JULIANA HOFFMANN QUIÑÓNEZ BENACCHIO

PLANEJAMENTO EM INTELIGÊNCIA ARTIFICIAL UTILIZANDO REDES DE PETRI CÍCLICAS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Luis Allan Künzle

CURITIBA

2008

JULIANA HOFFMANN QUIÑÓNEZ BENACCHIO

PLANEJAMENTO EM INTELIGÊNCIA ARTIFICIAL UTILIZANDO REDES DE PETRI CÍCLICAS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Luis Allan Künzle

CURITIBA

2008

AGRADECIMENTOS

Ao meu orientador Luis Allan pelo apoio, supervisão, confiança e paciência. Ao professor Fabiano, que mais que um co-orientador, foi responsável por meu ingresso no programa de mestrado, e que juntamente com o professor Marcos colaboraram com sugestões e discussões levantadas ao longo do período de pesquisa as quais foram fundamentais na elaboração e consolidação do trabalho.

Em especial agradeço ao meu marido Alcione, que esteve ao meu lado em todos os momentos de dificuldade e que me deu forças nos momentos de fraqueza para que eu não desistisse e continuasse na conclusão de mais esta etapa de nossas vidas.

A minha mãe e meus irmãos pelo carinho e compreensão durante este período já que muitas vezes deixei de estar presente em vários momentos.

Aos novos amigos que conheci durante o mestrado, pela amizade, companheirismo e momentos de descontração que ajudaram a enfrentar as dificuldades de estar em um ambiente novo.

Finalmente, agradeço a Deus, que me permitiu encontrar todas essas pessoas ao longo do meu caminho.

SUMÁRIO

LISTA DE FIGURAS

RESUMO

Em geral, a principal motivação no desenvolvimento de planejadores é a forma de representar o espaço de busca do problema. As redes de Petri têm sido utilizadas como estrutura de representação para resolver problemas de planejamento, mas o desempenho de planejadores baseados nessa representação não tem sido satisfatório quando comparado à outras abordagens. O relacionamento entre problemas de planejamento clássico em inteligência artificial e problemas de alcançabilidade em rede de Petri é o foco deste trabalho, dando continuidade aos trabalhos desenvolvidos dentro do grupo de pesquisa do Laboratório de Inteligência Artificial e Métodos Formais da Universidade Federal do Paraná. São propostas regras de tradução de um problema de planejamento descrito em *PDDL* para redes de Petri cíclicas e seguras. Com a utilização da técnica de desdobramento de redes de Petri, eficiente para solucionar problemas de alcançabilidade, a abordagem obtém planos que resolvem o problema de planejamento de forma satisfatória.

ABSTRACT

The way to represent the search space of a planning problem, in general, is the main motivation in the development of planners. Petri nets have been used as structure of representation to solve planning problems, but the performance of planners based on that representation has not been satisfactory when compared to other approaches. The relationship between artificial intelligence planning problems and Petri net reachability problems is the main focus of this research. In continuation of works within the Artificial Intelligence and Formal Methods Laboratory, part of the Federal University of Paraná, are proposed translation rules of a planning problem in PDDL into a safe and cyclic Petri net. Using Petri net unfolding, an efficient reachability analysis technique, the approach produce plans that solve planning problems satisfactorily.

CAPÍTULO 1

INTRODUÇÃO

Freqüentemente enfrentamos situações em que nos deparamos com a necessidade de planejar ações para que nossas metas possam ser alcançadas. Essa habilidade de planejar tarefas é considerada um dos aspectos fundamentais do comportamento inteligente.

Os princípios do comportamento inteligente podem ser definidos como atividades que somente um ser humano seria capaz de efetuar, entre elas, tarefas que envolvem raciocínio e percepção, como reconhecimento de imagens, aprendizagem, processamento de linguagem natural e planejamento.

Por mais de quarenta anos, o processo de automatização de planejamento tem sido um dos principais objetivos da pesquisa realizada na área de Inteligência Artificial. Sabendo que a complexidade para resolver problemas de planejamento é PSPACE-Completo [6], os avanços aconteceram em função da representação e eficiência dos algoritmos.

Na década de 90, o algoritmo *GRAPHPLAN* [4] inovou ao apresentar um grafo, chamado grafo de planos, como estrutura de representação que compactava o espaço de busca para o problema. A partir do grafo de planos, houve uma grande motivação por novas pesquisas na área. Diversos trabalhos trouxeram outros métodos de resolução, tornado a área de pesquisa dinâmica e multidisciplinar.

A utilização de redes de Petri como ferramenta para resolver problemas de planejamento vem sendo foco de estudos no grupo de pesquisa do Laboratório de Inteligência Artificial e Métodos Formais (LIAMF) da Universidade Federal do Paraná, desde 2000.

As redes de Petri são um formalismo matemático para modelar, analisar e projetar sistemas de processamento de informação com características concorrentes, assíncronas, distribuídas, paralelas e estocásticas [35]. Variações de redes de Petri são usadas em diversas áreas, como administração [36, 43], engenharia [19, 52] e computação [1].

Em 2000, Silva e colegas apresentaram o *Petriplan* [46], com a proposta de traduzir

um problema de planejamento em um problema de alcançabilidade em redes de Petri. A transformação direta entre os dois problemas é equivalente, uma vez que resolver alcançabilidade em redes de Petri limitadas também é PSPACE-Completo [24]. A demonstração dessa equivalência foi formalizada por Silva [45].

Nessa abordagem o grafo de planos do problema de planejamento é transformado em uma rede de Petri acíclica. O problema de alcançabilidade na rede acíclica é resolvido por meio de técnicas de programação inteira, fundamentada pelo fato de a equação fundamental ser uma condição necessária e suficiente [35].

A rede de Petri gerada pelo *Petriplan* serviu de base para outros trabalhos que apresentaram diferentes soluções para resolver o problema de alcançabilidade ou tratar simplificações na rede. O próprio algoritmo do *Petriplan* teve cinco versões, uma delas resultando em um novo trabalho de Silva, a Rede de Planos [9], um modelo para representar o espaço de estados do problema de forma mais eficiente que o grafo de planos e aproveitar melhor o poder de representação das redes de Petri.

Em geral, os modelos propostos não obtiveram desempenho satisfatório quando comparados aos planejadores existentes. Embora as simplificações para reduzir o tamanho da rede obtivessem um espaço de busca menor para o problema de alcançabilidade, os métodos para resolução disponíveis na época eram ineficientes para as redes obtidas.

O próximo passo das pesquisas foi o direcionamento para a utilização de redes de Petri cíclicas e a exploração de técnicas mais eficientes para resolver problemas de alcançabilidade.

Este documento trata da utilização de redes de Petri cíclicas para representar os problemas de planejamento em inteligência artificial. O objetivo principal do trabalho consistiu em estudar a técnica de desdobramento, utilizada para análise de redes de Petri e propor soluções para sua utilização nos modelos desenvolvidos pela equipe de pesquisa.

Os capítulos estão organizados da seguinte forma: o capítulo 2 é dedicado ao problema de planejamento em inteligência artificial, mais especificamente ao planejamento clássico e suas restrições. O objetivo do capítulo 3 é descrever o formalismo das redes de Petri, bem como sua representação gráfica e notação matricial. Também são apresentados alguns

métodos para resolução do problema de alcançabilidade com destaque para a técnica de desdobramento.

No capítulo 4, é feita uma revisão bibliográfica dos trabalhos relacionados com os problemas de planejamento clássico em inteligência artificial e a representação utilizando redes de Petri. São descritos o planejador *Petriplan*, o modelo da Rede de Planos e a abordagem que utiliza desdobramento de redes de Petri.

O capítulo 5 apresenta uma proposta para problemas de planejamento, utilizando uma rede de Petri cíclica como estrutura de representação e desdobramento para resolver o problema de alcançabilidade. O capítulo 6 mostra uma análise dos resultados obtidos. E finalmente, no capítulo 7, são apresentadas as conclusões e propostas de trabalhos futuros.

CAPÍTULO 2

PLANEJAMENTO EM INTELIGÊNCIA ARTIFICIAL

Este capítulo fornece uma visão geral da área de Planejamento em Inteligência Artificial e apresenta as definições básicas e conceitos relacionados aos problemas de planejamento na seção 2.1. A seção 2.2 é dedicada ao planejamento clássico e suas restrições. Uma breve descrição da representação *STRIPS* é apresentada na seção 2.3 e detalhes sobre a linguagem *PDDL* podem ser vistos na seção 2.4. Na seção 2.5, são relatadas algumas abordagens para resolução de problemas de planejamento clássico. E, finalmente, na seção 2.6, são descritas heurísticas de planejamento.

2.1 O problema de planejamento

O problema de planejamento está entre as áreas de maior interesse em Inteligência Artificial, pois inicialmente integrou duas de suas principais áreas: a busca e a prova de teoremas [42]. Motivações por novas pesquisas trouxeram outros métodos de resolução, entre eles os baseados em funções heurísticas, satisfatibilidade, programação por restrições, programação inteira, redes de Petri, entre outros.

Um problema de planejamento pode modelar e resolver diversas situações do mundo real como, por exemplo, atividades de logística. O problema de transportar cargas entre diferentes localidades necessita de tarefas ordenadas para atender um determinado pedido de entrega, tais como, carregar o caminhão, dirigir da localidade origem para a localidade destino e descarregar o caminhão.

A resolução de um problema de planejamento é o processo de se encontrar uma seqüência de ações para transformar uma situação atual em outra desejada.

Para demonstrar esse conceito, um problema do mundo real tende a ser muito amplo e nem sempre apresenta uma única descrição consensual, o que dificulta a análise. Por isso são criados problemas de exemplo, mais simplificados, concisos e fáceis de se ilustrar,

chamados de miniproblemas (do inglês *toy problems*) [42].

A anomalia de Sussman do mundo dos blocos é um miniproblema clássico de planejamento, descrito como um ambiente no qual existem três blocos sobre uma superfície plana, nomeados como `A`, `B` e `C`, que em seu estado inicial estão dispostos conforme ilustra a figura 2.1(a), e o objetivo é alcançar a situação representada na figura 2.1(b).

(a) Estado inicial (b) Estado final

Figura 2.1: Anomalia de Sussman

Além dos estados iniciais e finais, um problema deve possuir um conjunto de premissas que descrevem o ambiente e que representam o domínio do problema. A definição de um domínio é descrita basicamente pelas ações que podem ser executadas. Para o domínio do mundo dos blocos descrito acima, as duas ações disponíveis são empilhar e desempilhar.

A seqüência de ações: desempilhar `C` de cima de `A`; empilhar `B` sobre `C`; e empilhar `A` sobre `B` é uma solução para o problema, pois alcança o estado desejado. Essa seqüência, ou qualquer outra que encontre o objetivo, é chamada de plano.

É possível notar que um plano apresenta uma relação de ordem entre as ações que o compõem, portanto, considerando a ordenação das ações, pode-se classificar um plano como parcialmente ou completamente ordenado.

Um plano parcialmente ordenado é aquele em que apenas parte das ações estão ordenadas, garantindo apenas ordenações essenciais. Um plano completamente ordenado é aquele em que todas as ações estão ordenadas, impondo uma seqüência cronológica, partindo do estado inicial e chegando ao estado objetivo. No exemplo, o plano encontrado é totalmente ordenado.

O sistema projetado para encontrar o plano é chamado de planejador e recebe como entrada uma descrição do estado inicial do mundo, uma descrição do objetivo desejado e um conjunto das possíveis ações, todos codificados em uma linguagem de descrição formal.

Algumas das linguagens utilizadas, como *STRIPS*, *ADL* e *PDDL*, serão apresentadas nas seções 2.3 e 2.4.

De acordo com a descrição do domínio dos problemas, a área de abrangência dos planejadores se divide em duas categorias: planejamento clássico e planejamento não clássico. No primeiro, os ambientes são totalmente conhecidos, finitos, estáticos e discretos no tempo e nas ações. Por outro lado, no planejamento não clássico, o planejador se depara com situações imprevisíveis que podem ocorrer devido a fatores externos. Problemas de planejamento não clássico não fazem parte do escopo do trabalho.

2.2 Planejamento clássico

Existem diversos fatores que dificultam o processo de automatização do planejamento. Buscando simplificar esse processo, a abordagem clássica apresenta as seguintes restrições para o domínio, problema e planejador [16]:

- **Finito:** O ambiente deve possuir um conjunto finito de estados;
- **Totalmente conhecido:** O ambiente deve ser completamente observável;
- **Determinístico:** Se uma ação é aplicável a um estado, ela resulta em um único outro estado;
- **Estático:** Não possui dinâmica interna, ou seja, deve ficar no mesmo estado até que uma ação seja aplicada;
- **Objetivos restritos:** Os objetivos são especificados como um estado objetivo explícito ou como um conjunto de estados objetivo;
- **Planos seqüenciais:** A solução para o problema deve ser uma seqüência de ações finita e linearmente ordenada;
- **Ações discretas:** As ações não têm duração, ou seja, são transições de estado instantâneas;
- **Não influenciável:** O planejador não está relacionado com qualquer mudança externa que possa ocorrer no ambiente enquanto estiver planejando.

A dinâmica de um ambiente de planejamento envolvendo ações determinísticas e informação completa pode ser especificada por meio de um modelo de estados. Esse modelo consiste de um número finito de estados $\mathcal{S}$, um conjunto finito de ações $\mathcal{A}$, que podem ser executadas, e uma função de transição de estados γ, que descreve como uma ação aplicada permite passar de um estado a outro [16].

Um modelo de estados pode ser representado por um grafo orientado, no qual os vértices correspondem aos estados, e os arcos, rotulados com as ações, representam todas as possíveis transições de estados do mundo.

Por exemplo, o grafo do espaço de estados para o domínio do mundo dos blocos com três blocos `A`, `B` e `C` é ilustrado na figura 2.2.

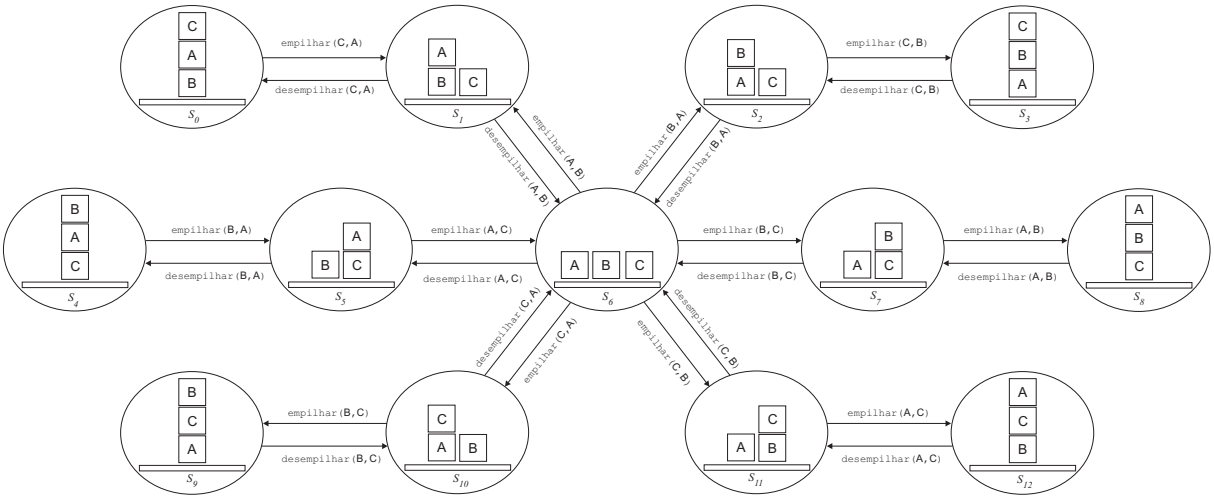


Figura 2.2: Grafo do espaço de estados para o domínio mundo dos blocos

Embora a suposição de determinismo feita na abordagem clássica realmente simplifique a automatização da tarefa de planejamento, a complexidade para os problemas de planejamento clássico é PSPACE-Completo [6].

2.3 Representação *STRIPS*

Até o início da década de 1970, os problemas de planejamento eram resolvidos por meio de prova de teoremas e lógica clássica [18]. A primeira proposta de solução integrando

um sistema formal e um algoritmo correspondente foi apresentado por Fikes e Nilsson em 1971. O sistema *STRIPS* (*STanford Research Institute Problem Solver*) [13] é composto por um modelo formal para representar as ações e os estados do mundo e um algoritmo que trata o problema como uma busca no espaço de estados.

A representação *STRIPS* é derivada do cálculo proposicional e usa uma representação de literais de primeira ordem, que devem ser básicos e livres de funções. Um problema de planejamento escrito em *STRIPS* é dividido em duas partes: a descrição do domínio e a descrição do problema.

O domínio contém o conjunto de literais e a descrição das ações. Para o domínio do mundo dos blocos descrito na seção 2.1, os estados são representados pelos literais:

- sobre(x,y) - indica que o bloco x está em cima do bloco y;
- sobre-a-mesa(x) - indica que o bloco x está em cima da mesa;
- livre(x) - indica que não há nada sobre o bloco x;

A definição de uma ação é formada por três partes: o nome da ação e uma lista de parâmetros; a pré-condição, que é uma conjunção de literais positivos que devem ser verdadeiros para que a ação possa ser executada; e o efeito, que é uma conjunção de literais positivos ou negativos que descrevem as alterações nos estados após a execução da ação.

É possivel dividir o efeito em uma lista de adição (add) para os literais positivos e uma lista de eliminação (del) para os literais negativos. Uma ação pode ser definida então como:

nome da ação(parâmetros)
pre: lista das pré-condições
add: lista dos efeitos adicionados (literais positivos)
del: lista dos efeitos removidos (literais negativos)

Por exemplo, a representação em *STRIPS* da ação desempilhar do domínio do mundo dos blocos pode ser descrita por:

desempilhar(x,y)

pre: sobre(x,y) $\wedge$ livre(x)

add: livre(y) $\wedge$ sobre-a-mesa(x)

del: sobre(x,y)

É importante destacar que na verdade essa é uma descrição genérica da ação, denominada esquema de ação, que será instanciada pelas constantes do problema, gerando várias outras ações.

A descrição do problema deve conter o conjunto de constantes presentes no problema, o estado inicial do mundo e o objetivo do problema. Os estados do mundo são representados por conjunções de literais instanciados. Por exemplo, o estado final da anomalia de Sussman, visto na figura 2.1(b), é representado em *STRIPS* da seguinte forma:

$$\text{sobre(A,B)} \wedge \text{sobre(B,C)}$$

De acordo com a hipótese do mundo fechado [41], literais não mencionados em um estado são assumidos como falsos. Portanto, para representar o estado inicial da anomalia de Sussman, todos os estados devem ser listados, e literais negativos podem ser retirados da descrição inicial, como, por exemplo, a proposição[1] $\neg$ livre(A). O estado inicial visto na figura 2.1(a) é descrito em *STRIPS* como:

$$s_i\text{: sobre(C,A)} \wedge \text{sobre-a-mesa(A)} \wedge \text{sobre-a-mesa(B)} \wedge \text{livre(B)} \wedge \text{livre(C)}$$

A partir do estado inicial do problema, s_i, é possível executar a ação desempilhar(C,A), pois as pré-condições sobre(C,A) $\wedge$ livre(C) são atendidas. O novo estado será resultante da remoção do literal sobre(C,A) e da adição dos literais livre(A) $\wedge$ sobre-a-mesa(C). O estado após a execução da ação desempilhar(C,A) é representado como:

$$s_1\text{: sobre-a-mesa(A)} \wedge \text{sobre-a-mesa(B)} \wedge \text{sobre-a-mesa(C)} \wedge \text{livre(A)} \wedge \text{livre(B)} \wedge \text{livre(C)}$$

[1]Neste texto os termos "literal" e "proposição" são usados como sinônimos.

Como pôde ser visto, a linguagem formal proposta pela representação *STRIPS* permite representar ações e estados de maneira simples. A independência entre a linguagem que descreve o problema e o algoritmo que resolve o problema de planejamento é uma das vantagens desta representação. Mas o custo desta simplicidade e independência é uma linguagem com muitas restrições, entre elas: não suporta predicados de igualdade, não é uma linguagem tipada, além de não ser lógica [42].

Devido a essas e outras limitações, surgiu a necessidade de uma linguagem mais expressiva, na qual fosse possível descrever uma gama maior de problemas. Entre diversas criações e derivações, merece destaque a Linguagem de Descrição de Ação ou *ADL* (*Action Description Language*) [37].

A linguagem *ADL* apresenta melhorias para representar os problemas. Comparada com a linguagem *STRIPS*, podemos destacar a inclusão de efeitos condicionais, a permissão de disjunção nos objetivos, suporte a variáveis tipadas e assume a hipótese do mundo aberto em que literais que não são mencionados são desconhecidos.

2.4 Linguagem *PDDL*

A Linguagem de Definição de Domínio de Planejamento ou *PDDL* (*Planning Domain Definition Language*) [32] é uma combinação das linguagens *STRIPS* e *ADL* com tarefas hierárquicas, recursos, tempo e metas com otimizações.

Foi desenvolvida especialmente para a primeira competição de planejadores realizada durante o congresso internacional *AIPS* (*Artificial Intelligence Planning and Scheduling Systems*), em 1998, com o objetivo de ser uma especificação padrão para representar os problemas de planejamento e permitir assim uma comparação justa e precisa entre os planejadores da competição.

Como nem todos os planejadores tratam determinados aspectos de uma descrição, a linguagem *PDDL* apresenta um sub-conjunto de características, denominados requisitos, que possibilitam agrupar os planejadores de acordo com suas particularidades. Alguns requisitos que podemos destacar são: ações descritas no formato da linguagem *STRIPS*, ações que necessitam das extensões da linguagem *ADL* e utilização de variáveis tipadas.

Para a representação de um problema de planejamento em *PDDL*, são necessários dois arquivos, o arquivo de domínio e o arquivo de problema, conforme ilustrado na figura 2.3.

