	0,00000	0,00000	0,00000	0,00000	0,00000
13	0,00000	0,00000	0,00000	0,00000	0,00000
14	0,00000	0,00000	0,00000	0,00000	0,00000
15	0,00000	0,00000	0,00000	0,00000	0,00000
16	0,00000	0,00000	0,00000	0,00000	0,00000
17	0,00000	0,00000	0,00000	0,09734	0,00000
18	0,00000	0,00000	0,00000	0,00000	0,00000
19	0,00000	0,00000	0,00000	0,00000	0,00000
20	0,00000	0,00000	0,00000	0,00000	0,00000
21	0,00000	0,00000	0,00000	0,00000	0,00000
22	0,00000	0,00000	0,00000	0,00000	0,00000
23	0,00000	0,00000	0,00000	0,00000	0,00000
24	0,00000	0,00000	0,00000	0,00000	0,00000
25	0,00000	0,00000	0,00000	0,00000	0,00000
26	0,00000	0,00000	0,00000	0,00000	0,00000
27	0,00000	0,00000	0,00000	0,00000	0,00000
28	0,00000	0,00000	0,00000	0,00000	0,00000
29	0,00000	0,00000	0,00000	0,00000	0,00000
30	0,00000	0,00000	0,00000	0,00000	0,00000
Total	0,00620	0,15139	0,14596	0,26529	0,06398

Tabela B.10: Pontuação dos planificadores por problema para o domínio Mystery.

Problema	Petriplan	BLACKBOX	HSP	IPP	STAN
01	0,00847	0,01695	0,00848	0,02543	0,03391
02	0,00000	0,04264	0,02843	0,01421	0,05686
03	0,00521	0,03130	0,02086	0,01043	0,04173
04	0,00000	0,00000	0,00000	0,00000	0,00000
05	0,00000	0,00000	0,00000	0,00000	0,00000
06	0,00000	0,00000	0,00000	0,00000	0,00000
07	0,00000	0,00000	0,00000	0,00000	0,00000
08	0,00000	0,00000	0,00000	0,00000	0,00000
09	0,00000	0,03545	0,02363	0,01182	0,04727
10	0,00000	0,00000	0,07533	0,00000	0,00000
11	0,00000	0,00970	0,02908	0,01939	0,03878
12	0,00000	0,00000	0,00000	0,00000	0,00000
13	0,00000	0,00000	0,00000	0,00000	0,05072
14	0,00000	0,00000	0,00000	0,00000	0,00000
15	0,00000	0,00000	0,06794	0,00000	0,00000
16	0,00000	0,00000	0,00000	0,00000	0,00000
17	0,00000	0,04936	0,03291	0,01645	0,06582
18	0,00000	0,00000	0,00000	0,00000	0,00000
19	0,00000	0,04447	0,02965	0,01482	0,05929
20	0,00000	0,00000	0,06192	0,00000	0,00000
21	0,00000	0,00000	0,00000	0,00000	0,00000
22	0,00000	0,00000	0,00000	0,00000	0,00000
23	0,00000	0,00000	0,00000	0,00000	0,00000
24	0,00000	0,00000	0,00000	0,00000	0,00000
25	0,00389	0,01556	0,00778	0,02334	0,03112
26	0,00000	0,02419	0,03628	0,01209	0,04838
27	0,00520	0,03116	0,01039	0,02077	0,04154
28	0,00000	0,00905	0,01810	0,02716	0,03621
29	0,00000	0,03040	0,01013	0,02027	0,04054
30	0,00000	0,04405	0,02937	0,01468	0,05874
Total	0,02277	0,38429	0,49028	0,23088	0,65089

Tabela B.11: Pontuação dos planificadores por problema para o domínio Mprime.

Problema	Petriplan	BLACKBOX	HSP	IPP	STAN
01	0,00000	0,03615	0,04820	0,01205	0,02410
02	0,00000	0,04512	0,03008	0,01504	0,06016
03	0,00000	0,04724	0,03543	0,02362	0,00000
04	0,00000	0,03281	0,04375	0,02187	0,00000
05	0,00000	0,00000	0,00000	0,03646	0,04861
06	0,00000	0,00000	0,08638	0,00000	0,00000
07	0,00000	0,00000	0,00000	0,03503	0,04671
08	0,00000	0,06291	0,04718	0,03145	0,00000
09	0,00000	0,03730	0,02487	0,01243	0,04973
10	0,00000	0,00000	0,08244	0,00000	0,00000
11	0,00000	0,03656	0,04874	0,02437	0,00000
12	0,00000	0,05141	0,03856	0,02571	0,00000
13	0,00000	0,00000	0,00000	0,00000	0,00000
14	0,00000	0,00000	0,00000	0,00000	0,00000
15	0,00000	0,00000	0,08008	0,00000	0,00000
16	0,00000	0,05563	0,04172	0,00000	0,00000
17	0,00000	0,07150	0,05363	0,03575	0,00000
18	0,00000	0,00000	0,00000	0,00000	0,00000
19	0,00000	0,00000	0,06996	0,00000	0,00000
20	0,00000	0,00000	0,08153	0,00000	0,00000
21	0,00000	0,00000	0,00000	0,00000	0,04575
22	0,00000	0,00000	0,08312	0,00000	0,00000
23	0,00000	0,00000	0,00000	0,00000	0,00000
24	0,00000	0,00000	0,08132	0,00000	0,00000
25	0,00000	0,03918	0,02938	0,00979	0,01959
26	0,00000	0,03933	0,05244	0,02622	0,00000
27	0,00000	0,05880	0,04410	0,02940	0,00000
28	0,00000	0,03708	0,04944	0,01236	0,02472
29	0,00000	0,05245	0,03934	0,02622	0,00000
30	0,00000	0,00000	0,08870	0,00000	0,00000
Total	0,00000	0,70347	1,28038	0,37779	0,31937

Tabela B.12: Pontuação dos planificadores por problema para o domínio Logistics.

Problema	Petriplan	BLACKBOX	HSP	IPP	STAN
01	0,00486	0,01946	0,00973	0,02919	0,03892
02	0,00504	0,02013	0,01007	0,03020	0,04026
03	0,00000	0,03984	0,02656	0,05312	0,01328
04	0,00000	0,00000	0,06442	0,00000	0,00000
05	0,00000	0,00000	0,06672	0,05004	0,00000
Total	0,00990	0,07943	0,17750	0,16255	0,09246

Tabela B.13: Pontuação dos planificadores por problema para o domínio Mprime.

Problema	Petriplan	BLACKBOX	HSP	IPP	STAN
01	0,00000	0,04896	0,02448	0,01224	0,03672
02	0,00000	0,03861	0,05148	0,02574	0,01287
03	0,00000	0,00000	0,00000	0,00000	0,00000
04	0,00000	0,03856	0,02570	0,01285	0,05141
05	0,00000	0,04537	0,03403	0,01134	0,02269
Total	0,00000	0,17149	0,13569	0,06217	0,12368

Tabela B.14: Pontuação dos planificadores por problema para o domínio Grid.

Problema	Petriplan	BLACKBOX	HSP	IPP	STAN
01	0,00000	0,02733	0,00000	0,04100	0,05467
02	0,00000	0,00000	0,00000	0,06209	0,00000
03	0,00000	0,00000	0,00000	0,00000	0,00000
04	0,00000	0,00000	0,00000	0,07396	0,00000
05	0,00000	0,00000	0,00000	0,00000	0,00000
Total	0,00000	0,02733	0,00000	0,17705	0,05467