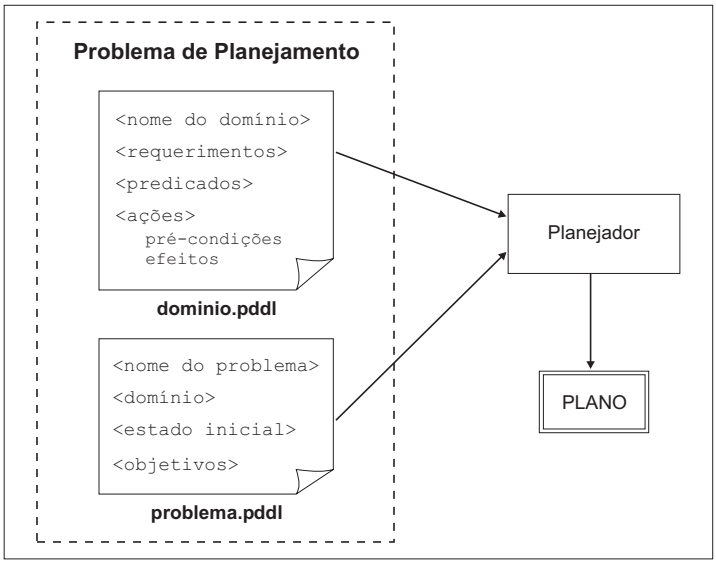


Figura 2.3: Estrutura básica de um problema de planejamento em *PDDL*

Os arquivos são exemplificados por meio do problema anomalia de Sussman do domínio mundo dos blocos, descrito na seção 2.1. No arquivo de domínio, demonstrado na figura 2.4, são especificados o nome do domínio, os requisitos da linguagem, os predicados existentes e as ações disponíveis.

```
(define (domain MUNDO_DOS_BLOCOS)
  (:requirements :strips :typing)

  (:predicates (sobre ?x - bloco ?y - bloco)
               (sobre-a-mesa ?x - bloco)
               (livre ?x - bloco))

  (:action empilhar
       :parameters (?x - bloco ?y - bloco)
       :precondition (and (livre ?x) (sobre-a-mesa ?x) (livre ?y))
       :effect (and  (not (livre ?y))
               (not (sobre-a-mesa ?x))
               (sobre ?x ?y)))

  (:action desempilhar
       :parameters (?x - bloco ?y - bloco)
       :precondition (and (sobre ?x ?y) (livre ?x))
       :effect (and (livre ?y)
               (livre ?x)
               (sobre-a-mesa ?x)
               (not (sobre ?x ?y)))))
```

Figura 2.4: Domínio mundo dos blocos em *PDDL*

O arquivo para descrição do problema, apresentado na figura 2.5, contém o nome do problema, o domínio, os objetos, o estado inicial e o objetivo.

```
(define (problem ANOMALIA_DE_SUSSMAN)
  (:domain MUNDO_DOS_BLOCOS)
  (:objects A B C - bloco)
  (:INIT (sobre-a-mesa A)
         (sobre-a-mesa B)
         (livre B)
         (livre C)
         (sobre C A))
  (:GOAL (AND (sobre A B)
         (sobre B C)
         (sobre-a-mesa C))))
```

Figura 2.5: Problema anomalia de Sussman em *PDDL*

A Competição Internacional de Planejadores *IPC (International Planning Competition)* acontece a cada dois anos em paralelo com a Conferência Internacional de Planejamento e Escalonamento *ICAPS (International Conference on Planning and Scheduling)*. Para cada competição novas funcionalidades e extensões são adicionadas à linguagem *PDDL* criando novas versões.

Na competição realizada em 2008, utilizou-se a versão *PDDL3.1* [14], estendida em relação à qualidade da solução por meio de métricas de plano e custo para as ações[2]. Entretanto, os problemas de planejamento abordados neste trabalho estão escritos na versão básica do *PDDL* utilizando os requisitos `:strips` e `:typing`.

2.5 Abordagens para planejamento clássico

Como já mencionado, desde o surgimento do sistema *STRIPS*, em 1971, até o início da década de 1990, os planejadores tratavam os problemas com procedimentos de busca no espaço de estados. Em 1992, o planejador *UCPOP* [38] partiu para uma busca no espaço de planos, no qual se tentava construir um plano a partir de operadores de planos.

Também em 1992, Kautz e Selman propuseram a tradução da representação *STRIPS* para o cálculo proposicional e desenvolveram o planejador *SATPLAN* [25]. O processo de tradução do problema de planejamento a um problema de se resolver uma instância *SAT* tirou proveito do estado da arte de métodos para *SAT*, resultando em um planejador com excelentes resultados.

Três anos depois, Blum e Furst apresentaram o *GRAPHPLAN* [4], que a partir de uma descrição STRIPS constrói um grafo que compacta o espaço de busca para o problema de

[2]Disponível em: <http://ipc.informatik.uni-freiburg.de/PddlExtension> Acesso em: 20 set. 2008

planejamento.

O planejador *HSP* [5] surgiu em 1998, utilizando uma função heurística proposta por Bonet e Geffner, que guiava um procedimento de busca de subida de encosta no algoritmo. Funções heurísticas eficientes também foram utilizadas nos planejadores *FF* [23] e *LPG* [15], que obtiveram respectivamente os melhores desempenhos nas competições de 2000 e 2002.

Seguindo a linha de planejadores baseados no grafo de planos, o *BLACKBOX* [26] gera em sua primeira fase o grafo de planos que depois é traduzido em uma instância *SAT*, que então é resolvida por um resolvedor *SAT*.

Outras técnicas como a programação inteira [50] e a técnica de satisfação de restrições *CSP (Constraint Satisfaction Problem)* [3] também foram utilizadas como abordagens na resolução de problemas de planejamento.

Entre as abordagens relatadas, o planejador *GRAPHPLAN* merece destaque pela estrutura grafo de planos, muito utilizada por outros planejadores, pela influência nos trabalhos realizados pelo grupo de pesquisa nos últimos anos e principalmente por ter sido base do planejador *Petriplan* que será abordado no capítulo de trabalhos relacionados.

O planejador *GRAPHPLAN*, desenvolvido por Blum e Furst [4], inovou ao apresentar um algoritmo simples e uma estrutura capaz de compactar a representação do espaço de busca do problema. Essa estrutura foi chamada de grafo de planos.

O grafo de planos é composto por dois tipos de nós, distribuídos em camadas, sendo que os nós proposições permanecem nas camadas pares, e os nós ações são distribuídos nas camadas ímpares. As camadas de proposição do grafo representam os estados do mundo, e as camadas de nós ações representam as ações que transformam um estado em outro.

Os nós proposições e os nós ações são conectados por arestas, que podem ser classificadas como: arestas proposição-ação, que conectam os nós proposições aos nós ações da camada posterior, ligando as pré-condições das ações às proposições correspondentes; arestas ação-proposição, que conectam os nós ações aos nós proposições da camada posterior, ligando os efeitos das ações às proposições correspondentes da camada seguinte; arestas de manutenção, que conectam dois nós proposições e têm como objetivo garantir

que na camada `(n+2)` estarão as proposições da camada `n`; e arestas de exclusão mútua, que representam as relações de exclusão existentes entre ações ou proposições.

A figura 2.6 apresenta um grafo de planos com três camadas, no qual os nós proposições são representados por círculos; os nós ações, por retângulos; as arestas de manutenção, por linhas pontilhadas, e as arestas de exclusão mútua são representadas por uma linha grossa.

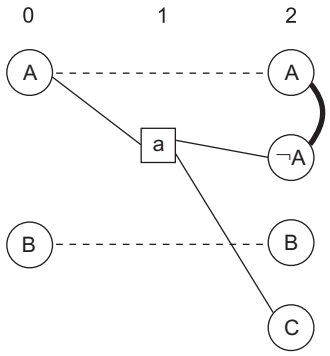


Figura 2.6: Grafo de planos de três camadas

O algoritmo do *GRAPHPLAN* é constituído de duas fases: expansão do grafo de planos e extração da solução. A cada passo o algoritmo executa essas duas fases sucessivamente até que uma solução seja alcançada.

A expansão do grafo é feita até que a condição de parada seja satisfeita, ou seja, encontrar na última camada do grafo todas as proposições do estado objetivo, sendo que essas não podem estar em conflito. Essa condição não garante que exista uma solução para o problema, mas a fase de extração da solução deve ser realizada. O objetivo é verificar se a partir dessa última camada é possível obter um plano que seja solução do problema por meio de uma busca regressiva.

Caso uma solução não seja encontrada, deve-se voltar à fase de expansão e depois novamente à fase de extração da solução. Quando as duas últimas camadas ímpares do grafo forem idênticas, pode-se chegar à conclusão de que não existe solução para o problema.

2.6 Heurísticas de planejamento

A maneira mais simples para se resolver um problema de planejamento é realizando uma busca exaustiva no espaço de estados e encontrar um caminho nesse grafo que leva do estado inicial a um estado objetivo. Infelizmente, essa estratégia é ineficiente, pois, dependendo do tamanho do problema, o espaço de estados pode se tornar um espaço de busca exponencial, inviabilizando qualquer solução em termos de tempo e memória.

Uma estratégia de busca é dita cega se ela não leva em conta informações específicas sobre o problema a ser resolvido. Introduzindo certas medidas heurísticas, é possível desenvolver uma busca mais otimizada, direcionando o processo e diminuindo as possibilidades a cada passo.

O componente fundamental para direcionar o procedimento de busca é uma medida de qualidade para cada nó, de maneira que o nó com a melhor avaliação seja sempre o escolhido. Uma boa medida seria o custo da melhor solução que passa pelo nó. Podemos chamar de função heurística, denotada por $h(n)$, o custo estimado da solução de custo mais baixo passando por n [42].

Durante a busca, um nó é selecionado para expansão com base em uma função de avaliação $f(n)$, que avalia os nós combinando $g(n)$, o custo do menor caminho do nó inicial até o nó em questão, e $h(n)$, o custo para ir do nó até o objetivo:

$$f(n) = g(n) + h(n)$$

A questão é que tal função tem a mesma complexidade de resolver o problema, uma vez que, sabendo o custo de um caminho ótimo no espaço de estados que leva do estado inicial ao estado final, o caminho também é conhecido.

É preciso determinar uma estimativa para o verdadeiro valor da função, ou seja, uma cota inferior para a função de custo ótimo, assim ela pode ser considerada uma heurística admissível, isto é, que não superestima o custo ótimo para alcançar o objetivo.

Definição 1 (Heurística admissível) *Uma heurística é admissível se, para cada nó, o valor retornado por esta heurística nunca ultrapassa o custo real do melhor caminho desse nó até o objetivo [42].*

Heurísticas admissíveis são naturalmente otimistas, pois estimam que o custo da solução do problema é menor do que ele realmente é. Tendo em vista que $g(n)$ é o custo exato para se alcançar n, tem-se como consequência imediata que $f(n)$ nunca irá superestimar o custo verdadeiro de uma solução passando por n.

Deve-se assegurar também que o caminho ótimo para qualquer estado repetido é sempre o primeiro a ser seguido. Essa propriedade é mantida por meio do requisito de consistência ou monotonicidade.

Definição 2 (Heurística monotônica) *Uma heurística $h(n)$ é monotônica se, para todo nó n e todo sucessor n' de n gerado por qualquer ação a, o custo estimado de alcançar o objetivo a partir de n não é maior que o custo do passo de se chegar a n' somado ao custo estimado de alcançar o objetivo a partir de n' [42]:*

$$h(n) \leq c(n, a, n') + h(n')$$

Haslum e Geffner desenvolveram uma família de funções heurísticas admissíveis e polinomiais, definida como h^m para $m = 1, 2, \ldots$, em que à medida que m aumenta, a heurística é mais precisa, porém mais custosa [20].

A heurística é definida por meio da simplificação do problema para uma versão relaxada. A versão original é vista como um problema do caminho mais curto de única origem [10], no qual o objetivo é encontrar um caminho mais curto desde um determinado nó de origem até todos os outros nós do grafo.

A heurística h^m, que fornece uma aproximação admissível por meio do relaxamento, é definida por:

$$h^m(C) = \begin{cases} 0 & \text{se} \quad C \subseteq S_0 \\ \min_{<B,a> \in R(C)}[cost(a) + h^m(B)] & \text{se} \quad |C| \leq m \\ \max_{D \subset C, |D|=m} h^m(D) & \text{se} \quad |C| > m \end{cases} \tag{2.1}$$

O relaxamento aproxima o custo de conjuntos C intratáveis, $|C| > m$, ao custo do subconjunto mais custoso $D \subset C, |D| \leq m$.

2.7 Considerações

Este capítulo apresentou o problema de planejamento, sua descrição como abordagem clássica e algumas propostas de resolução desenvolvidas nos últimos anos. Foi destacada a linguagem *PDDL*, que vem evoluindo juntamente com os planejadores e adicionando funcionalidades a cada nova competição.

Verificou-se que o problema de planejamento clássico é PSPACE-Completo, o que inviabiliza qualquer abordagem exaustiva, motivando pesquisas para tentar reduzir o espaço de busca e propor novas estruturas para representar o problema.

Diversas áreas de conhecimento já foram exploradas, entre elas destacamos as funções heurísticas, o grafo de planos, métodos para *SAT* e programação por restrições. A utilização de redes de Petri como ferramenta para resolver problemas de planejamento é uma das linhas de estudo do grupo de pesquisa em planejamento em inteligência artificial do Laboratório de Inteligência Artificial e Métodos Formais da UFPR.

O próximo capítulo descreve o formalismo das redes de Petri e seu relacionamento entre problemas de planejamento clássico em inteligência artificial e problemas de alcançabilidade em redes de Petri.

CAPÍTULO 3

REDES DE PETRI

Neste capítulo são apresentados os principais conceitos relativos às redes de Petri (RdP), tais como a representação gráfica e o formalismo matemático. A seção 3.4 é dedicada ao problema de alcançabilidade em RdP, um dos focos deste trabalho. São descritas algumas técnicas para se encontrar uma solução para este problema, destacando a técnica de desdobramento (do termo inglês *unfolding*), que tem sido aplicada com sucesso neste tipo de caso.

3.1 Definições

O conceito de redes de Petri foi introduzido por Carl Adam Petri em sua tese de doutorado em 1962 [39], como uma teoria para descrever relações entre condições e eventos.

As redes de Petri são uma ferramenta matemática para modelar, analisar e projetar sistemas a eventos discretos, possibilitando a representação de propriedades como paralelismo, concorrência, controle, conflitos e sincronização [35].

Além do formalismo matemático, as redes de Petri também podem ser representadas graficamente. A representação gráfica consiste de um grafo dirigido, ponderado e bipartido contendo dois tipos de nós, chamados lugares e transições. Os nós são conectados por segmentos orientados chamados arcos. Resumidamente, os elementos da representação gráfica de uma RdP são:

- **Lugares:** representados como círculos, são condições, atividades, recursos ou um estado parcial do sistema;
- **Transições:** desenhadas como barras ou retângulos, representam um evento. Uma transição possui um certo número de lugares de entrada e de saída contectados, chamados respectivamente de pré-condições e pós-condições do evento;

- **Marcas:** representam o estado de um sistema e graficamente são círculos menores dentro dos lugares. A presença de uma marca em um lugar indica que a proposição representada por ele é verdadeira, ou ainda a presença de uma ou mais marcas podem indicar a quantidade de recursos disponíveis;
- **Arcos:** representados como setas, conectam tanto lugares a transições como transições a lugares. Determinam o fluxo das marcas na rede. Cada arco tem um peso, representado por um número inteiro sobre a seta, que determina a quantidade de marcas consumidas ou produzidas pela transição. Caso esse número não exista, o peso do arco é assumido como 1.

Formalmente, uma Rede de Petri pode ser definida como [35]:

$$N = (P, T, F, W, M)$$

sendo que:

$$\begin{aligned}
P = \{p_1, p_2, \ldots, p_m\} &\quad \text{é um conjunto finito de lugares;} \\
T = \{t_1, t_2, \ldots, t_k\} &\quad \text{é um conjunto finito de transições;} \\
F \subseteq (P \times T) \cup (T \times P) &\quad \text{é um conjunto de arcos (relação de fluxo);} \\
W : F \rightarrow \mathbb{N} &\quad \text{é uma função de peso;} \\
M : P \rightarrow \mathbb{N} &\quad \text{é o número de marcas;} \\
P \cap T = \emptyset \text{ e } P \cup T \neq \emptyset &\quad P \text{ e } T \text{ são conjuntos disjuntos.}
\end{aligned}$$

A relação de fluxo F é definida pelo pré-conjunto e pelo pós-conjunto dos lugares e transições da rede. Para $x \in N$, o pré-conjunto de x, denominado $\bullet x$ e o pós-conjunto de x, denominado $x\bullet$, podem ser definidos como:

$$\bullet x = \{y \in P \cup T \mid W(y, x) \geq 1\}$$

$$x\bullet = \{y \in P \cup T \mid W(x, y) \geq 1\}$$

As redes de Petri Lugar/Transição são um tipo de rede que permite mais de uma marca

nos lugares e arcos ponderados. Uma marcação pode ser representada pela quantidade de marcas ou simplesmente pelo número que indica essa quantidade. O peso do arco é indicado por meio de um rótulo, e quando esse valor for unitário, o arco não precisa ser rotulado. A figura 3.1 mostra um exemplo desse tipo de rede.

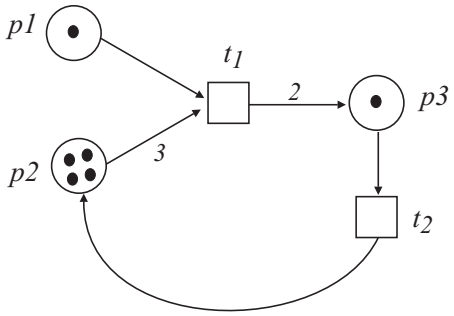


Figura 3.1: Rede de Petri Lugar/Transição

Uma outra classe de rede de Petri muito utilizada é a Condição/Evento, brevemente definida a seguir e ilustrada na figura 3.2, que simula o funcionamento de um semáforo.

Definição 3 (Rede Condição/Evento) *Uma rede é chamada Condição/Evento quando a rede permite no máximo uma marca em cada lugar e todos os seus arcos possuem pesos unitários. Nesta rede as transições representam eventos (E) e os lugares representam condições (B), resultando na rede* $N = (B, E, F)$ *[40].*

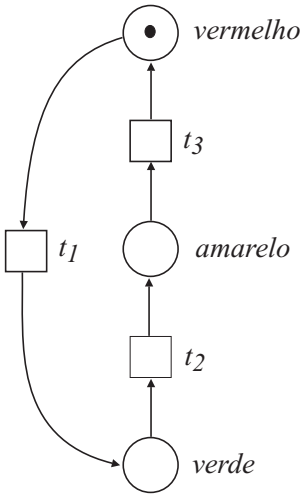


Figura 3.2: Rede de Petri Condição/Evento

O comportamento de um sistema pode ser descrito pelas mudanças em seus estados. Um sistema modelado em uma rede de Petri simula essa dinâmica de acordo com uma regra de disparo de transição, que determina a mudança de estado por meio do consumo e geração de marcas.

Definição 4 (Regra de disparo de transição) *Uma transição t está habilitada, se seus lugares de entrada estiverem marcados com a quantidade de marcas maior ou igual ao valor do peso indicado no arco. Se a transição for disparada, ela remove a quantidade de marcas dos lugares de entrada e adiciona nos lugares de saída a quantidade de marcas igual ao valor do peso do arco entre a transição e os respectivos lugares de saída [35].*

Na rede da figura 3.1, a transição t_1 está habilitada, uma vez que seus lugares de entrada, p_1 e p_2 estão marcados e satisfazem a regra. O resultado do disparo da transição t_1 pode ser visto na figura 3.3.

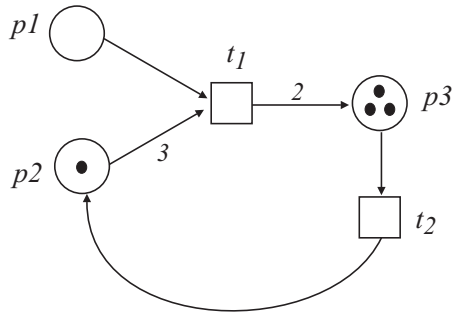


Figura 3.3: Rede da figura 3.1 após o disparo da transição t_1

O disparo da transição t_1 retira três das quatro marcas do lugar p_2, uma marca do lugar p_1 e adiciona duas marcas no lugar de saída p_3.

3.2 Representação matricial

Uma rede de Petri também pode ser definida como um conjunto de matrizes de números naturais, cujo comportamento dinâmico é descrito por um sistema linear [7].

A relação de fluxo F e a função de peso W são agrupadas e armazenadas em matrizes que definem a ligação entre os lugares e as transições da rede.

A matriz de incidência de entrada `Pre` descreve os arcos orientados que ligam lugares a transições. $\texttt{Pre}(p,t)$ representa o peso do arco (p,t). Se $\texttt{Pre}(p,t) = 0$, então não existe um arco entre o lugar p e a transição t.

A matriz de incidência de saída `Pos` descreve os arcos orientados que ligam transições a lugares. $\texttt{Pos}(p,t)$ representa o peso do arco (t,p). Se $\texttt{Pos}(p,t) = 0$, então não existe um arco entre a transição t e o lugar p.

A partir das matrizes `Pre` e `Pos` calcula-se a matriz de incidência `C`, (`C` = `Pos` - `Pre`), que fornece o balanço das marcas na rede quando do disparo das transições.

As próximas definições serão exemplificadas utilizando a rede da figura 3.1. A representação matricial da rede traz as seguintes matrizes:

$$\texttt{Pre} = \begin{array}{c} \begin{array}{cc} t_1 & t_2 \end{array} \\ \left[\begin{array}{cc} 1 & 0 \\ 3 & 0 \\ 0 & 1 \end{array}\right] \end{array} \begin{array}{c} p_1 \\ p_2 \\ p_3 \end{array} \qquad \texttt{Pos} = \begin{array}{c} \begin{array}{cc} t_1 & t_2 \end{array} \\ \left[\begin{array}{cc} 0 & 0 \\ 0 & 1 \\ 2 & 0 \end{array}\right] \end{array} \begin{array}{c} p_1 \\ p_2 \\ p_3 \end{array} \qquad \texttt{C} = \begin{array}{c} \begin{array}{cc} t_1 & t_2 \end{array} \\ \left[\begin{array}{cc} -1 & 0 \\ -3 & 1 \\ 2 & -1 \end{array}\right] \end{array} \begin{array}{c} p_1 \\ p_2 \\ p_3 \end{array}$$

Para representar a marcação da rede utiliza-se um vetor coluna M, cujo tamanho é a quantidade de lugares. $M(p)$ define o número de marcas em um lugar p, assim, $M(p_1) = 1$, $M(p_2) = 4$ e $M(p_3) = 1$. Portanto, a marcação inicial da rede é o vetor transposto $M_0 = [1\ 4\ 1]^T$.

Utilizando a notação matricial para a regra de disparo de transição [35], uma transição t está habilitada se e somente se $\forall p \in P,\ M(p) \geq \texttt{Pre}(p,t)$.

Se a transição for disparada, uma nova marcação será definida por meio da remoção de $\texttt{Pre}(p,t)$ marcas dos lugares de entrada e da adição de $\texttt{Pos}(p,t)$ marcas nos lugares de saída. A nova marcação M' é obtida por meio de: $\forall p \in P,\ M'(p) = M(p) - \texttt{Pre}(p,t) + \texttt{Pos}(p,t)$.

Para efeito de simplificação, vamos utilizar a notação $\texttt{Pre}(.,t)$ para a coluna da matriz `Pre` referente à transição t, e assim da mesma forma para as matrizes `Pos` e `C`. Com isso, a equação para obter a nova marcação pode ser escrita como: $M' = M - \texttt{Pre}(.,t) + \texttt{Pos}(.,t)$ ou ainda $M' = M + \texttt{C}(.,t)$.

Na rede exemplo, a marcação M' obtida após o disparo de t_1 sobre a marcação inicial

M, ($M \xrightarrow{t_1} M'$) é encontrada a partir da equação: $M' = [1\ \ 4\ \ 1]^T + [\text{-}1\ \text{-}3\ \ 2]^T = [0\ \ 1\ \ 3]^T$ e pode ser vista na figura 3.3.

Para generalizar essa fórmula e calcular uma nova marcação após o disparo de uma seqüência de transições é necessário a utilização do vetor $\bar{s}$, chamado vetor característico da seqüência s. Esse vetor é de tamanho igual ao número de transições da rede, sendo $\bar{s}(t)$ o número de vezes que a transição t foi disparada. A equação fundamental das redes de Petri é dada por:

$$M' = M + \mathtt{C}.\bar{s} \tag{3.1}$$

3.3 Propriedades

As propriedades relativas às redes de Petri são divididas entre estruturais, que não dependem da marcação, e comportamentais, que estão relacionadas com as RdP marcadas. Neste trabalho, a atenção é voltada apenas para a propriedade estrutural de aciclicidade e para as propriedades comportamentais de alcançabilidade e limitabilidade.

Definição 5 (Rede de Petri Acíclica) *Uma RdP é acíclica se nenhum subconjunto de seus arcos formam um ciclo direcionado, ou seja, se não contiver uma seqüência de lugares e transições* $\{x_1, x_2, \ldots, x_n, x_1\}$ *para* $n \geq 2$ *tal que todos os pares* (x_1, x_2), (x_2, x_3), $\ldots$, (x_{n-1}, x_n), (x_n, x_1) *correspondam a arcos na rede [35].*

Definição 6 (Rede de Petri Limitada) *Uma RdP é limitada se para toda marcação, o número de marcas em cada lugar da rede é limitado. A rede é dita k-limitada quando a capacidade é limitada a k marcas. Se k = 1, a rede 1-limitada é chamada de rede segura (safe net) [7].*

Definição 7 (Alcançabilidade) *O conjunto de marcações acessíveis* $A(N, M)$ *de uma RdP marcada é o conjunto das marcações alcançáveis a partir da marcação inicial, por meio de uma seqüência de disparo s [7].*

$$A(N, M) = \{M_i\ ,\ \exists s\ M \xrightarrow{s} M_i\}$$

Quando o conjunto de marcações acessíveis é finito, pode ser representado sob a forma de um grafo, denominado grafo de alcançabilidade, no qual os nós são as marcações acessíveis e os arcos são as transições disparadas entre as marcações. A figura 3.4 ilustra todas as possíveis marcações e todos os possíveis disparos em cada marcação para a rede da figura 3.1.

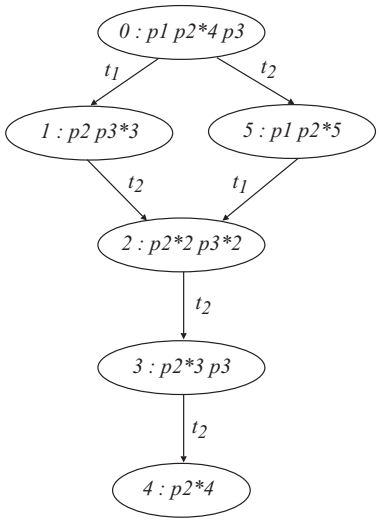


Figura 3.4: Grafo de alcançabilidade da rede da figura 3.1

Saber se uma determinada marcação é alcançável a partir da marcação inicial da rede é um problema relevante na área. Essa verificação e a busca por uma seqüência de transições que leva a marcação inicial até a marcação desejada é definida como o problema de alcançabilidade em redes de Petri, que será aprofundada na seção 3.4.

3.4 O problema de alcançabilidade

A equação fundamental (3.1) pode ser utilizada para determinar uma seqüência de transições s, tal que $M_0 \xrightarrow{s} M$. Porém, a existência de um vetor que atenda a equação não é uma condição necessária para que a marcação M seja realmente alcançada a partir da marcação inicial, uma vez que a ordem de disparo é perdida e a solução encontrada pode trazer vetores $\bar{s}$ que não correspondem a seqüências possíveis de disparos na rede.

O problema de alcançabilidade pode ser definido como o problema de verificar se uma

dada marcação M é alcançável a partir da marcação M_0, ou seja, se:

$$M \in A(N, M_0)$$

É preciso destacar que muitas vezes essa exata marcação que está sendo buscada não será encontrada e sim uma marcação maior que irá contê-la. Chamamos M_s a marcação que está contida em uma marcação M ($M_s \subseteq M$), ou ainda, que $M_s \leq M$. Temos então o problema de alcançabilidade de sub-marcação, definido como o problema de verificar se existe um M tal que:

$$M \in A(N, M_0) \text{ sendo que } M_s \subseteq M$$

O problema de alcançabilidade de sub-marcação é teoricamente equivalente ao problema de alcançabilidade, que se sabe ser um problema decidível usando espaço exponencial [29]. Em relação à complexidade computacional, sabe-se que para resolver alcançabilidade em redes de Petri acíclicas é NP-Completo [48] e, em redes limitadas ou k-limitadas, é PSPACE-Completo [24].

Silva, em seus trabalhos [44, 45], utilizou técnicas de programação inteira sobre a equação fundamental das redes de Petri (3.1) para resolver problemas de alcançabilidade em redes acíclicas. Essa abordagem foi fundamentada no fato de a equação fundamental ser uma condição necessária e suficiente para solucionar alcançabilidade em redes de Petri acíclicas [35].

O grafo de alcançabilidade, descrito na seção 3.3, é utilizado para resolver problemas de alcançabilidade, envolvendo métodos de busca e técnicas heurísticas, mas pode ser usado apenas para redes pequenas devido à explosão do espaço de estados.

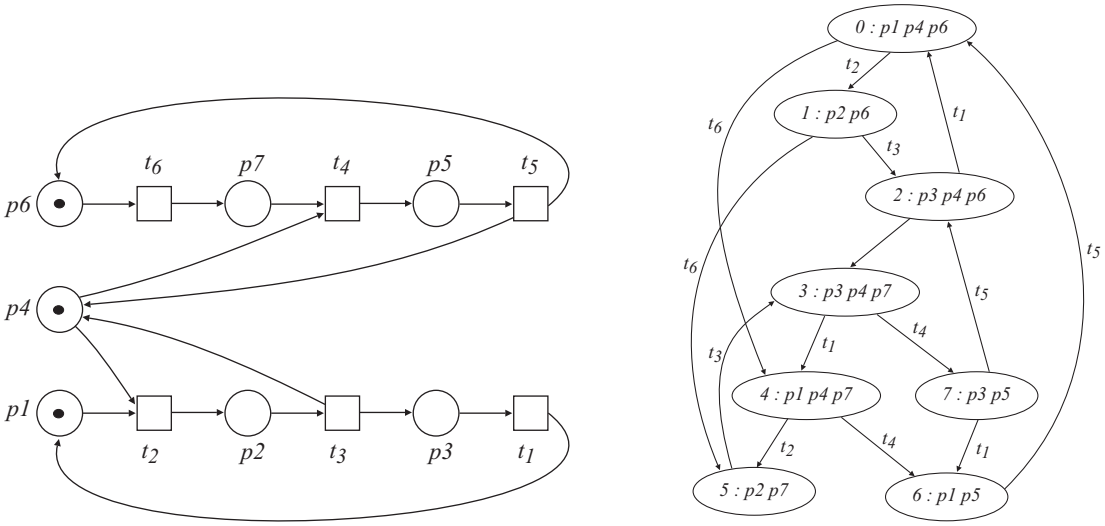
A técnica de desdobramento [33, 11] tem sido aplicada com sucesso para resolver problemas de alcançabilidade, reduzindo significativamente o tamanho do espaço de estados, quando comparada com o método do grafo de alcançabilidade.

3.5 Desdobramento

O conceito de desdobramento (do termo inglês *unfolding*) foi introduzido originalmente por McMillan [33] com o objetivo de evitar o problema da explosão do espaço de estados na análise dos sistemas modelados em redes de Petri.

A técnica é um método para resolver o problema de alcançabilidade que explora e preserva as informações de concorrência da rede. O desdobramento de uma rede é uma outra rede, acíclica, finita e que possui todas as marcações alcançáveis da rede original.

Para o entendimento da técnica de desdobramento, é necessário definir os seguintes termos relativos à rede de Petri, que serão exemplificados a partir da rede da figura 3.5.



(a) Rede (b) Grafo de alcançabilidade

Figura 3.5: Rede de Petri e grafo de alcançabilidade

Definição 8 (Relação de Ordem) *Seja $N = (P, T, F, W, M)$ uma rede acíclica e seja $x_1, x_2 \in P \cup T$.*

- *$x_1 \leqslant x_2$ (x_1 precede x_2): x_1 precede x_2 se (x_1, x_2) pertence ao fechamento reflexivo transitivo de $F(F^*)$, ou seja, se existe um caminho entre x_1 e x_2.*
- *$x_1 \# x_2$ (x_1 e x_2 estão em conflito): x_1 e x_2 estão em conflito se existir transições distintas $t_1, t_2 \in T$ tal que ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$, $t_1 \leqslant x_1$ e $t_2 \leqslant x_2$*
- *$x_1 \shortparallel x_2$ (x_1 e x_2 são concorrentes): x_1 e x_2 são concorrentes se não são precedentes $\neg\,(x_1 \leqslant x_2)$ e $\neg\,(x_2 \leqslant x_1)$ e não estiverem em conflito $\neg\,(x_1 \# x_2)$.*

Na rede da figura 3.5, podem ser vistos alguns exemplos de relações de ordem:

$$t_2 \shuffle t_6 \ , \ t_2 \# t_4 \ , \ p_1 \shuffle p_6 \ , \ p_2 \leqslant p_3 \ , \ t_2 \leqslant t_3 \leqslant t_1 \ , \ p_3 \shuffle p_5$$

Definição 9 (Rede de Ocorrência) *Uma rede Condição/Evento $N = (B, E, F)$ é uma rede de ocorrência se e somente se:*

(i) $\forall b \in B \, , |{}^\bullet b| \leq 1$ *(as condições têm no máximo uma transição de entrada)*

(ii) F *é acíclico (o fechamento transitivo é irreflexivo)*

(iii) Nenhum evento $e \in E$ *está em auto-conflito (um nó* $x \in B \cup E$ *está em auto-conflito se* $x \# x$*)*

Definição 10 (Processos de Ramificação) *São rotulamentos de uma rede que representam todos os caminhos possíveis mantendo as informações sobre concorrência e conflitos. O processo de ramificação de uma rede N é o par $\beta = (ON, \varphi)$ sendo que $ON = (B, E, F)$ é uma rede de ocorrência e φ é uma função de rotulamento que satisfaz as seguintes propriedades:*

(i) $\varphi(B) \subseteq S$ *e* $\varphi(E) \subseteq T$ *(preserva a natureza dos nós)*

(ii) $\forall e \in E$*, a restrição de* φ *para* ${}^\bullet e$ *é uma função bijetora entre* ${}^\bullet e$ *(em* N*) e* ${}^\bullet \varphi(e)$ *(em* β*), e simetricamente para* $e^\bullet$ *e* $\varphi(e)^\bullet$ *(preserva o ambiente das transições)*

(iii) Sendo $Min(O)$ *o conjunto dos elementos mínimos de* $B \cup E$ *que possuem seu pré-conjunto vazio, a restrição de* φ *para* $Min(O)$ *é uma função bijetora entre* $Min(O)$ *e* M_0 *(*β *começa pela marcação inicial). Considerando redes em que não existem eventos com pré-conjunto vazio,* $Min(O)$ *só podem ser condições (*B*), portanto* $Min(O) = M_0$*.*

(iv) $\forall e_1, e_2 \in E$*, se* ${}^\bullet e_1 = {}^\bullet e_2$ *e* $\varphi(e_1) = \varphi(e_2)$ *então* $e_1 = e_2$ *(*β *não duplica as transições de* N*)*

Embora o processo de ramificação possa ser infinito, é possível truncá-lo em uma sub-rede, denominado prefixo finito, que possui todas as marcações alcançáveis [11]. Esse prefixo é o desdobramento, mas, para formalizar sua definição, ainda são necessários os conceitos de configuração, configuração local, prefixo finito completo e evento de corte. A figura 3.6 é o desdobramento da rede exemplo.

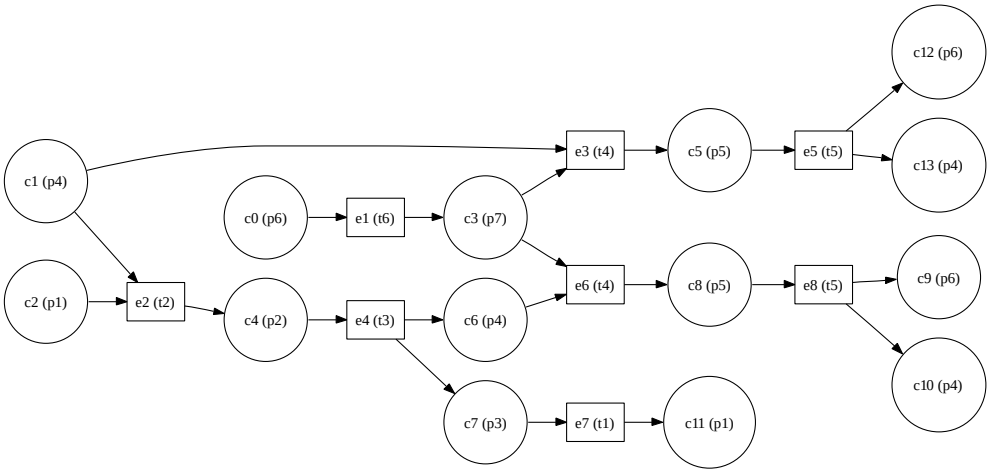


Figura 3.6: Desdobramento da rede da figura 3.5

Definição 11 (Configuração) *Uma configuração C é um conjunto de transições que satisfazem as seguintes condições:*

(i) $\forall e' \leqslant e,\ e \in C \Rightarrow e' \in C$

(ii) $\forall e_1, e_2 \in C,\ e_1 \neq e_2 \Rightarrow^{\bullet} e_1 \cap^{\bullet} e_2 = \emptyset$

Uma configuração pode ser associada com uma marcação $\texttt{Mark}(C)$ que corresponde a uma marcação alcançável a partir de M_0 após todas as transições de C terem sido disparadas.

$$\texttt{Mark}(C) = p((Min(N) \cup C^{\bullet}) \backslash {}^{\bullet}C)$$

Na figura 3.6, a seqüência de transições/eventos $\{e_2, e_4, e_1, e_6\}$ é uma configuração e a marcação alcançável a partir dessa configuração é $\varphi(\{c_7, c_8\}) = p_3\, p_5$.

Definição 12 (Configuração Local) *A configuração local de um evento e, denotada por* $[e]$*, é a configuração mínima que contém* e *e todos os seus precedentes*

Por exemplo, a configuração mínima para o evento e_8 no desdobramento da figura 3.6 é $[e_8] = \{e_2, e_4, e_1, e_6, e_8\}$

Definição 13 (Prefixo Finito Completo) *Na maioria dos casos, o desdobramento* β *da rede é infinito, mas é possível truncá-lo em uma subrede denominada prefixo finito completo* β'*, que possui todas as informações de* β*. O prefixo* β' *de* β *é completo se, para cada marcação alcançável* M*, existe uma configuração* $C \in \beta'$ *tal que:*

1. $\mathbf{Mark}(C) = M$
2. *para cada transição* t *habilitada por* M*, existe uma configuração* $C \cup \{e\}$ *tal que* $e \notin C$ *e* $\varphi(e) = t$

A rede da figura 3.6 é um prefixo finito completo que foi cortado do desdobramento sem perder informações. Esse corte é feito identificando eventos que levam a uma marcação já representada, esses eventos são chamados eventos de corte.

Definição 14 (Evento de corte) *Seja* $\prec$ *uma ordem adequada (adequate order) nas configurações do desdobramento e seja* β *o seu prefixo contendo um evento* e*. O evento* e *é um evento de corte (cut-off event) de* β *se* β *contém a configuração local* $[e']$ *tal que:*

(i) $\mathbf{Mark}([e]) = \mathbf{Mark}([e'])$

(ii) $[e'] \prec [e]$

Quando $[e'] < [e]$, pode-se dizer que $[e']$ é menor que $[e]$ se ele tiver menos eventos, ou seja, $|[e']| < |[e]|$. Com isso é fácil verificar que essa ordem parcial é uma ordem adequada, portanto $[e'] \prec [e]$.

Os eventos e_5 e e_7, representados no desdobramento da figura 3.6, são eventos de corte porque levam à marcação inicial $M_0 = p_1\, p_4\, p_6$. Já o evento e_8 é um evento de corte, pois a marcação alcançada $p_3\, p_4\, p_6$ é mesma alcançada pelo evento e_4, $\mathtt{Mark}([e_8]) = \mathtt{Mark}([e_4])$, porém com $[e_4] \prec [e_8]$.

A construção do prefixo finito completo é feita seguindo o algoritmo melhorado de Esparza, Römer e Vogler [12], também chamado de algoritmo *ERV Unfolding*. O algoritmo é apresentado a seguir.

Algoritmo 1 Construção do Prefixo Finito Completo

entrada: Uma rede N, sendo que $M_0 = \{p_1, \ldots, p_k\}$
saída: Um prefixo finito completo do unfolding `Unf`
`Unf` $\leftarrow$ lugares de M_0
`pe` $\leftarrow$ transições habilitadas por M_0
`cut-off` $\leftarrow \emptyset$
while `pe` $\neq \emptyset$ **do**
  escolha um evento $e = (t, X)$ de `pe` tal que $[e]$ seja mínima respeitando a ordem $\prec$
  if $[e] \cap$ `cut-off` $= \emptyset$ **then**
    adicione e e seu pós-conjunto em Unf
    `pe` $\leftarrow$ `PE(Unf)` {Atualiza as transições habilitadas}
    if (e é um evento de corte) **then**
      cut-off $\leftarrow$ cut-off $\cup\ e$
    end if
  else
    `pe` $\leftarrow$ `pe` $\setminus \{e\}$
  end if
end while

O algoritmo *ERV Unfolding* foi utilizado para implementar diversas ferramentas que geram o desdobramento de uma rede de Petri, entre elas podemos destacar o *PUNF* [27], o *ERVunfold* [28], e o *MOLE*[1]. A ferramenta *MOLE* foi utilizada por ser considerada a versão mais simples e eficiente para as redes de Petri descritas neste trabalho.

3.6 Ferramenta *MOLE*

O *MOLE* é uma ferramenta disponibilizada sob licença pública geral (*GNU GPL*) para gerar desdobramentos de redes de Petri seguras.

Desenvolvida por Stefan Römer e Stefan Schwoon para ser compatível com o ambiente do projeto *PEP (Programming Environment based on Petri Nets)*[2] [47], mantido pelo grupo de Sistemas Paralelos do Departamento de Ciência da Computação da Universidade de Oldenburg na Alemanha.

[1]Disponível em: <http://www.fmi.uni-stuttgart.de/szs/tools/mole/> Acesso em: 20 set. 2008
[2]Disponível em: <http://theoretica.informatik.uni-oldenburg.de/~pep/> Acesso em: 20 set. 2008

A entrada do programa é um arquivo com a descrição da rede no formato próprio do ambiente, e a saída é o desdobramento em um arquivo no formato também utilizado pelas ferramentas do ambiente *PEP*. Anexo à ferramenta há um aplicativo para conversão dos arquivos de saída para o formato gráfico.

A estrutura mínima do arquivo de entrada que descreve a rede deve conter em ordem as seguintes partes [17]:

1. Cabeçalho;
2. Lista dos lugares;
3. Lista das transições;
4. Lista dos arcos de transição para lugar;
5. Lista dos arcos de lugar para transição.

O cabeçalho deve iniciar com a palavra-chave `PEP`. A segunda linha deve especificar o tipo de rede, sendo que `PTNet` é a abreviação para uma rede de Petri Lugar/Transição. A terceira e última linha do cabeçalho irá conter a palavra-chave que define o formato da rede utilizada, neste caso `FORMAT_N`.

```
PEP
PTNet
FORMAT_N
%Comentarios podem ser feitos somente depois do cabeçalho
PL
%lista do lugares
TR
%lista das transições
TP
%lista dos arcos de transição para lugar
PT
%lista dos arcos de lugar para transição
```

As listas de lugares, transições, arcos de transição para lugar e arcos de lugar para transição devem vir logo abaixo do cabeçalho, indicados pelas respectivas palavras-chave `PL`, `TR`, `TP` e `PT`. Todas devem estar presentes no arquivo mesmo no caso de a lista a ser descrita estar vazia.

Por ser um formato de arquivo desenvolvido para um ambiente gráfico, a posição dos lugares e das transições precisa ser especificada por meio de coordenadas `num@num`, sendo que `num` é um número inteiro positivo ou negativo. É preciso especificar também um nome e um identificador único para os lugares e transições. Para os lugares é permitido definir a marcação `M num m num`, em que `M` é relativo à marcação inicial e `m` à marcação corrente.

Para especificar os arcos, deve-se seguir a sequência `identificador_da_transição < identificador_do_lugar` para os arcos de transição para lugar (`TP`) e a sequência `identificador_do_lugar > identificador_da_transição` para os arcos de lugar para transição (`PT`).

A figura 3.7 descreve o arquivo de entrada referente à rede da figura 3.5 que foi utilizado como entrada no *MOLE* para gerar o desdobramento apresentado na figura 3.6. O arquivo foi criado como um arquivo de texto e salvo com a extensão `ll_net`. Por não se utilizar a parte gráfica do ambiente *PEP*, as coordenadas estão todas definidas como `0@0`.

```
PEP
PTNet
FORMAT\_N
PL
1"p1"0@0M1m1
2"p2"0@0M0m0
3"p3"0@0M0m0
4"p4"0@0M1m1
5"p5"0@0M0m0
6"p6"0@0M1m1
7"p7"0@0M0m0
TR
1"t1"0@0
2"t2"0@0
3"t3"0@0
4"t4"0@0
5"t5"0@0
6"t6"0@0
TP
1<1
2<2
3<3
3<4
4<5
5<4
5<6
6<7
PT
1>2
2>3
3>1
4>2
4>4
5>5
6>6
7>4
```

Figura 3.7: Arquivo de entrada para o *MOLE* referente à rede da figura 3.5

3.7 Considerações

As redes de Petri podem ser utilizadas como estrutura de representação para resolver problemas de planejamento. Em 2000, Silva e Colegas [46] fizeram uma proposta de transformação do grafo de planos em uma rede de Petri acíclica e trataram o problema de planejamento como um problema de alcançabilidade. O planejador resultante chamou-se *Petriplan* e será abordado neste trabalho no capítulo 4.

A partir desse trabalho, foram derivados vários outros dentro do grupo de pesquisa em planejamento em inteligência artificial do LIAMF. O próprio algoritmo do *Petriplan* teve cinco versões, uma delas resultando em um novo trabalho de Silva, a Rede de Planos [45], que abandona o grafo de planos e traduz os problemas de planejamento em *PDDL* diretamente em uma rede de Petri acíclica. A Rede de Plano também será descrita no capítulo 4.

Entre outros trabalhos derivados, podemos destacar Montaño [34], que apresenta uma técnica na qual o problema de alcançabilidade é tratado por um resolvedor *SAT* baseado em fórmulas não-clausais, a partir do seu mapeamento para fórmulas em *NNF* (*Negation Normal Form*). Carvalho [8] desenvolveu o planejador *GAPNet* (*Genetic Algorithm for Reachability on Petri Nets*), que propõe uma solução para o problema de alcançabilidade utilizando o paradigma dos algoritmos genéticos.

CAPÍTULO 4

TRABALHOS RELACIONADOS

Neste capítulo são descritos trabalhos relacionados com o problema de planejamento em inteligência artificial utilizando redes de Petri. O planejador *Petriplan*, desenvolvido por Silva e colegas [46], é apresentado na seção 4.1. A seção 4.2 contém a descrição do modelo chamado Rede de Planos, também desenvolvido por Silva [45]. Na seção 4.3, é feita uma breve descrição da abordagem que utiliza desdobramento de redes de Petri, apresentada por Hickmott e colegas [22].

4.1 *Petriplan*

Esta seção apresenta o planejador *Petriplan* [46], uma abordagem para o problema de planejamento em que o grafo de planos é transformado em uma rede de Petri, e o problema de alcançabilidade de sub-marcação é resolvido por meio de técnicas de programação inteira.

Por compactar o espaço de busca, o grafo de planos foi utilizado por muitos planejadores como uma fase de pré-processamento, conforme apresentado na seção 2.5. O *Petriplan* também utiliza essa representação na fase inicial e em seguida transforma o problema de planejamento em um problema de alcançabilidade em redes de Petri. O algoritmo é composto por três fases, sendo elas [44]:

1. Construção do grafo de planos;
2. O grafo de planos é transformado em uma rede de Petri equivalente (o problema de planejamento original é mapeado como um problema de alcançabilidade de sub-marcação);
3. O problema de alcançabilidade em redes de Petri é convertido em um problema de Programação Inteira.

Na terceira fase, o problema de Programação Inteira (PI), baseado na equação fundamental das redes de Petri, é resolvido por sistemas específicos de otimização. Se nenhuma solução for encontrada, o algoritmo volta para a primeira fase e realiza uma nova expansão do grafo. Esse processo continua até que uma solução seja encontrada ou que a condição de falha aconteça durante a expansão do grafo.

As estruturas do grafo de planos, obtido na primeira fase, são traduzidas diretamente em estruturas de redes de Petri equivalentes, demonstradas na figura 4.1 e descritas a seguir [46]:

- **nós-ação:** um nó ação do grafo é traduzido em uma única transição na rede de Petri (figura 4.1(a));
- **nós-proposição:** um nó proposição é traduzido em um lugar e uma transição, com um arco do lugar para a transição (figura 4.1(b));
- **arestas-efeito:** são arestas de nós-ação para nós-proposição, e são traduzidas em um arco que vai da transição que representa o nó ação para o lugar representando o nó proposição (figura 4.1(c));
- **arestas-pré-condição:** são arestas de nós-proposição para nós ação. 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 (figura 4.1(d));
- **relações de 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 (figura 4.1(e)).

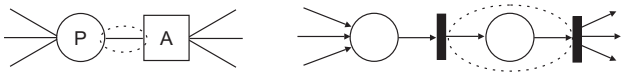
As relações de exclusão mútua entre as proposições do grafo de planos não são representadas na rede, uma vez que elas são utilizadas apenas durante a construção do grafo para determinar as exclusões mútuas entre as ações.

O estado inicial do problema de planejamento é representado por uma marca em cada lugar que representa a camada zero do grafo de planos. A marcação inicial da rede é

(a) tradução dos nós-ação

(b) tradução dos nós-proposição

(c) tradução das arestas-efeito



(d) tradução das arestas-pré-condição

(e) tradução das relações de exclusão mútua

Figura 4.1: Regras de tradução utilizadas no *Petriplan* [46]

definida pelas marcas nos lugares do estado final e pelas marcas que controlam as relações de exclusão mútua. Finalmente o objetivo do problema é a sub-marcação da rede que contém os lugares marcados referentes aos nós-proposições do estado objetivo representado no grafo de planos.

Com a rede de Petri traduzida, o problema de planejamento precisa ser mapeado em um problema de alcançabilidade de sub-marcação. Seja N a rede obtida pela tradução do grafo de planos, M_0 a marcação inicial, M uma marcação alcançável da rede e M_s a sub-marcação alcançável que contém os lugares que representam as proposições do estado objetivo, sendo que $M_s \subseteq M$. O problema de alcançabilidade de sub-marcação pode ser definido como sendo encontrar uma seqüência de transições s que a partir da marcação inicial M_0 alcance a sub-marcação M_s.

A rede resultante do processo de tradução é uma rede acíclica, portanto a equação fundamental da redes de Petri (3.1) pode ser usada para definir o problema de programação inteira associado ao problema de alcançabilidade [45].

O problema de PI correspondente é definido pela restrição vetorial $\mathtt{C}.\bar{s} \geq M_s - M_0$, sendo que $\mathtt{C}$ é a matriz de incidência descrita na seção 3.2. Qualquer vetor $\bar{s}$ que seja solução para a equação representa quantas vezes cada transição da rede é disparada para obter M a partir de M_0, sendo que $M(p) \geq M_s(p)$, para todo lugar p na rede N. Ou seja, o vetor característico $\bar{s}$ resolve o problema de alcançabilidade de sub-marcação que alcança a sub-marcação M_s a partir da marcação inicial M_0.

Depois de encontrada a solução para a problema de PI da terceira fase, o plano para o problema de planejamento original deve ser estabelecido. A partir do vetor $\bar{s}$, são selecionadas apenas as transições que foram disparadas, ou seja, que $\bar{s}(t) > 0$, formando o conjunto T. Mas essas transições devem ser somente as transições que representam nós-ações no grafo de planos, portanto considera-se o subconjunto $A \subset T$ de transições ações.

Finalmente é preciso encontrar quais ações de A pertencem realmente ao plano. Para isso, removem-se as ações de A que não estão no caminho que ligam os lugares que representam o estado inicial aos lugares que representam o estado objetivo, percorrendo a rede no sentido contrário ao dos arcos. As ações resultantes do conjunto A são as ações que definem o plano.

4.2 Rede de Planos

A Rede de Planos [9] é uma abordagem para representar o espaço de estados do problema de forma mais eficiente que o grafo de planos. Em seus recentes trabalhos [9] [45], Silva e colegas abandonaram o grafo de planos e traduziram o problema de planejamento direto da linguagem PDDL para uma rede de Petri.

Nessa nova estrutura é possível aproveitar melhor o poder de representação das redes de Petri. Uma diferença importante está na representação das inconsistências entre as ações. A rede de planos permite representar essas inconsistências por estruturas no próprio

modelo, enquanto que no grafo de planos elas são tratadas por relações de exclusão mútua, usadas como meta-informação no procedimento de busca [45].

A partir da descrição do problema de planejamento em PDDL, são definidos dois conjuntos: $\mathcal{P}$ que contém todas as proposições instanciadas, e $\mathcal{A}$ que contém todas as ações instanciadas.

Para representar as ações e as proposições em uma rede de Petri, é utilizada a convenção de que as transições da rede representam as ações de $\mathcal{A}$, e os lugares representam as pré-condições e os efeitos das ações, que são as proposições presentes em $\mathcal{P}$.

A ação instanciada move(r_1,r_2) e sua representação em rede de Petri pode ser visualizada na figura 4.2. Os efeitos negativos não são representados no modelo, prática comum nos planejadores baseados na linguagem *STRIPS*, por isso a notação tracejada. São úteis apenas para o cálculo das relações de conflito entre as ações.

```
(:action move
     :parameters (?from ?to)
     :precondition (at ?from)
     :effect (and (at ?to)
             (not (at ?from)))
)
```

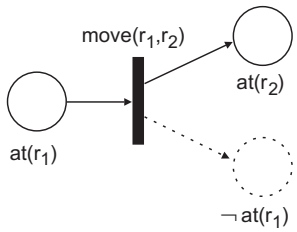


Figura 4.2: Estrutura de representação da ação move(r_1,r_2) [45]

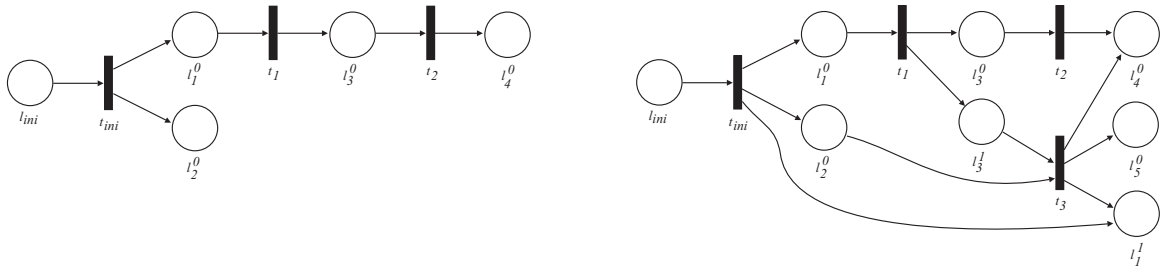
A construção da rede de planos inicia com a inclusão de um lugar para cada proposição verdadeira do estado inicial do problema. Assim como no grafo de planos, o próximo passo na construção do modelo consiste na inclusão das ações que têm suas pré-condições presentes no modelo.

A inclusão de uma transição t é permitida se todos os lugares que representam suas pré-condições estiverem presentes na rede. Se um desses lugares já for uma pré-condição para uma outra transição da rede, é necessário fazer uma cópia do lugar para que não exista conflito entre as duas transições.

A cópia do pós-conjunto da transição também deve ser feita para a nova transição t. Os efeitos de t que não estiverem presentes na rede são incluídos, e os que já existem e não são usados como pré-condição, o pós-conjunto da transição é atualizado, ligando a

transição t ao lugar existente. Para os efeitos que já são usados como pré-condição, é necessária a cópia do lugar juntamente com o seu pré-conjunto.

Esse processo é exemplificado na figura 4.3, na qual a inclusão de uma transição t_3, que tem como pré-condições os lugares l_2 e l_3 e como efeitos l_1, l_4 e l_5, na rede da figura 4.3(a) resulta na rede da figura 4.3(b). Os lugares são representados como l_i^k, sendo que o índice i representa a proposição associada ao lugar, e o índice k identifica cada cópia do lugar, sendo l_1^0 a primeira ocorrência do lugar l_1.



(a) (b)

Figura 4.3: Regra para inclusão de uma transição [45]

A transição t_3 pode ser incluída pois suas duas pré-condições estão presentes na rede, l_2^0 e l_3^0. O lugar l_2^0 não é pré-condição para nenhuma outra transição, enquanto que l_3^0 faz parte do pré-conjunto de t_2. Por isso a cópia do lugar é criada e o lugar l_3^1 é adicionado na rede figura 4.3(b), juntamente com o arco que liga como efeito de t_3 o lugar l_4^0, também efeito de t_2.

Entre os efeitos de t_3, l_5 ainda não está na rede, portanto é adicionada sua primeira ocorrência l_5^0. O lugar l_4 está presente e não é pré-condição de nenhum lugar, por isso é feita apenas a ligação de t_3 ao lugar l_4^0. Já o lugar l_1, que está na rede e é pré-condição da transição t_1, tem sua cópia criada l_1^1, e seu pré-conjunto atualizado.

As ações incluídas na rede, as cópias dos lugares e os novos lugares incluídos como efeito das ações constituem uma camada da rede de planos. Assim como no grafo de planos, todas as transições de uma mesma camada da rede de planos são concorrentes. Entretanto, as ações podem apresentar inconsistências entre si e, para evitar que planos com ações conflitantes sejam gerados, é preciso incluir estruturas capazes de controlar esses disparos.

O disparo de uma transição t numa camada da rede de planos representa a execução da ação correspondente em um determinado ponto do plano. Mas a existência da transição naquela camada não implica que ela deve ser disparada naquele momento, portanto é preciso disponibilizar uma estrutura capaz de oferecer esse controle.

A figura 4.4 mostra a estrutura de controle de disparo para uma transição na rede de planos. Os lugares e transições que fazem parte da estrutura de controle são descritos como: O lugar I indica o início da execução das ações nesta camada; A transição Θ_t indica o início do controle do disparo de t; O lugar l_1 é um conflito na rede que permite apenas o disparo de uma das transições: t ou λ_t, sendo que λ_t representa o não-disparo de t; A transição ϕ_t representa o fim do controle do disparo da transição t; O lugar F indica o fim da execução das ações da camada.

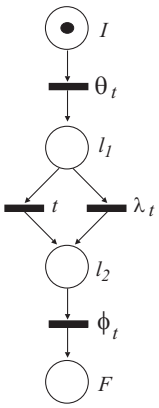


Figura 4.4: Estrutura de controle de disparo para uma transição [45]

Duas transições a e b em uma camada da rede de planos podem ocorrer de forma concorrente. A estrutura de controle da figura 4.4 pode ser estendida para controlar o disparo de transições concorrentes e pode ser vista na figura 4.5.

Essa estrutura permite a execução dos seguintes sub-planos no problema de planejamento: λ, (a), (b), (a;b), (b;a) e (a|b). Sendo que λ indica um plano vazio, (x) indica que o plano é composto pela ação x; (x;y) indica que o plano é dado pela ação x seguida da ação y e (x|y) indica que as ações x e y são concorrentes no plano.

Para resolver os conflitos entre as ações e substituir os *mutex* dos modelos propostos sobre o grafo de planos, são estabelecidas relações de ordenação entre as ações. Existem diferentes tipos de inconsistências e conflitos entre ações e proposições, mas no caso das

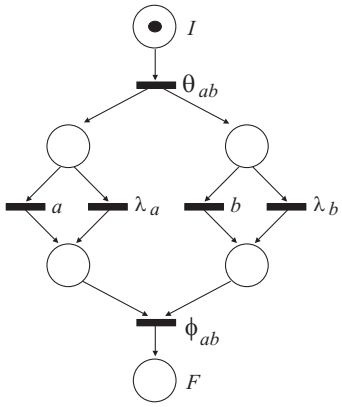


Figura 4.5: Estrutura de controle de disparo para transições concorrentes [45]

transições da rede de planos, consideram-se apenas as inconsistências entre as ações. A partir destas inconsistências, definem-se suas respectivas relações de ordenação:

1. t_a **e** t_b **não-concorrentes:** para efeitos inconsistentes em que t_a tem como efeito a negação de algum efeito de t_b. Neste caso apenas o disparo concorrente das transições gera a inconsistência, portanto $t_a \nparallel t_b$ se ocorrer:

- apenas o disparo de t_a (a);
- apenas o disparo de t_b (b);
- o disparo de t_a seguido do disparo de t_b (a;b);
- o disparo de t_b seguido do disparo de t_a (b;a);
- nenhum disparo (λ).

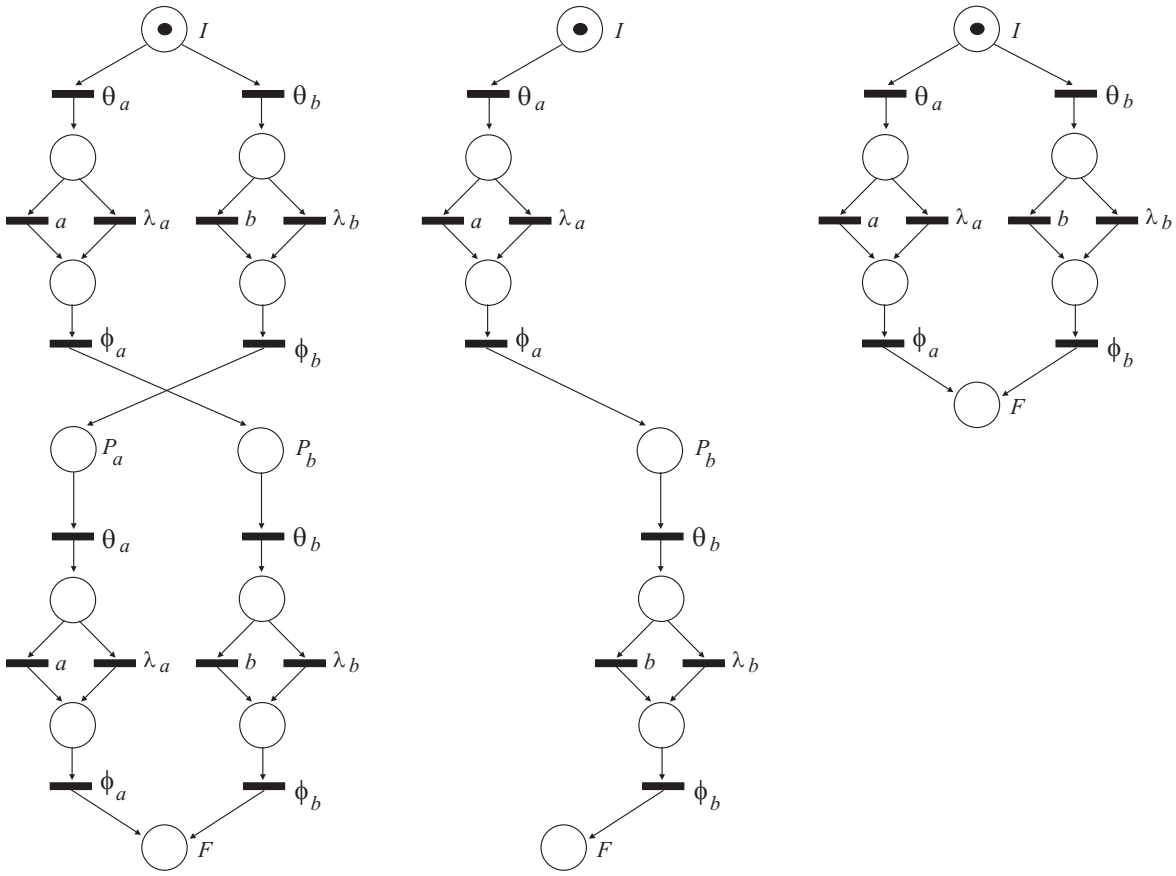
2. t_a **precede** t_b**:** para casos de interferência, em que t_b tem como efeito a negação de alguma pré-condição de t_a. Somente a antecipação do disparo da transição t_a pode resolver a inconsistência, portanto $t_a \lhd t_b$ se ocorrer:

- apenas o disparo de t_a (a);
- apenas o disparo de t_b (b);
- o disparo de t_a seguido do disparo de t_b (a;b);
- nenhum disparo (λ).

3. t_a **e** t_b **são mutuamente exclusivos:** quando t_b tem como efeito a negação de alguma pré-condição de t_a, que por sua vez tem como efeito a negação de alguma pré-condição de t_b, os disparos devem ser mutuamente exclusivos, ou seja, quando ocorrer:

- apenas o disparo de t_a (a);
- apenas o disparo de t_b (b);
- nenhum disparo (λ).

As estruturas de controle para as três relações de ordenação entre duas transições a e b são apresentadas na figura 4.6.



(a) disparo $t_a \rightarrowtail t_b$ (b) disparo $t_a \triangleleft t_b$ (c) disparo $t_a \diamond t_b$

Figura 4.6: Estruturas de controle de disparo para as relações de ordenação [45]

As estruturas básicas de controle apresentadas nas figuras 4.5 e 4.6 podem ser combinadas para controlar mais de uma relação de ordenação sobre um conjunto de ações. Para

evitar que mais de um caminho seja disparável na rede, é incluído na estrutura um lugar de sincronização I'. E quando necessário, um lugar Q deve ser incluído para impedir que o controle de uma relação termine na estrutura de controle de outra.

As quatro relações apresentadas ($\shortparallel, \nVdash, \lhd, \diamond$) definem as ordenações possíveis entre duas transições quaisquer de uma camada. Ao contrário do grafo de planos, o modelo proposto por Silva utiliza as relações $\nVdash$ e $\lhd$ no processo de construção, gerando um modelo que representa de forma mais completa o problema de planejamento.

O processo de construção obtém uma rede acíclica, portanto as mesmas considerações quanto à solução dos problemas de alcançabilidade utilizadas em [44] são aplicadas às redes obtidas pela construção da rede de planos.

4.3 *PUP - Planning via Unfolding of Petri nets*

A proposta para traduzir um problema de planejamento em uma rede de Petri também foi desenvolvida por Hickmott e colegas em seus trabalhos [21] e [22]. O foco principal foi determinar quais técnicas utilizadas na análise de redes de Petri poderiam ser aplicadas com sucesso em problemas de planejamento.

A técnica de desdobramento de redes de Petri, eficiente para solucionar problemas de alcançabilidade, foi investigada para ser utilizada como um novo método para a busca ótima de planos parcialmente ordenados, uma vez que o tamanho da rede gerada pelo desdobramento pode ser exponencialmente menor que o tamanho representado pelo espaço de estados [22].

O planejador *PUP* é formado por duas fases:

1. O problema de planejamento é traduzido para uma rede de Petri segura;

2. O problema de alcançabilidade na rede é resolvido por desdobramento.

O primeiro passo da tradução é mapear o problema de planejamento em um problema equivalente no qual todas as ações são seguras. Para que uma ação seja segura, ela não pode afetar pós-condições que não sejam suas pré-condições, ou seja, todas as variáveis de estado modificadas por ela devem ser pré-condições.

Assim, cada ação $o = \langle p, e\rangle$ é substituída por $2^{|e'|}$ ações, sendo que e' é o conjunto de literais representados em e mas não em p, de forma que todas as combinações estejam expressas. Por exemplo, a ação $o = \langle p, e\rangle$ em que $p = \{a, \neg b, c\}$ e $e = \{\neg a, b, d, \neg e\}$, representada na figura 4.7(a), é substituída por quatro ações seguras $o_i = \langle p_i, e_i\rangle$, descritas como:

$$
\begin{array}{lll}
o_1 = \{a, \neg b, c, d, \neg e\} & e_1 = \{\neg a, b\} & e'_1 = \{\} \\
o_2 = \{a, \neg b, c, \neg d, \neg e\} & e_2 = \{\neg a, b, d\} & e'_2 = \{d\} \\
o_3 = \{a, \neg b, c, d, e\} & e_3 = \{\neg a, b, \neg e\} & e'_3 = \{\neg e\} \\
o_4 = \{a, \neg b, c, \neg d, e\} & e_4 = \{\neg a, b, d, \neg e\} & e'_4 = \{d, \neg e\}
\end{array}
$$

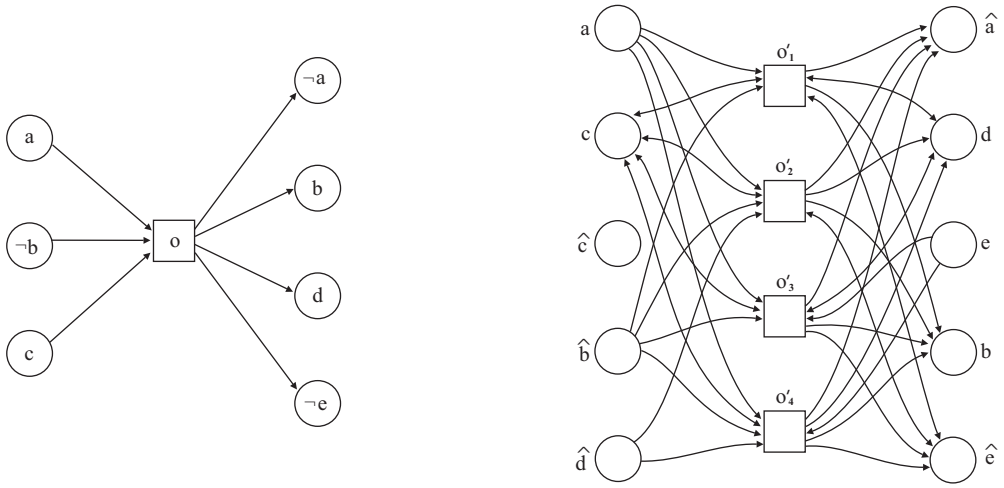
O segundo passo da tradução é deixar as ações apenas com pré-condições positivas. As pré-condições negativas são eliminadas pela substituição de cada $\neg a$ por sua respectiva pré-condição positiva $\hat{a}$. O conjunto $\hat{\mathcal{A}}$ é dado por $\{\hat{a}|a \in \mathcal{A}\}$, sendo que $\hat{a}$ é verdadeiro quando a é falso [22]. As quatro ações seguras o_i descritas anteriormente são substituídas por quatro novas ações nas quais os literais negativos são substituídos por seus respectivos literais positivos:

$$
\begin{array}{ll}
o'_1 = \langle\{a, \hat{b}, c, d, \hat{e}\}, & \{\neg a, \hat{a}, \neg\hat{b}, b\}\rangle \\
o'_2 = \langle\{a, \hat{b}, c, \hat{d}, \hat{e}\}, & \{\neg a, \hat{a}, \neg\hat{b}, b, \neg\hat{d}, d\}\rangle \\
o'_3 = \langle\{a, \hat{b}, c, d, e\}, & \{\neg a, \hat{a}, \neg\hat{b}, b, \neg e, \hat{e}\}\rangle \\
o'_4 = \langle\{a, \hat{b}, c, \hat{d}, e\}, & \{\neg a, \hat{a}, \neg\hat{b}, b, \neg\hat{d}, d, \neg e, \hat{e}\}\rangle
\end{array}
$$

Finalmente, o último passo na tradução é mapear o problema de planejamento equivalente em uma rede de Petri, tal que os lugares são $P = \mathcal{A} \cup \hat{\mathcal{A}}$, as transições são o conjunto de ações seguras obtidas sobre uma ação $o \in \mathcal{O}$ e a relação de fluxo F é obtida por meio de $t = \langle p, e\rangle \in T$ sendo que $\{(a, t)|a \in p\} \cup \{(t, a)|a \in e \text{ ou } a \in p \text{ e } \neg a \notin e\}$.

A figura 4.7(b) ilustra esse mapeamento para uma única ação, resultando na rede de Petri gerada para a ação o transformada nas quatro ações seguras o'_i descritas acima.

A tradução do problema de planejamento para uma rede de Petri segura foi necessária



(a) (b)

Figura 4.7: Tradução de uma ação de um problema de planejamento em uma RdP

devido uma limitação da ferramenta utilizada para gerar o desdobramento. Conforme apresentado na seção 3.6, a ferramenta *MOLE* aceita apenas redes seguras como entrada.

Para resolver o problema de alcançabilidade na rede por meio do desdobramento é preciso criar uma condição de parada quando o desdobramento encontrar a marcação ou sub-marcação desejada. Através da opção disponibilizada pelo *MOLE* para terminar o desdobramento quando uma determinada transição é alcançada, basta criar uma transição final em que suas pré-condições são os lugares que representam os literais do estado objetivo do problema de planejamento.

Durante o desdobramento, o algoritmo ordena os items a serem expandidos de tal maneira que cada camada é explorada inteiramente antes da próxima, fazendo o equivalente a uma busca em largura. Uma vez que não é necessário o desdobramento completo, mas apenas um caminho para o objetivo, o algoritmo pode ser otimizado para uma busca orientada ao objetivo ao invés de uma busca exaustiva.

A proposta foi mudar as relações de ordem seguidas pelo *MOLE* por outras que guiam melhor a ferramenta na busca pela solução. Nos experimentos realizados por Hickmott e colegas [22], o algoritmo foi modificado para implementar a busca A^* e a heurística h^1 descrita na seção 2.6.

4.4 Considerações

O número de transições de uma rede de Petri está diretamente ligado à dificuldade em se encontrar uma seqüência de disparos de transições que resolva o problema de alcançabilidade. No modelo proposto por Silva em [44], o número de transições na rede obtida pela tradução é proporcional ao número de vértices do grafo de planos. Na verdade, para cada proposição ou ação do grafo existe uma transição na rede resultante.

Sabe-se também que ao mapear o problema de alcançabilidade como um problema de programação inteira acaba existindo uma variável associada a cada transição da rede. E quanto menor for o número de variáveis de um problema de programação inteira menor será o espaço de busca a ser explorado durante a solução do problema.

Por isso, em [45] Silva propôs simplificações e traduções alternativas no modelo, removendo estruturas para reduzir o número de transições da rede, com o objetivo de obter uma rede menor e conseqüentemente um problema de programação inteira menor.

As simplificações removem estruturas redundantes que estão presentes no grafo de planos. Assim, a representação do problema de planejamento por redes de Petri, em termos de espaço, é mais eficiente que o grafo de planos [45].

Os experimentos realizados por Silva em [45] mostram uma redução significativa no tamanho da rede quando a tradução alternativa proposta é combinada com a remoção de relações de exclusão mútua redundantes, embora ainda sem resultados significativos.

O modelo inicial desenvolvido por Silva evoluiu para outros, nos quais cada vez mais se explorou o poder representacional das redes de Petri. O modelo resultante, a rede de planos, é mais preciso que o grafo de planos, permitindo um processo mais eficiente de busca pela solução.

Entretanto, a estratégia de reordenação das ações ainda carrega muitas redundâncias e faz com que grande parte das reordenações não levem a uma solução do problema, tornando o processo de busca caro. O tamanho das redes geradas ainda é um ponto fraco, uma vez que problemas maiores e mais complexos geram um número de lugares e transições praticamente intratáveis.

Já o trabalho recentemente apresentado por Hickmott [22] traz resultados significativos

em relação à rede gerada. A rede resultante terá sempre um número de lugares igual ao dobro de variáveis de estado do problema de planejamento, e ainda com a vantagem de não precisar da definição de operadores *mutex*.

Inspirados na eficiência dos planejadores baseados em heurísticas e explorando o potencial da técnica de análise de redes de Petri, foram implementadas heurísticas para guiar a ferramenta na construção do desdobramento.

CAPÍTULO 5

PLANEJAMENTO EM INTELIGÊNCIA ARTIFICIAL UTILIZANDO REDES DE PETRI CÍCLICAS

Este capítulo apresenta uma abordagem para o problema de planejamento utilizando redes de Petri cíclicas como estrutura de representação. A seção 5.1 descreve a motivação para o desenvolvimento da modelagem. As demais seções demonstram os passos para a construção do modelo juntamente com exemplos de utilização.

5.1 Desenvolvimento da modelagem

Dando continuidade aos trabalhos desenvolvidos dentro do grupo de pesquisa, a modelagem proposta teve como objetivo aperfeiçoar o modelo de representação utilizando redes de Petri. A eficiência da técnica de desdobramento para resolver problemas de alcançabilidade em redes de Petri foi o principal motivador para o desenvolvimento da modelagem.

Os trabalhos apresentados nas seções 4.1 e 4.2 descreveram duas propostas de planejadores baseados em redes de Petri acíclicas. Como a técnica de desdobramento pode ser aplicada com sucesso em redes de Petri cíclicas, a modelagem partiu para uma tradução do problema de planejamento em uma rede de Petri dessa classe.

Figura 5.1: Modelagem proposta

Na modelagem proposta, ilustrada na figura 5.1, o problema de planejamento descrito em *PDDL* é traduzido diretamente para uma rede de Petri cíclica seguindo regras de transformação, que serão descritas na seção 5.2. O problema de alcançabilidade na rede

de Petri obtida pela tradução é resolvido então por desdobramento que gera uma seqüência de transições da rede, mapeadas para ações do problema de planejamento que resultam no plano.

A modelagem pode ser dividida em dois processos: no primeiro passo a rede de Petri é gerada partir do *PDDL*; em seguida, o desdobramento da rede é feito pela ferramenta *MOLE*, modificada para gerar o plano conforme descrito na seção 4.3.

A implementação da modelagem foi totalmente desenvolvida no ambiente de planejamento IPE (*Ipê Planning Environment*) [31], por ser a plataforma padrão para o desenvolvimento de planejadores dentro do grupo de pesquisa.

O primeiro passo da tradução para a rede de Petri consiste em mapear as proposições e as ações do problema de planejamento em estruturas da rede. As proposições serão representadas por lugares, e as ações serão transições da rede.

Uma ação é representada por uma transição, sendo que as proposições que fazem parte da lista das pré-condições da ação serão os lugares que fazem parte do pré-conjunto da transição e as proposições que fazem parte da lista dos efeitos serão os lugares do pós-conjunto da transição.

A ação desempilhar(C,A), do domínio dos blocos descrito na seção 2.3, é representada por uma transição conforme ilustra a figura 5.2.

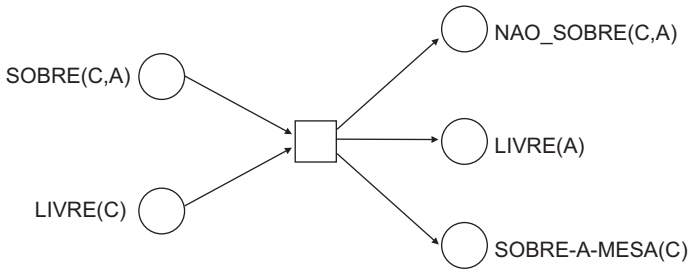


Figura 5.2: Ação desempilhar(C,A) modelada como uma transição

As proposições que são pré-condições ou efeitos negativos devem ser representadas de forma complementar à sua proposição positiva equivalente. Para isso são definidos lugares complementares que representam a proposição negativa de forma positiva, ou seja, um lugar que estará marcado caso a proposição negativa seja verdadeira.

O lugar complementar de uma proposição positiva recebe o nome da própria proposição acrescido da palavra "NAO_" à frente, representando assim uma proposição negada, por exemplo, o lugar NAO_SOBRE(C,A) é complementar ao lugar SOBRE(C,A). A complementaridade dos lugares é mantida pela premissa de que quando um lugar está marcado o outro não deve estar.

Com esse mapeamento inicial é possível verificar a existência de casos especiais que necessitam de maior atenção e que são tratados por meio das regras de transformação descritas a seguir.

5.2 Regras de transformação

O simples mapeamento de proposições e ações respectivamente como lugares e transições não é suficiente para modelar um problema de planejamento como uma rede de Petri.

Como vimos na seção 3.1, o comportamento de um sistema modelado em uma rede de Petri é simulado por meio do consumo e geração das marcas. Por isso, é preciso levar em conta a regra de disparo de transição (definição 4), em que todas as marcas nos lugares de entrada da transição são consumidas. Outro ponto importante a ser respeitado é a complementaridade entre os lugares que representam as proposições positivas e negativas.

Para cobrir as situações que precisam manter a relação entre os lugares complementares e as situações nas quais a influência do consumo de marcas pelas transições afetam a execução das ações, foram definidos quatro casos:

- **Caso 1 (Ações com pré-condições positivas que não fazem parte da lista de efeitos):** se uma proposição é pré-condição de uma ação, mas não é afetada por ela, essa proposição deve se manter verdadeira após a execução da ação. Caso tratado pela regra alfa, ilustrada na figura 5.3(a);

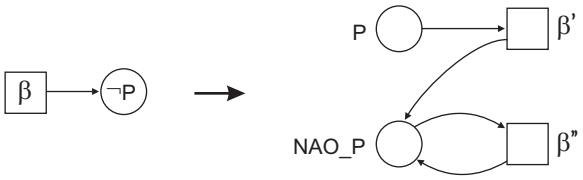
- **Caso 2 (Ações com efeitos negativos que não fazem parte da lista de pré-condições):** para manter a complementaridade entre as proposições positivas e negativas, se existe uma proposição que é um efeito negativo de uma ação, a sua proposição positiva deve ser uma pré-condição. Mas como uma proposição pode

assumir os valores verdadeiro ou falso, é preciso que existam duas ações, sendo que uma tem como pré-condição a proposição positiva e a outra a proposição negativa, como demonstrado pela regra beta na figura 5.3(b);

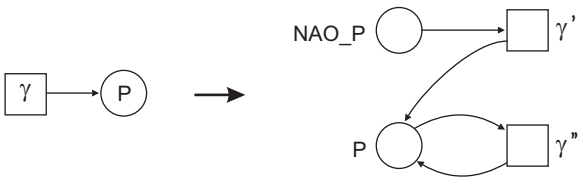
- **Caso 3 (Ações com efeitos positivos que não fazem parte da lista de pré-condições):** se uma proposição é um efeito positivo de uma ação, assim como no caso 2, sua proposição complementar deve fazer parte da lista de pré-condições. Da mesma forma, duas ações devem existir para manter a complementaridade, como contempla a regra gama apresentada na figura 5.3(c);

- **Caso 4 (Ações com pré-condições negativas que não fazem parte da lista de efeitos):** assim como no caso 1, a proposição que é pré-condição da ação, mas não é afetada por ela, deve se manter verdadeira após a execução da ação. A figura 5.3(d) ilustra a regra delta.

(a) Regra alfa



(b) Regra beta



(c) Regra gama

(d) Regra delta

Figura 5.3: Regras de transformação

As seções 5.2.1 até 5.2.4 descrevem caso a caso as regras de transformação e apresentam exemplos das transições resultantes do mapeamento de algumas ações do domínio logística descrito na figura 5.4.

```
(define (domain logistica)
(:requirements :strips :typing)
(:types caminhao - veiculo
        pacote
        veiculo - obj
        localidade - lugar
        lugar
        obj - objeto)

(:predicates (em ?obj - obj ?loc - lugar)
             (dentro ?pct - pacote ?vei - veiculo)
             (limpo ?obj - obj)
             (vazio ?vei - veiculo)
             (dep ?loc - lugar)
             (motor-ligado ?vei - veiculo)
             (tanque-cheio ?vei - veiculo))

(:action carregar-limpo
   :parameters (?pct - pacote ?cam - caminhao ?loc - lugar)
   :precondition (and (em ?cam ?loc) (em ?pct ?loc) (limpo ?cam)
                 (limpo ?pct) (vazio ?cam))
   :effect (and (not (em ?pct ?loc)) (dentro ?pct ?cam) (not (vazio ?cam))))

(:action carregar-sujo
   :parameters (?pct - pacote ?cam - caminhao ?loc - lugar)
   :precondition (and (em ?cam ?loc) (em ?pct ?loc) (vazio ?cam))
   :effect (and (not (em ?pct ?loc)) (dentro ?pct ?cam) (not (vazio ?cam))
           (not (limpo ?pct)) (not (limpo ?cam))))

(:action descarregar
  :parameters (?pct - pacote ?cam - caminhao ?loc - lugar)
  :precondition (and (em ?cam ?loc) (dentro ?pct ?cam))
  :effect (and (not (dentro ?pct ?cam)) (em ?pct ?loc) (vazio ?cam)))

(:action dirigir
  :parameters (?cam - caminhao ?loc-de - lugar ?loc-para - lugar)
  :precondition (and (em ?cam ?loc-de) (motor-ligado ?cam))
  :effect (and (not (em ?cam ?loc-de)) (em ?cam ?loc-para)
          (not (tanque-cheio ?cam))))

(:action lavar
  :parameters (?cam - caminhao ?loc - lugar)
  :precondition (and (em ?cam ?loc) (dep ?loc) (vazio ?cam))
  :effect (limpo ?cam))

(:action abastecer
  :parameters (?cam - caminhao ?loc - lugar)
  :precondition (and (not (motor-ligado ?cam)) (em ?cam ?loc) (dep ?loc))
  :effect (tanque-cheio ?cam))

(:action ligar-motor
  :parameters (?cam - caminhao)
  :precondition (not (motor-ligado ?cam))
  :effect (motor-ligado ?cam))

(:action desligar-motor
  :parameters (?cam - caminhao)
  :precondition (motor-ligado ?cam)
  :effect (not (motor-ligado ?cam)))
)
```

Figura 5.4: Domínio logística

O domínio usado como exemplo foi modificado a partir do domínio *logistics*[1] de forma que contivesse ações que modelassem as quatros regras.

Para simplificar o problema foram definidos apenas caminhões como veículos, descartando assim aviões e a possibilidade de transporte entre diferentes cidades. A figura 5.5 apresenta um problema para o domínio logística utilizando dois pacotes, um caminhão e três localidades.

```
(define (problem prob01)
   (:domain logistica)
   (:objects pacote1 pacote2 - pacote
             caminhao1 - caminhao
             local1 local2 local3 - localidade)
   (:init (em caminhao1 local1)
          (vazio caminhao1)
          (limpo caminhao1)
          (dep local3)
          (em pacote1 local1)
          (em pacote2 local2)
          (limpo pacote1))
   (:goal (and (em pacote1 local2)
          (em pacote2 local1)
          (limpo pacote1)
          (limpo caminhao1)
          (tanque-cheio caminhao1))))
```

Figura 5.5: Problema exemplo para o domínio logística

5.2.1 Regra alfa

A regra alfa transforma a ação que tem uma pré-condição positiva não afetada pela ação em uma transição em que o lugar que representa a pré-condição irá fazer parte ao mesmo tempo do pré e do pós-conjunto da transição. Esta regra deve ser aplicada para todas as proposições que são pré-condições da ação e que não fazem parte da lista de efeitos. A figura 5.6 ilustra o mapeamento para a ação com duas proposições neste caso.

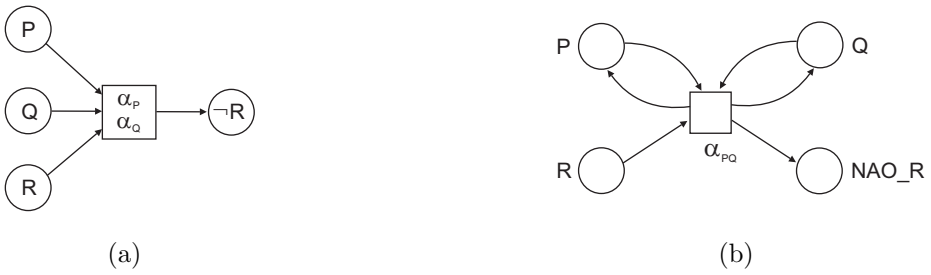


Figura 5.6: Regra alfa

[1]Domínio que trata de problemas de logística envolvendo o transporte de pacotes entre localidades e cidades, utilizando caminhões e aviões. Será descrito com mais detalhes na seção 6.1.

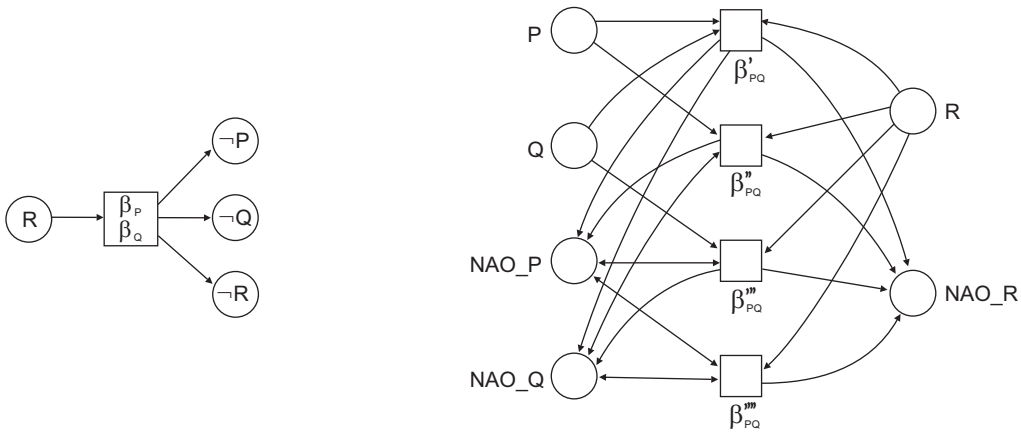
Como essa situação acontece em praticamente todas as ações, os exemplos de aplicação da regra alfa podem ser vistos juntamente com as demais regras.

As duas setas que ligam do lugar para ação e da ação para o lugar serão unificadas em uma única seta bi-direcionada para diminuir a quantidade de setas e assim facilitar a visualização.

5.2.2 Regra beta

A regra beta trata das ações com efeitos negativos que não fazem parte da lista de pré-condições. A ação que possuir esse caso especial deve ser transformada em duas transições, sendo que ambas terão o lugar complementar que representa o efeito negativo como um lugar do pós-conjunto da transição. Para manter a complementaridade, uma transição terá o lugar da proposição positiva como parte do pré-conjuto e a outra terá o lugar que representa a proposição negativa como pré-condição.

Assim como na regra alfa, a regra beta deve ser aplicada para todas as proposições que são efeitos negativos e que não fazem parte da lista de pré-condições. A cada proposição negativa duas novas transições são produzidas, gerando combinações entre as proposições, como pode ser visto na figura 5.7.



(a) (b)

Figura 5.7: Regra beta

Na figura 5.7(b), podem-se verificar as quatro transições com todas as combinações possíveis: {P, Q}, {P, NAO_Q}, {NAO_P, Q} e {NAO_P, NAO_Q}.

Como exemplo de aplicação, tem-se a ação carregar-sujo que possui duas proposições negativas como efeito, NAO_LIMPO(PACOTE1) e NAO_LIMPO(CAMINHAO1). A figura 5.8(a) descreve as quatro ações instanciadas. O mapeamento para as transições da rede de Petri estão na figura 5.8(b). A regra alfa pode ser vista pelo lugar EM(CAMINHAO1, LOCAL1).

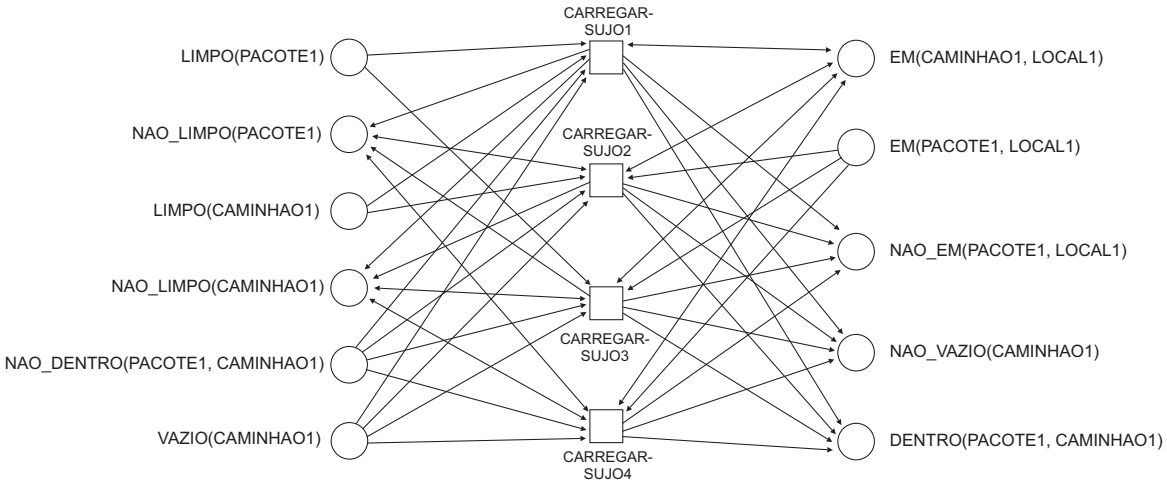
```
AÇÃO: CARREGAR-SUJO1 PACOTE1 CAMINHAO1 LOCAL1
PRÉ-CONDIÇÕES:
   EM(CAMINHAO1, LOCAL1)
   EM(PACOTE1, LOCAL1)
   NAO_DENTRO(PACOTE1, CAMINHAO1)
   VAZIO(CAMINHAO1)
   LIMPO(PACOTE1)
   LIMPO(CAMINHAO1)
EFEITOS:
   DENTRO(PACOTE1, CAMINHAO1)
   EM(CAMINHAO1, LOCAL1)
   NAO_EM(PACOTE1, LOCAL1)
   NAO_VAZIO(CAMINHAO1)
   NAO_LIMPO(PACOTE1)
   NAO_LIMPO(CAMINHAO1)

AÇÃO: CARREGAR-SUJO2 PACOTE1 CAMINHAO1 LOCAL1
PRÉ-CONDIÇÕES:
   EM(CAMINHAO1, LOCAL1)
   EM(PACOTE1, LOCAL1)
   NAO_DENTRO(PACOTE1, CAMINHAO1)
   VAZIO(CAMINHAO1)
   NAO_LIMPO(PACOTE1)
   LIMPO(CAMINHAO1)
EFEITOS:
   DENTRO(PACOTE1, CAMINHAO1)
   EM(CAMINHAO1, LOCAL1)
   NAO_EM(PACOTE1, LOCAL1)
   NAO_VAZIO(CAMINHAO1)
   NAO_LIMPO(PACOTE1)
   NAO_LIMPO(CAMINHAO1)

AÇÃO: CARREGAR-SUJO3 PACOTE1 CAMINHAO1 LOCAL1
PRÉ-CONDIÇÕES:
   EM(CAMINHAO1, LOCAL1)
   EM(PACOTE1, LOCAL1)
   NAO_DENTRO(PACOTE1, CAMINHAO1)
   VAZIO(CAMINHAO1)
   LIMPO(PACOTE1)
   NAO_LIMPO(CAMINHAO1)
EFEITOS:
   DENTRO(PACOTE1, CAMINHAO1)
   EM(CAMINHAO1, LOCAL1)
   NAO_EM(PACOTE1, LOCAL1)
   NAO_VAZIO(CAMINHAO1)
   NAO_LIMPO(PACOTE1)
   NAO_LIMPO(CAMINHAO1)

AÇÃO: CARREGAR-SUJO4 PACOTE1 CAMINHAO1 LOCAL1
PRÉ-CONDIÇÕES:
   EM(CAMINHAO1, LOCAL1)
   EM(PACOTE1, LOCAL1)
   NAO_DENTRO(PACOTE1, CAMINHAO1)
   VAZIO(CAMINHAO1)
   NAO_LIMPO(PACOTE1)
   NAO_LIMPO(CAMINHAO1)
EFEITOS:
   DENTRO(PACOTE1, CAMINHAO1)
   EM(CAMINHAO1, LOCAL1)
   NAO_EM(PACOTE1, LOCAL1)
   NAO_VAZIO(CAMINHAO1)
   NAO_LIMPO(PACOTE1)
   NAO_LIMPO(CAMINHAO1)
```

(a)



(b)

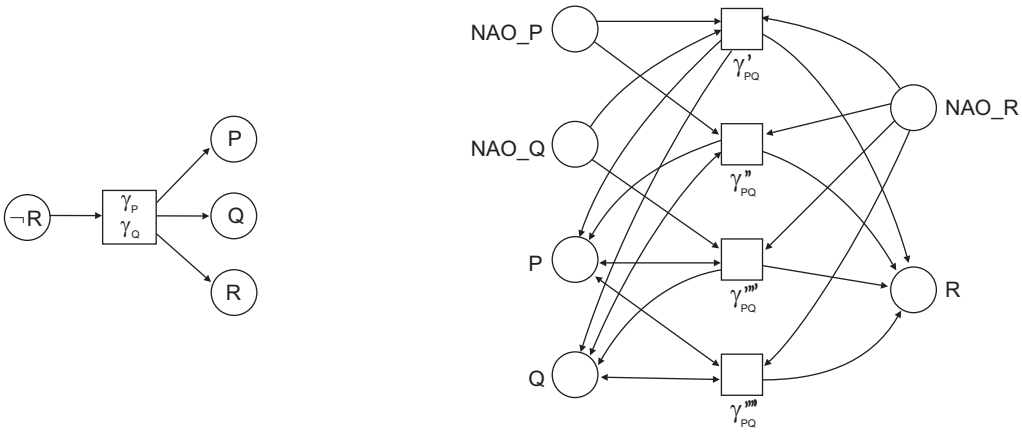
Figura 5.8: Aplicação da regra beta - ação carregar-sujo

A ação também possui a regra gama que devia ser aplicada para o efeito positivo DENTRO(PACOTE1, CAMINHAO1), mas foi simplificada para a existência da pré-condição NAO_DENTRO(PACOTE1, CAMINHAO1) para facilitar a visualização da regra beta.

5.2.3 Regra gama

A transformação de uma ação com efeito positivo que não faz parte da lista de pré-condições deve ser feita por meio da regra gama. Assim como na regra beta, a ação deve ser transformada em duas transições, que neste caso terão o lugar que representa o efeito positivo como um efeito da transição. Sendo que uma transição terá esse mesmo lugar de efeito positivo como pré-condição e a outra terá o lugar complementar que representa a proposição negativa como parte do pré-conjunto.

A figura 5.9 ilustra o mapeamento para uma ação com duas proposições como efeitos positivos que não fazem parte da lista de pré-condições. Nota-se que existe uma terceira proposição como efeito positivo, porém existe também a proposição negativa complementar como pré-condição, o que não sugere a aplicação da regra gama.



(a) (b)

Figura 5.9: Regra gama

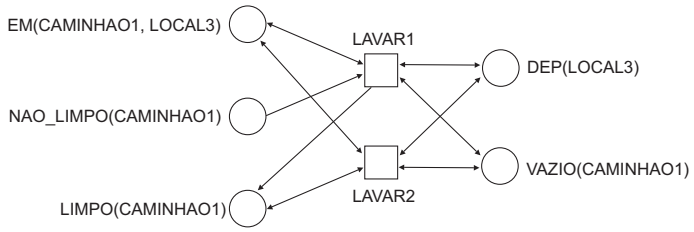
As quatro combinações possíveis: {NAO_P, NAO_Q}, {NAO_P, Q}, {P, NAO_Q} e {P, Q} podem ser observadas na figura 5.9(b).

Para exemplificar o uso da regra gama, a figura 5.10 mostra a ação lavar que possui o efeito positivo LIMPO(CAMINHAO1). As duas novas ações são criadas para cada um dos lugares complementares.

```
AÇÃO: LAVAR1(CAMINHAO1, LOCAL3)
PRÉ-CONDIÇÕES:
   EM(CAMINHAO1, LOCAL3)
   DEP(LOCAL3)
   VAZIO(CAMINHAO1)
   NAO_LIMPO(CAMINHAO1)
EFEITOS:
   EM(CAMINHAO1, LOCAL3)
   DEP(LOCAL3)
   VAZIO(CAMINHAO1)
   LIMPO(CAMINHAO1)

AÇÃO: LAVAR2(CAMINHAO1, LOCAL3)
PRÉ-CONDIÇÕES:
   EM(CAMINHAO1, LOCAL3)
   DEP(LOCAL3)
   VAZIO(CAMINHAO1)
   LIMPO(CAMINHAO1)
EFEITOS:
   EM(CAMINHAO1, LOCAL3)
   DEP(LOCAL3)
   VAZIO(CAMINHAO1)
   LIMPO(CAMINHAO1)
```

(a)



(b)

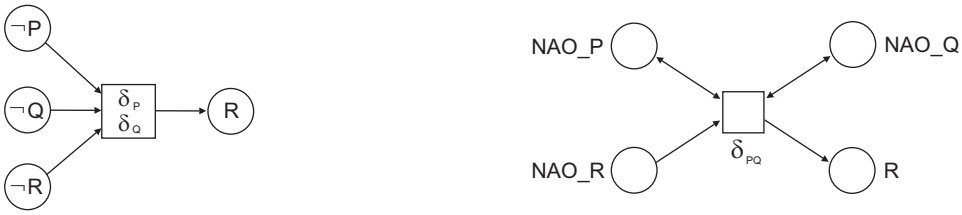
Figura 5.10: Aplicação da regra gama - ação lavar

É possível notar que a transição `LAVAR2` representa laços elementares, ou seja, o pós-conjunto da transição é igual ao seu pré-conjunto. Transições como esta podem ser eliminadas do modelo, simplificando o processo de desdobramento.

5.2.4 Regra delta

Da mesma forma que na regra alfa, a regra delta transforma a ação que tem uma pré-condição, neste caso negativa, não afetada pela ação em uma transição em que o lugar que representa a pré-condição irá fazer parte ao mesmo tempo do pré e do pós-conjunto da transição.

Deve ser aplicada para todas as proposições que são pré-condições da ação e que não fazem parte da lista de efeitos. Na figura 5.11, pode-se visualizar a transformação de uma ação com duas proposições negativas neste caso.



(a) (b)

Figura 5.11: Regra delta

A ação abastecer foi criada para podermos exemplificar a utilização da regra delta. A instanciação da ação e o mapeamento para a transição são apresentados na figura 5.12.

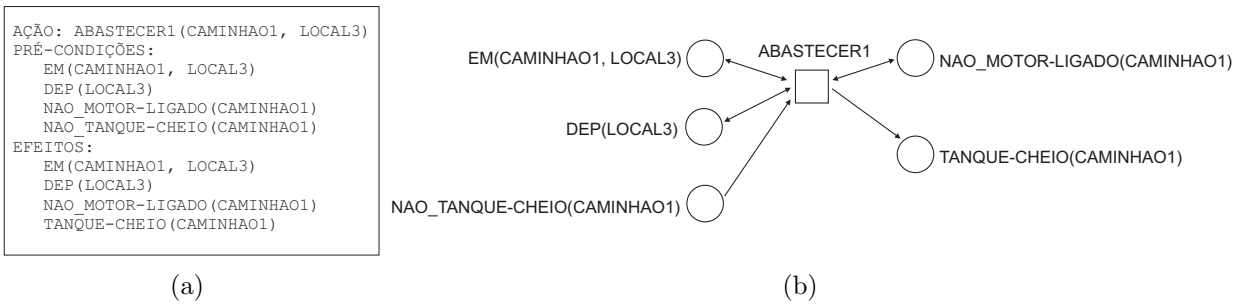


Figura 5.12: Aplicação da regra delta - ação abastecer

Neste exemplo, já foi aplicada também a simplificação por laço elementar para a segunda transição criada pela regra gama no efeito positivo TANQUE-CHEIO(CAMINHAO1).

5.3 Adaptações no desdobramento

Após a tradução do problema de planejamento em uma rede de Petri, o segundo passo na modelagem é resolver o problema de alcançabilidade por meio de desdobramento, utilizando a ferramenta *MOLE* (apresentada na seção 3.6). Mas para isso, são necessárias adaptações que irão auxiliar o processo de desdobramento na busca pela solução que será então mapeada para a solução do problema de planejamento inicial, resultando no plano.

Como descrito na seção 4.3, é preciso criar uma condição de parada para o desdobramento quando a marcação ou sub-marcação desejada for encontrada. O *MOLE* disponibiliza a opção de terminar o desdobramento quando uma determinada transição for alcançada.

A primeira adaptação, que deve ser feita ainda na rede, é criar uma transição final em que suas pré-condições são os lugares que representam as proposições do estado objetivo do problema de planejamento. A transição criada foi denominada *FIM* e contém apenas pré-condições e nenhum efeito.

Como o desdobramento cria o prefixo garantindo que os lugares tem no máximo uma transição de entrada, quando a transição *FIM* for alcançada, uma busca regressiva partindo dos lugares de seu pré-conjunto em direção aos lugares da marcação inicial irá

encontrar a seqüência inversa das transições disparadas. É importante destacar que não é possível identificar as transições concorrentes, resultando assim planos parcialmente ordenados.

Com essas alterações já é possível utilizar a ferramenta *MOLE* para encontrar a solução para o problema de planejamento modelado em uma rede de Petri. Originalmente o algoritmo de desdobrameno implementado pelo *MOLE* ordena os items a serem expandidos de tal maneira que cada camada é explorada inteiramente antes da próxima, fazendo o equivalente a uma busca em largura. Essa busca exaustiva é realizada em virtude de o processo de desdobramento ser total na construção do prefixo finito completo.

No caso de encontrar o plano para o problema de planejamento, a produção do prefixo completo não é necessária. O algoritmo pode ser otimizado para uma busca orientada ao objetivo por meio da utilização de heurísticas que podem guiar a seqüência de expansões feitas pelo desdobramento. A fila de expansão do *MOLE* foi reestruturada para usar a heurística h^1, recomendada como a mais eficiente no artigo de Hickmott e colegas [22].

Na heurística h^1, descrita na seção 2.6, o custo dos lugares objetivo é considerado como zero, e o custo de cada lugar da rede corresponde à soma do custo do menor caminho da marcação inicial até o lugar em questão mais o custo para ir do lugar até o objetivo.

Para implementar a heurística h^1 no desdobramento, a fila de expansão é ordenada de acordo com a soma dos pesos dos lugares marcados em cada expansão. Como utilizado pelo planejador *PUP* (seção 4.3), a ordem de expansões $\prec$ é substituída por uma ordem $\prec_h$ tal que:

Definição 15 ($\prec_h$) *Sendo h uma heurística e C uma configuração, definimos $g(C) = \sum_{e \in C} cost(\varphi(e))$, e $f(C) = g(c) + h(\texttt{Mark}(C))$. Definimos $C \prec_h C'$ se e somente se $f(C) < f(C')$ ou $f(C) = f(C')$ e $|C| < |C'|$. [22]*

Ou seja, cada marcação é ordenada pela soma das heurísticas de seus lugares marcados, marcações com os menores valores são encaminhadas para o começo da fila de expansão.

Para auxiliar na realização dos experimentos, foram disponibilizadas também a opção para definir um limite de tempo de execução e número de expansões; saída em arquivo com a formatação do plano em *PDDL* e arquivos com resultados para gerar os gráficos.

5.4 Construção do modelo

Para exemplificar a construção do modelo, utilizamos o problema proposto por Weld [51]. O problema consiste em preparar um jantar surpresa, sendo que os objetivos são: preparar um jantar, embrulhar um presente e remover o lixo. As quatro ações possíveis são cozinhar, embrulhar, carregar ou empurrar o lixo.

A ação cozinhar requer mãos limpas (maosLimpas) e obtém o jantar (jantar). A ação embrulhar tem que ser realizada em silêncio (silencio) e obtém o presente embrulhado (presente). Empurrar elimina o lixo (negando lixo) usando um carrinho de mão e produz barulho (negando silencio). Carregar também elimina o lixo (negando lixo), mas suja as mãos (negando maosLimpas).

No estado inicial do problema, as proposições maosLimpas, lixo e silencio são verdadeiras, como pode ser visto na figura 5.13, que mostra a rede de Petri obtida pela tradução do problema do jantar. Os lugares nao_jantar e nao_presente também fazem parte da marcação inicial.

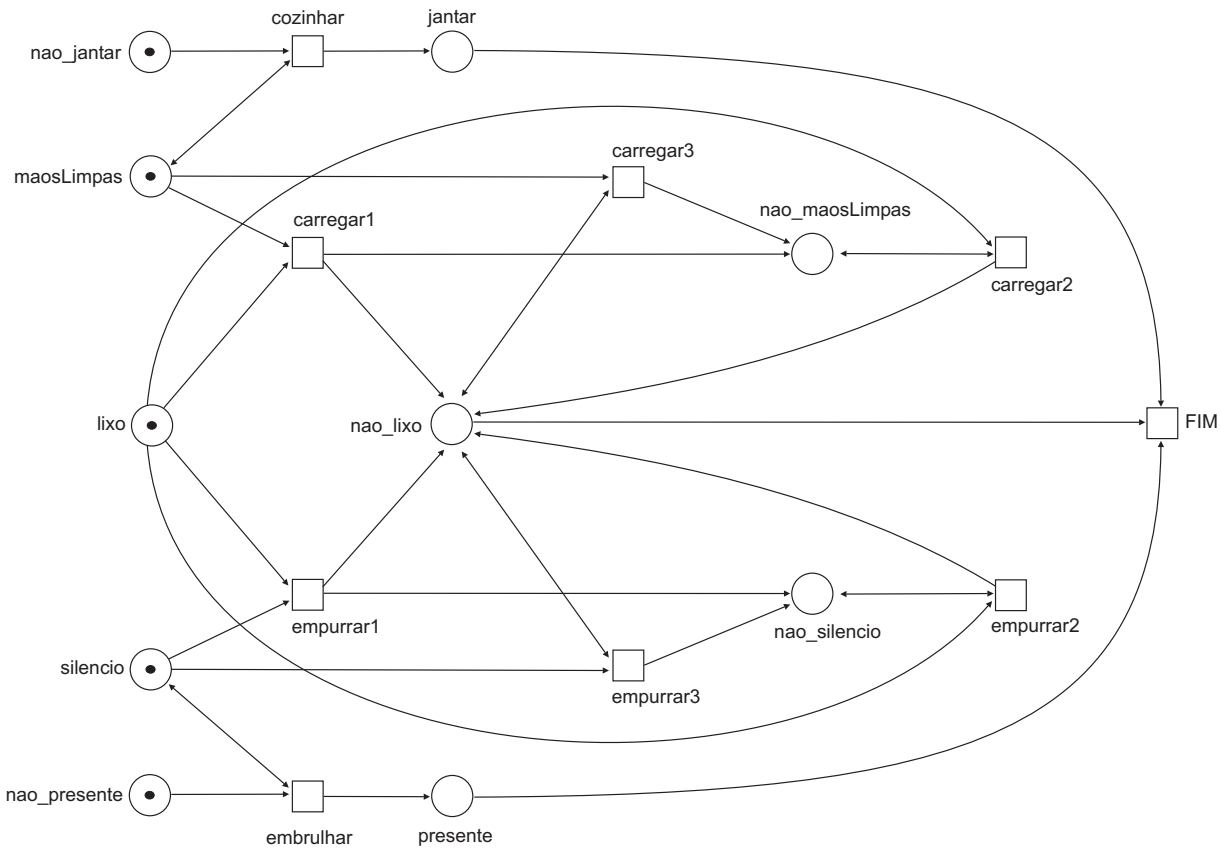


Figura 5.13: Rede de Petri para o problema do jantar

As proposições do estado objetivo, jantar, presente e nao_lixo são as pré-condições da transição *FIM*, criando assim a condição de parada para o desdobramento. Com a rede de Petri descrita no formato do arquivo `ll_net`, o próximo passo é executar a ferramenta *MOLE* com os parâmetros necessários para o desdobramento produzir o plano para o problema.

A figura 5.14 apresenta o desdobramento feito pelo *MOLE* utilizando a heurística h^1 com as transições que fazem parte do plano sombreadas para diferenciar das transições de corte que surgem durante o desdobramento.

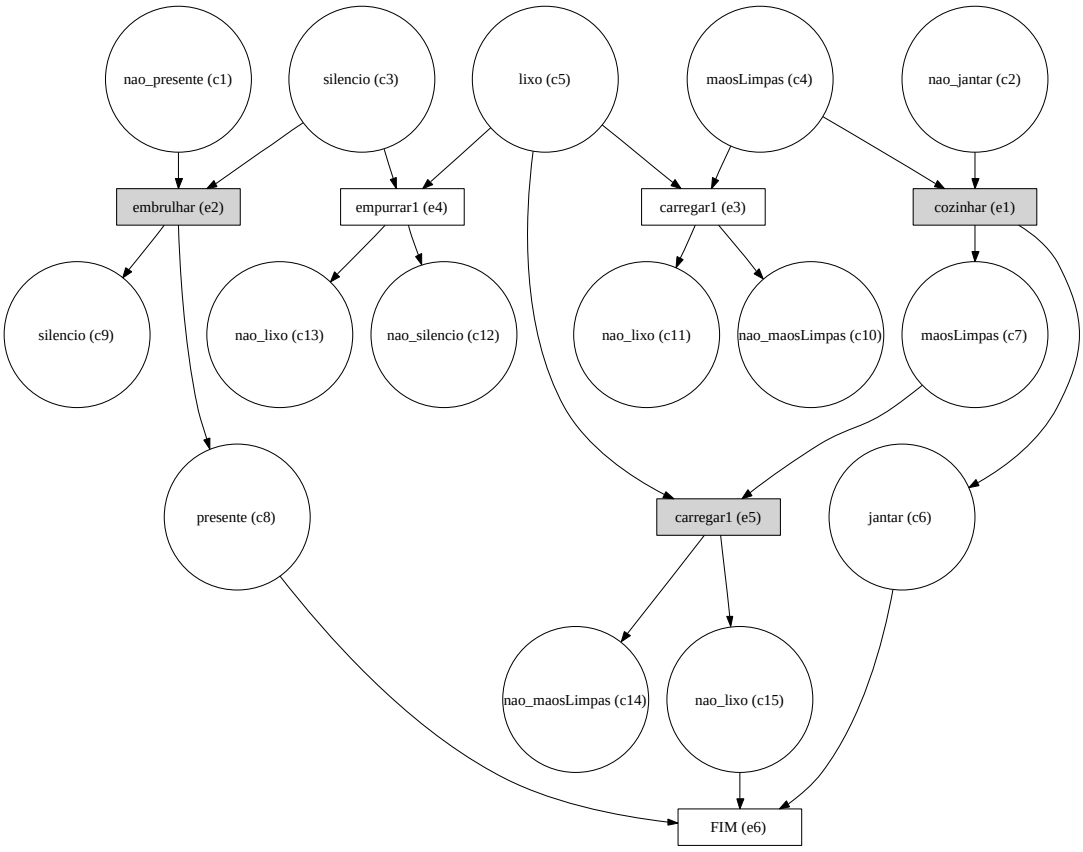


Figura 5.14: Desdobramento com plano para o problema do jantar

Para a ferramenta ser executada como um planejador, a saída foi personalizada para criar o arquivo com o plano no formato *PDDL*, como mostra a figura 5.15.

```
0:(EMBRULHAR)
1:(COZINHAR)
2:(CARREGAR)
```

Figura 5.15: Plano para o problema do jantar

5.5 Considerações

Este capítulo descreveu uma modelagem para representar problemas de planejamento utilizando redes de Petri. Ao contrário dos métodos desenvolvidos dentro do grupo de pesquisa, descritos no capítulo 4, a abordagem produz uma rede cíclica.

A utilização de redes cíclicas para modelar problemas de planejamento é um desafio incontornável, uma vez que os problemas reais contêm inevitavelmente estruturas repetitivas ou cíclicas.

No trabalho apresentado por Hickmott no congresso *IJCAI* em 2007 [22], a aplicação da técnica de desdobramento foi um dos principais resultados. As dificuldades associadas ao problema de alcançabilidade em redes cíclicas foram em partes superadas pelo uso de desdobramento.

CAPÍTULO 6

RESULTADOS EXPERIMENTAIS

Neste capítulo é mostrada uma análise dos resultados obtidos pela modelagem implementada para domínios de planejamento descritos na linguagem *STRIPS*. Os domínios propostos para os experimentos são descritos na seção 6.1.

A bateria de testes foi realizada para analisar o tamanho da rede, número de expansões feitas pelo desdobramento, tempo de execução e tamanho da solução. Os resultados são apresentados na seção 6.3.

6.1 Domínios

Os domínios utilizados nos experimentos foram propostos nas competições de planejamento *AIPS-2000* [2] e *IPC-2002* [30]. Embora a cada competição são introduzidos novos desafios para avaliar o desempenho do planejadores participantes, as descrições em *PDDL* para problemas na versão em *STRIPS* são sempre disponibilizadas. A seguir a descrição dos seis domínios propostos:

- ***Blocks:*** Consiste de um conjunto de blocos sobre uma mesa, todos quadrados e do mesmo tamanho. Os blocos podem ser empilhados, mas apenas um bloco pode ficar diretamente em cima de outro. Uma garra robô pode pegar um bloco e depositá-lo sobre a mesa ou em cima de outro bloco. A garra só pode segurar um bloco por vez, ou seja, não pode capturar um bloco que tenha outro em cima dele. O objetivo é empilhar blocos de acordo com uma determinada especificação.

- ***Driverlog:*** Existem pacotes que devem ser transportados por caminhões entre diferentes localidades. Os caminhões são dirigidos por motoristas que podem estar em diversos locais e devem caminhar para chegar aos locais onde estão os caminhões. Os caminhos por onde andam os motoristas são diferentes das vias por onde seguem

os caminhões, mas que possuem ligações por meio de determinadas conexões entre os mesmos.

- ***Elevator:*** Conhecido também como *Miconic*, modela o funcionamento de elevadores. Neste domínio existe um elevador e pessoas em diferentes andares cujo objetivo é levar as pessoas aos diferentes andares de forma eficiente.

- ***Logistics:*** Representa problemas de logística de pacotes. Existe um número de pacotes que deve ser transportado a diferentes lugares. O transporte é feito por caminhões, quando as localidades são na mesma cidade; ou por aviões, quando é preciso mudar de cidade, sendo que os aviões só podem se movimentar entre aeroportos.

- ***Rovers:*** Existem veículos de exploração autônomos (*rovers*) que devem navegar a superfície de um planeta a procura de amostras de solo e rochas para serem enviadas a um módulo de pouso.

- ***Satellite:*** Inspirado na programação de satélites espaciais. Os satélites são equipados com instrumentos a bordo para capturar imagens de diferentes fenômenos. Para capturar uma imagem, o satélite deve ligar o instrumento adequado, sendo que apenas um instrumento por vez pode estar ligado, calibrá-lo e então tirar a foto.

A escolha dos domínios foi determinada por serem considerados pela comunidade científica como bons representantes de problemas de planejamento em geral. Foram escolhidos também pelas características da ações, uma vez que era necessário testar as regras de transformação descritas na seção 5.2.

O domínio *blocks* foi utilizado por ser um problema clássico e relativamente simples, apresentando apenas ações mapeadas pela regra gama. Os domínios *driverlog*, *elevator*, *logistics* e *rovers* aplicam as regras alfa e gama.

As regras alfa, beta e gama estão presentes simultaneamente no domínio *satellite*. Não foram encontrados, entre os domínios disponibilizados nas competições, problemas que possuíssem ações com pré-condições negativas para aplicar a regra delta.

6.2 Metodologia

Os experimentos foram realizados em problemas de diferentes níveis de complexidade para todos os domínios propostos, sendo que cada domínio contém uma quantidade diferente de problemas. A tabela 6.1 apresenta o número de problemas testados nos experimentos.

Domínios	Problemas
blocks	35
driverlog	20
elevator	130
logistics	28
rovers	20
satellite	20

Tabela 6.1: Quantidade de problemas por domínio

Geralmente o problema é identificado pelo nome `prob<nome-dominio>-id`, sendo que `id` é um número identificador, que normalmente está na forma "`n-i`" em que "`n`" é o tamanho do problema e "`i`" é o i-ésimo problema deste tamanho. Por exemplo, `probBLOCKS-4-2` é o segundo problema do domínio *blocks* de tamanho 4.

Os gráficos de resultados estão arranjados pelo tamanho do problema, sendo que problemas múltiplos com o mesmo tamanho estão arranjados em posições fracionárias no eixo de identificação para poderem ficar adjacentes no gráfico.

Os domínios *driverlog*, *rovers* e *satellite* não estão organizados pelo tamanho do problema, por isso são identificados por um número denominado ID na competição.

6.3 Análise dos resultados

Inicialmente os experimentos foram feitos para avaliar o tamanho da rede obtida pela modelagem. O número de lugares e transições foi utilizado como parâmetro para medir o crescimento da rede à medida que aumentasse a complexidade dos problemas.

Para analisar o número de expansões feitas pelo desdobramento, o tempo de execução e o tamanho da solução foram realizadas comparações com o *MOLE* em sua execução

normal, ou seja, utilizando busca em largura, e utilizando a heurística h^1, conforme modificações especificadas na seção 5.3.

Para todos os experimentos, o tempo de execução foi limitado a trinta minutos. Pontos que não aparecem nos gráficos foram retirados por excederem esse limite. A tabela 6.2 apresenta a quantidade total de problemas resolvidos por domínio.

	Problemas		
Domínios	Propostos	Resolvidos	
blocks	35	32	91,42%
driverlog	20	9	45%
elevator	130	130	100%
logistics	28	10	35,71%
rovers	20	8	40%
satellite	20	5	25%

Tabela 6.2: Número de problemas resolvidos nos experimentos

A seguir os resultados dos experimentos são demonstrados por meio de gráficos analisados para cada um dos parâmetros de avaliação. Os critérios de análise são descritos à medida que são apresentados nos gráficos.

6.3.1 Tamanho da rede

Os gráficos apresentados nas figuras 6.1 a 6.6 mostram o crescimento da rede por meio do número de lugares e transições.

O número de lugares da rede está diretamente relacionado com a instanciação das proposições, sendo que para cada proposição instanciada deve-se criar a seu respectivo lugar complementar, duplicando assim a quantidade de lugares. O processo de instanciação consiste em fazer a combinação de cada parâmetro com todos os objetos possíveis.

As transições são criadas a partir das ações instanciadas, obedecendo às regras de transformação para cada uma, gerando assim todas as combinações possíveis, podendo gerar algumas transições que não serão usadas e algumas que não serão válidas. Entre as simplificações, as transições que representam um laço elementar podem ser cortadas da rede, assim como transições que possuem conflito de lugares complementares.

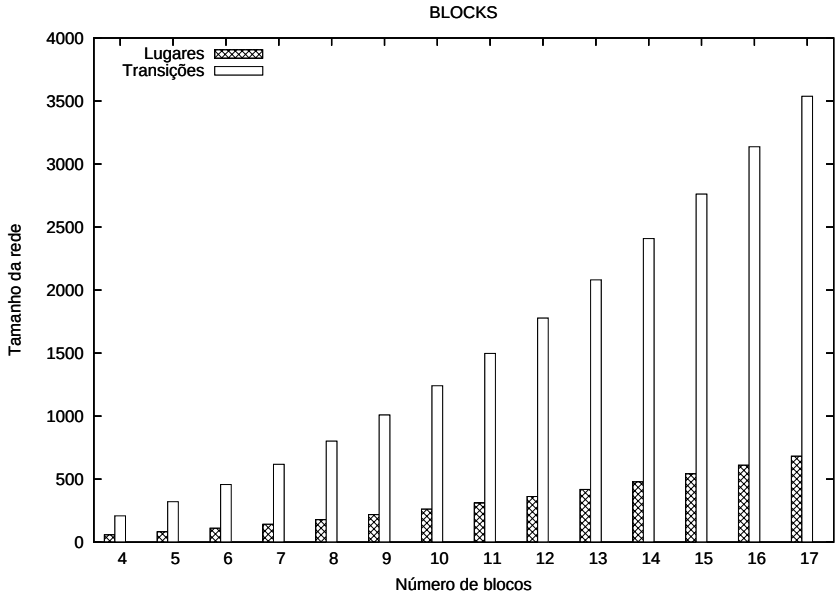


Figura 6.1: Tamanho da rede para o domínio *blocks*

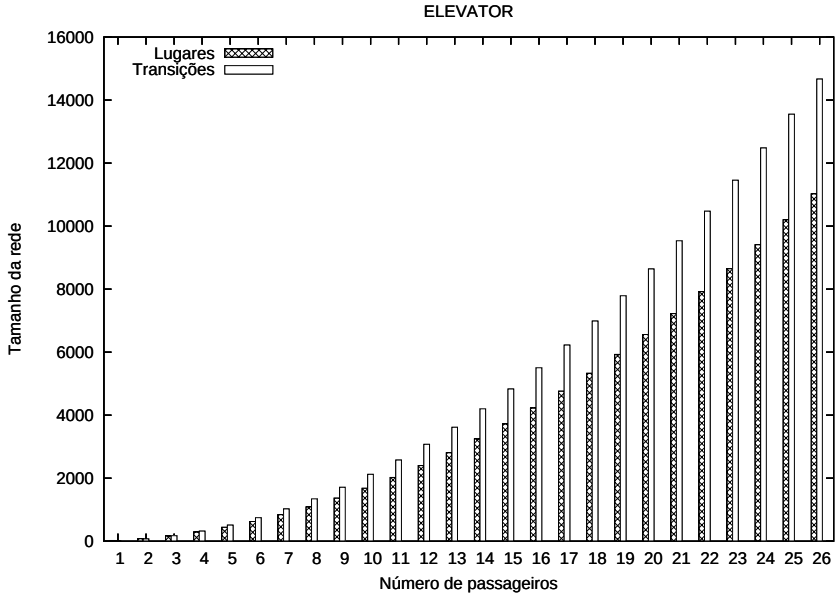


Figura 6.2: Tamanho da rede para o domínio *elevator*

Para os domínios *blocks* e *elevator*, a complexidade dos problemas aumenta respectivamente de acordo com a quantidade de blocos e passageiros. Por exemplo, o domínio *blocks* possui cinco predicados, quatro ações e apenas o tipo bloco como parâmetro.

No caso dos lugares, instancia-se os predicados: `(handempty)`, que não possui parâmetro; `(on ?x ?y)` que referencia dois blocos, e `(ontable ?x)`, `(clear ?x)` e `(holding ?x)` que possuem apenas um bloco como parâmetro.

Para o problema de quatro blocos, a instanciação gera $1 + (1 \times 4^2) + (3 \times 4^1) = 29$ proposições, que duplicadas, resultam em 58 lugares. A rede com cinco blocos terá $2 \times (1 + (1 \times 5^2) + (3 \times 5^1)) = 82$ lugares, com seis terá 110 e assim por diante.

As transições para o domínio *blocks* são instanciadas a partir das quatro ações com a combinação de todos os objetos possíveis do tipo bloco. As ações `pick-up(?x)` e `put-down(?x)` possuem um bloco como parâmetro, `stack(?x ?y)` e `unstack(?x ?y)` têm dois blocos. Além dos parâmetros, as regras de transformação também influenciam na quantidade de transições.

Todas as ações aplicam a regra gama, porém com quantidade diferente de proposições: `pick-up(?x)` têm uma, `put-down(?x)` e `stack(?x ?y)` apresentam três proposições neste caso cada uma e `unstack(?x ?y)` possui duas. Para o problema de quatro blocos, a instaciação das ações gera 40 transições e as regras de transformação aumentam mais 168, totalizando 208 transições na rede. Com isso, verifica-se que o aumento do tamanho da rede se deve principalmente à aplicação das regras.

O domínio *logistics* apresenta um aumento diferente no tamanho da rede, como pode ser visto no gráfico da figura 6.3.

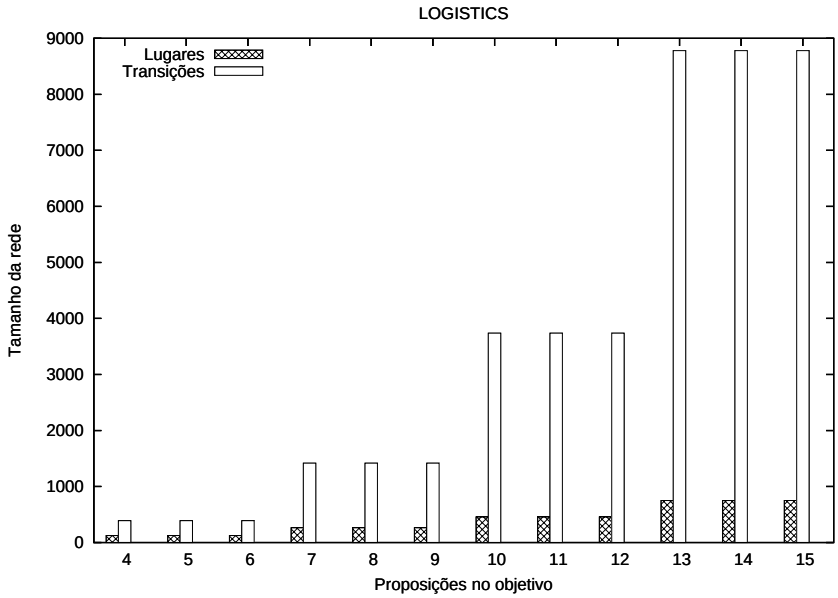


Figura 6.3: Tamanho da rede para o domínio *logistics*

A diferença está na complexidade do problema que aumenta não pela quantidade de objetos e sim pelo número de proposições no objetivo. Por exemplo, os problemas

de tamanho quatro, cinco e seis apresentam o mesmo número de pacotes, caminhões, localidades, cidades, aeroportos e aviões, por essa razão o tamanho da rede permanece estável. Somente quando a quantidade de objetos aumenta que o tamanho da rede altera.

Os demais domínios, *driverlog*, *rovers* e *satellite*, não possuem um padrão no aumento da complexidade dos problemas, por isso são referenciados apenas pelo identificador na competição.

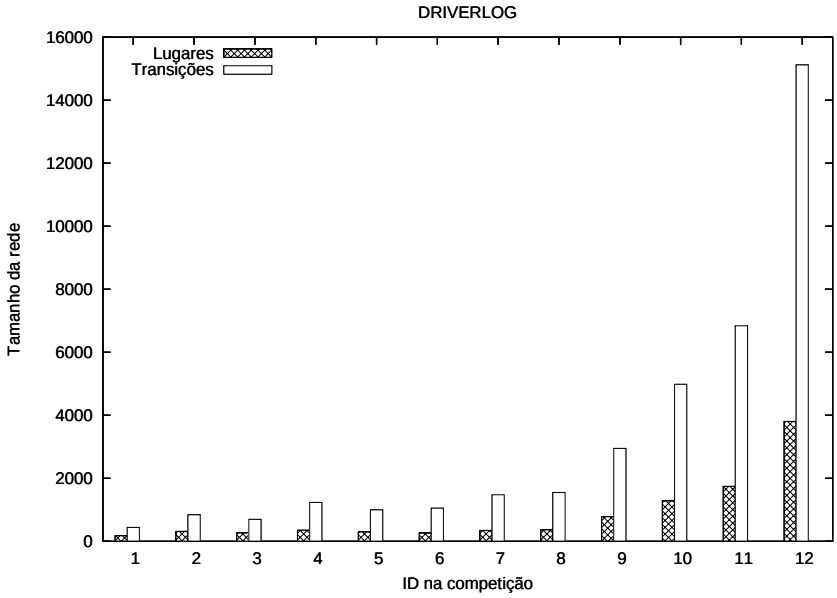


Figura 6.4: Tamanho da rede para o domínio *driverlog*

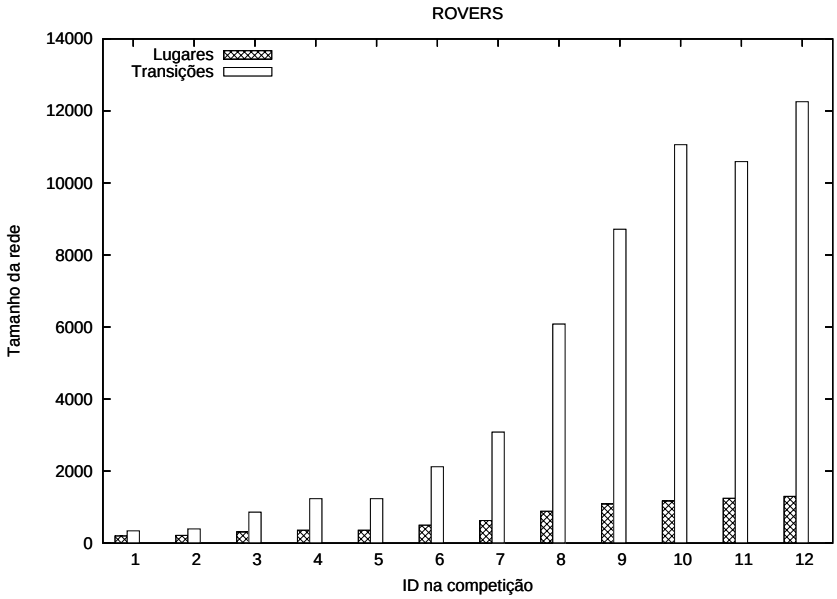


Figura 6.5: Tamanho da rede para o domínio *rovers*

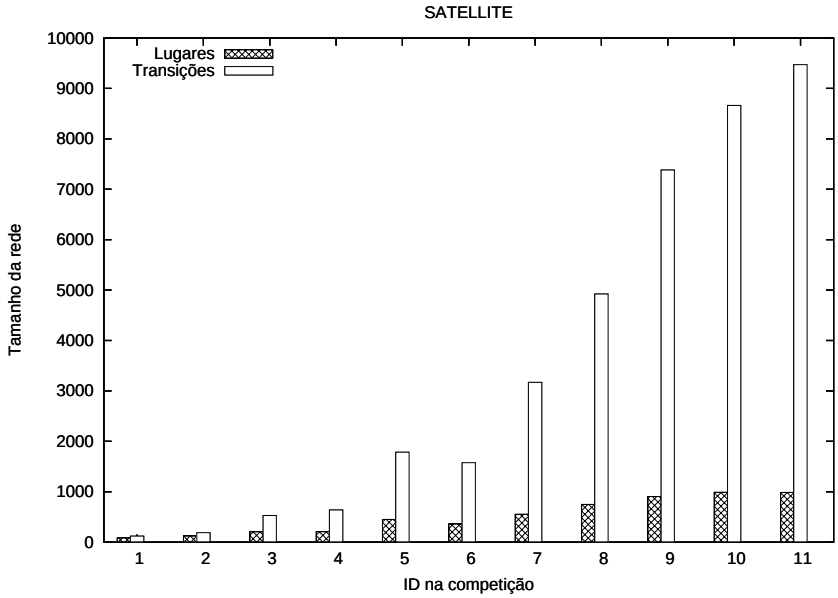


Figura 6.6: Tamanho da rede para o domínio *satellite*

É possível notar uma pequena variação na quantidade de lugares e transições entre os diferentes problemas. Isso acontece porque o número de objetos não aumenta de forma seqüencial.

Por exemplo, no domínio *satellite*, do problema cinco para o problema seis, a quantidade de objetos do tipo instrumento diminui enquanto a quantidade de tipos de direção aumenta. Como vimos anteriormente, essa quantidade influencia diretamente no processo de instanciação de lugares e transições.

6.3.2 Expansões no desdobramento

O número de expansões no desdobramento está relacionado com o processo de busca. A análise dos resultados foi feita por comparação entre os processos de busca exaustiva e busca com heurística.

Nos gráficos, a legenda mole 0 identifica o *MOLE* utilizando busca em largura, enquanto mole h1 referencia o *MOLE* utilizando a heurística h^1.

As figuras 6.7 a 6.12 apresentam os gráficos com as expansões realizadas para todos os problemas que encontraram a solução dentro do limite de trinta minutos de execução.

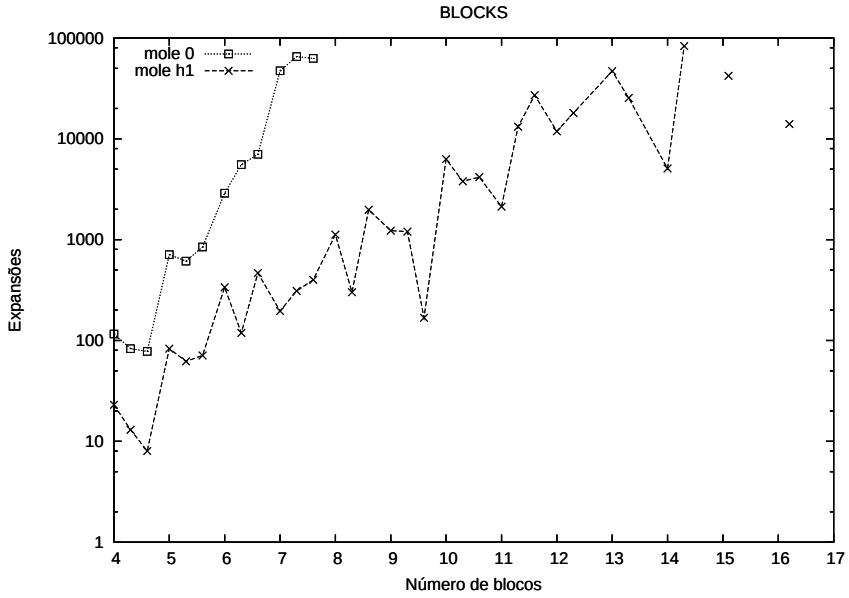


Figura 6.7: Expansões para o domínio *blocks*

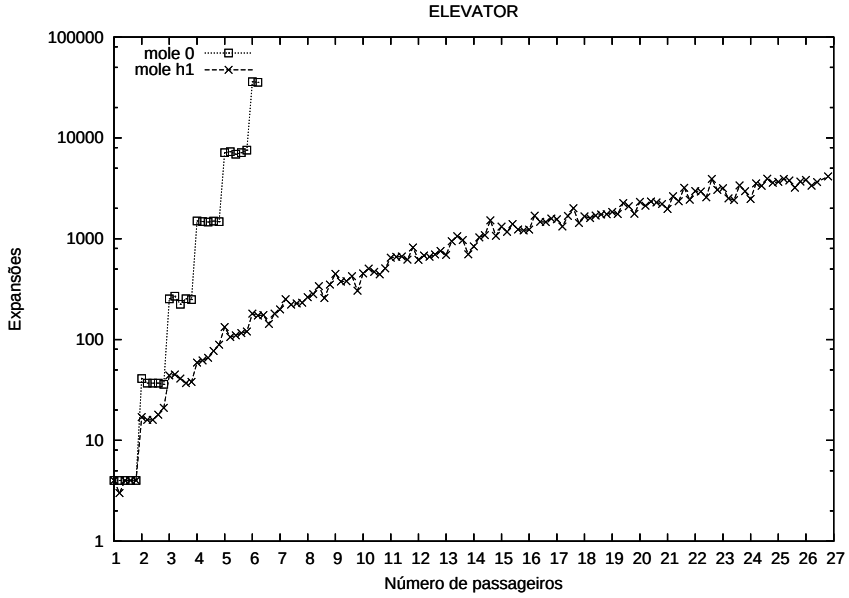


Figura 6.8: Expansões para o domínio *elevator*

Os domínios *blocks* e *elevator* demonstram que a busca em largura aumenta de forma exponencial o número de expansões rapidamente. Para os primeiros problemas, o número chega facilmente na casa de dez mil expansões em menos de dez segundos (o tempo de execução será detalhado na seção 6.3.3).

Em alguns casos, a busca guiada pela heurística h^1 não apresentou melhorias em relação à busca em largura. A figura 6.9 ilustra a situação encontrada no domínio *logistics*.

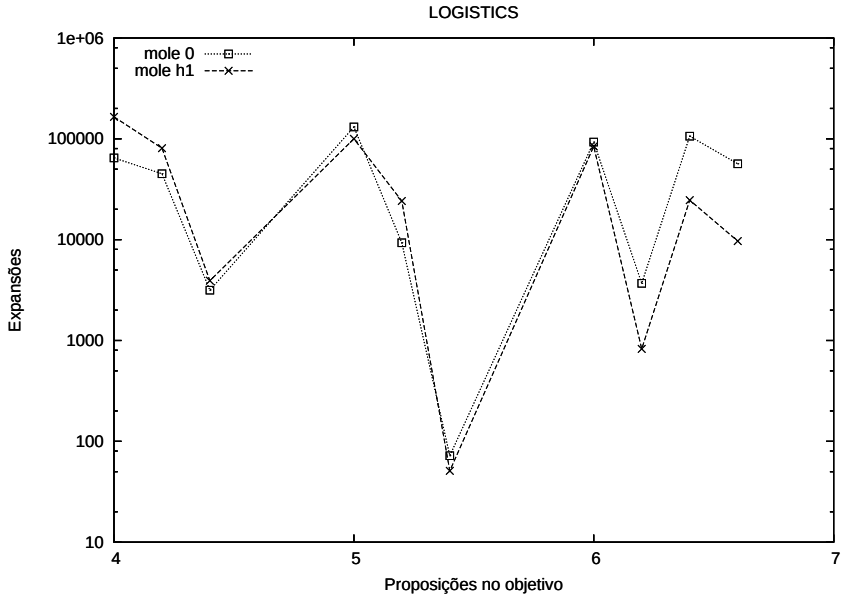


Figura 6.9: Expansões para o domínio *logistics*

A diferença entre o número de expansões com o *MOLE* na versão original e com o *MOLE* utilizando a heurística h^1 é mínima. Foi o único domínio que teve a mesma quantidade de problemas resolvidos pelas duas ferramentas. Observa-se também uma variação entre problemas do mesmo tamanho mas com objetivos diferentes, o que pode sugerir a dificuldade do desdobramento em alcançar múltiplos objetivos.

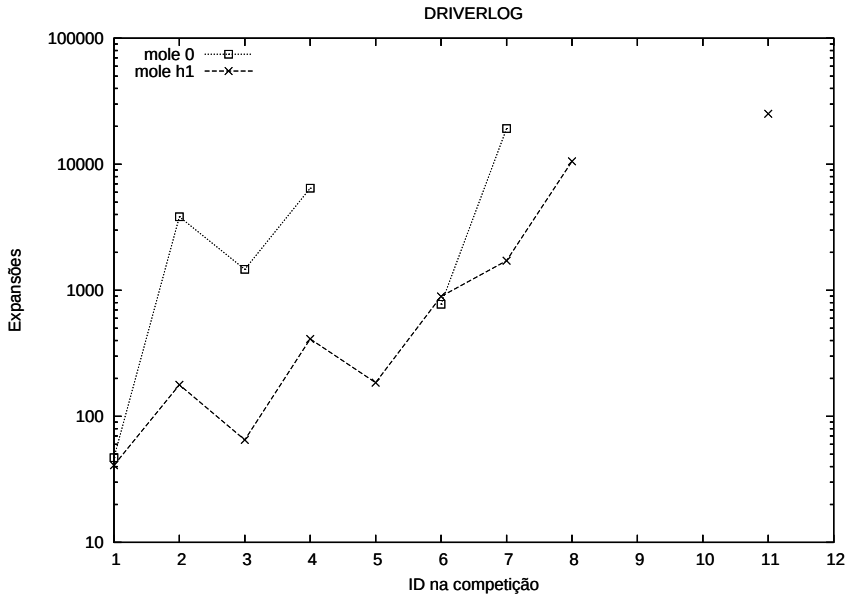


Figura 6.10: Expansões para o domínio *driverlog*

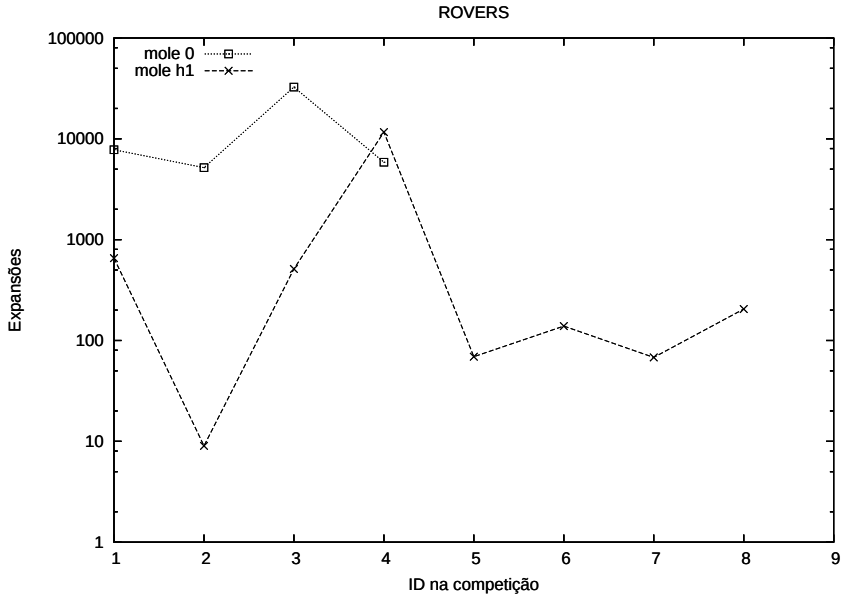


Figura 6.11: Expansões para o domínio *rovers*

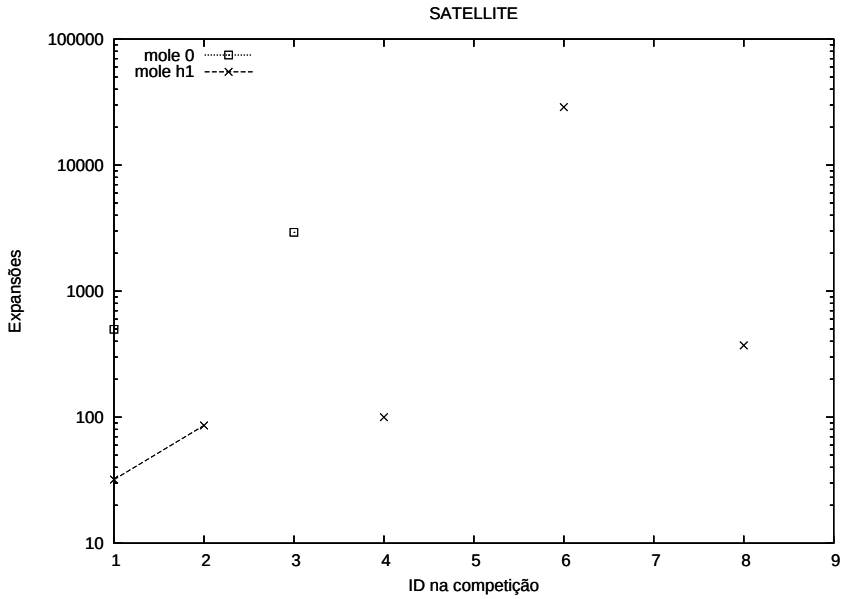


Figura 6.12: Expansões para o domínio *satellite*

Como os domínios *driverlog*, *rovers* e *satellite* não apresentaram resultados satisfatórios, tendo apenas 20 a 40% de problemas resolvidos, não é possível analisar com detalhes, mas podemos afirmar que a heurística reduz o número de expansões na maioria dos casos.

6.3.3 Tempo de execução

Os gráficos a seguir mostram o tempo de execução para todos os problemas analisados (figuras 6.13 a 6.18). Pode-se afirmar que o tempo de execução é uma conseqüência do número de expansões. Em todos os casos em que as expansões aumentaram exponencialmente, o limite de tempo de execução foi ultrapassado.

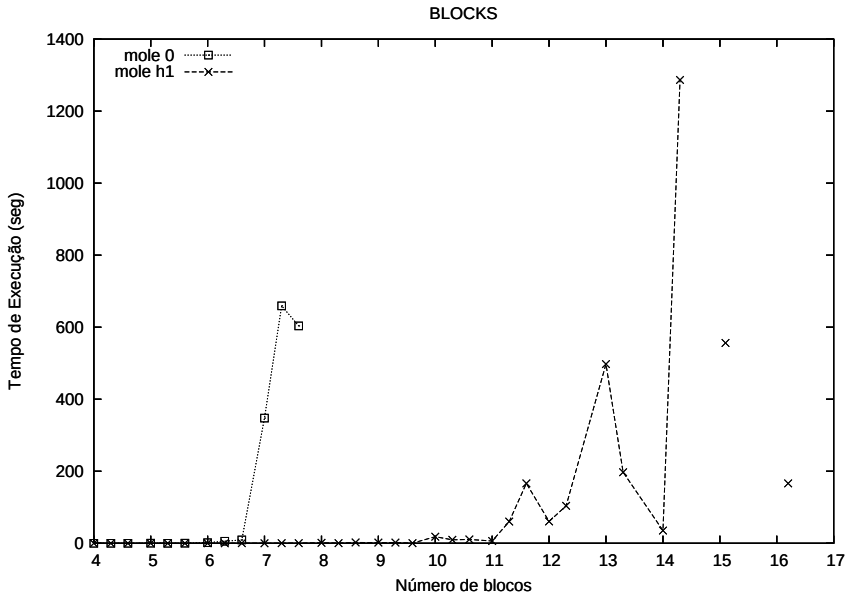


Figura 6.13: Tempo de execução para o domínio *blocks*

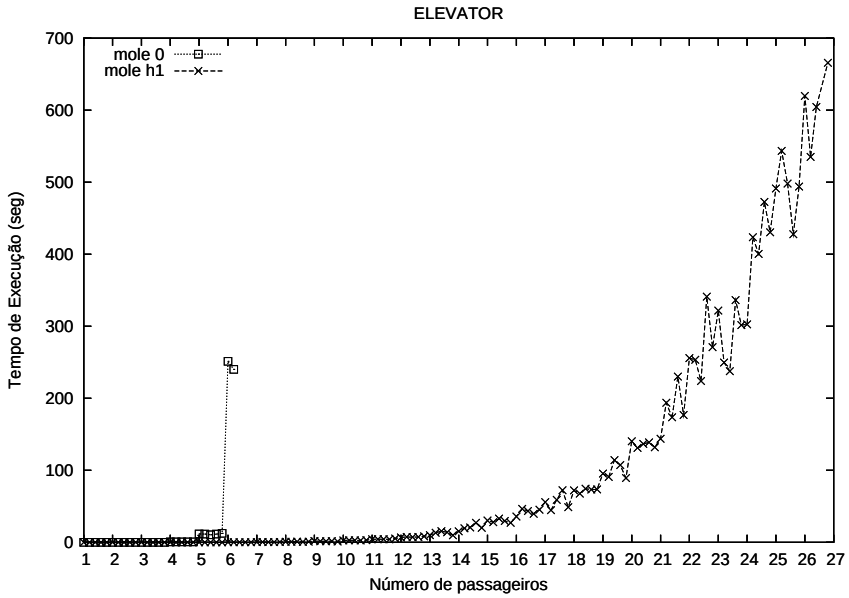


Figura 6.14: Tempo de execução para o domínio *elevator*

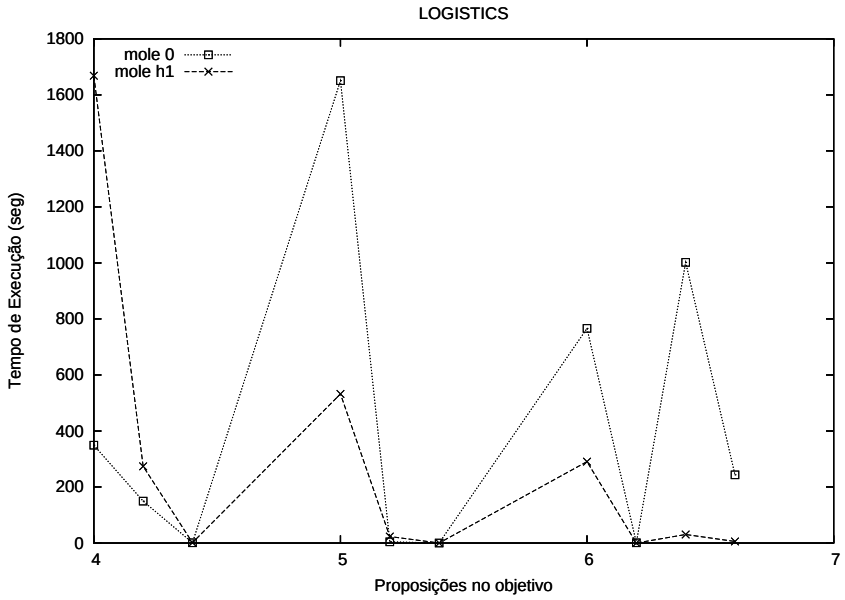


Figura 6.15: Tempo de execução para o domínio *logistics*

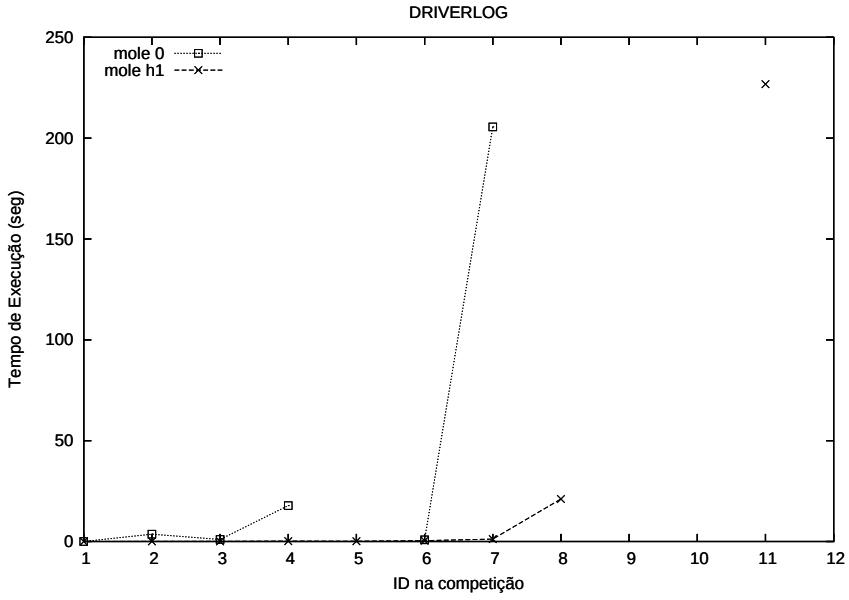


Figura 6.16: Tempo de execução para o domínio *driverlog*

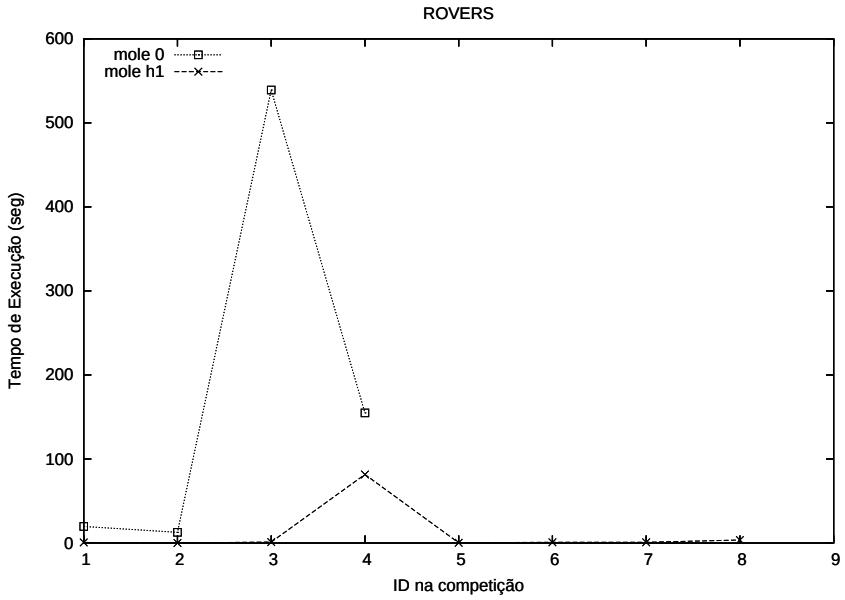


Figura 6.17: Tempo de execução para o domínio *rovers*

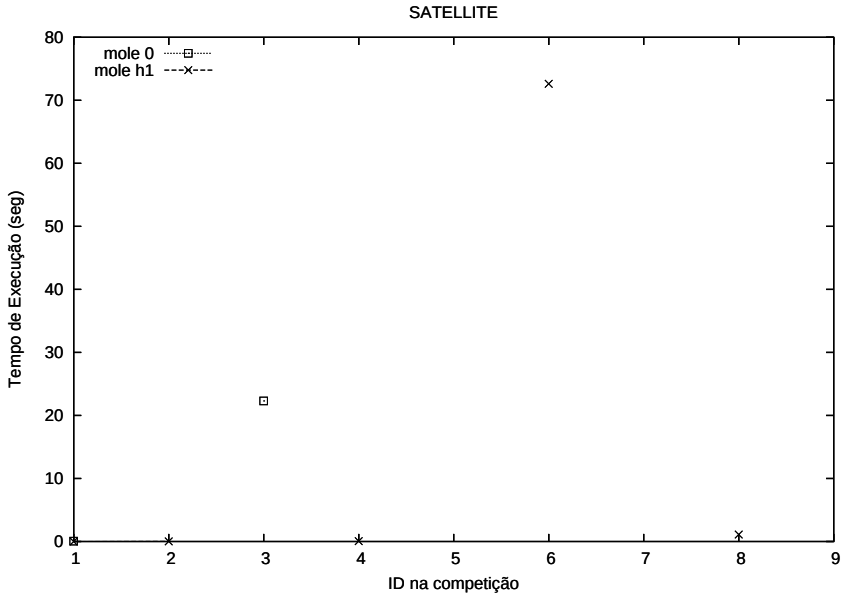


Figura 6.18: Tempo de execução para o domínio *satellite*

6.3.4 Tamanho do plano

Os gráficos das figuras 6.19 a 6.24 apresentam a comparação do tamanho do plano encontrado pelos desdobramentos com e sem heurística. O tamanho do plano é o número total de ações da seqüência encontrada, sem considerar concorrência.

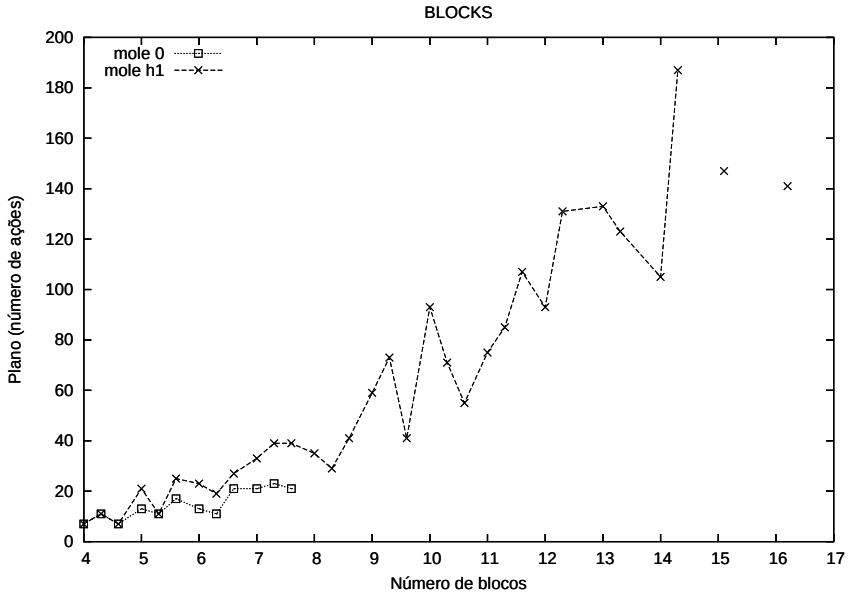


Figura 6.19: Tamanho do plano para o domínio *blocks*

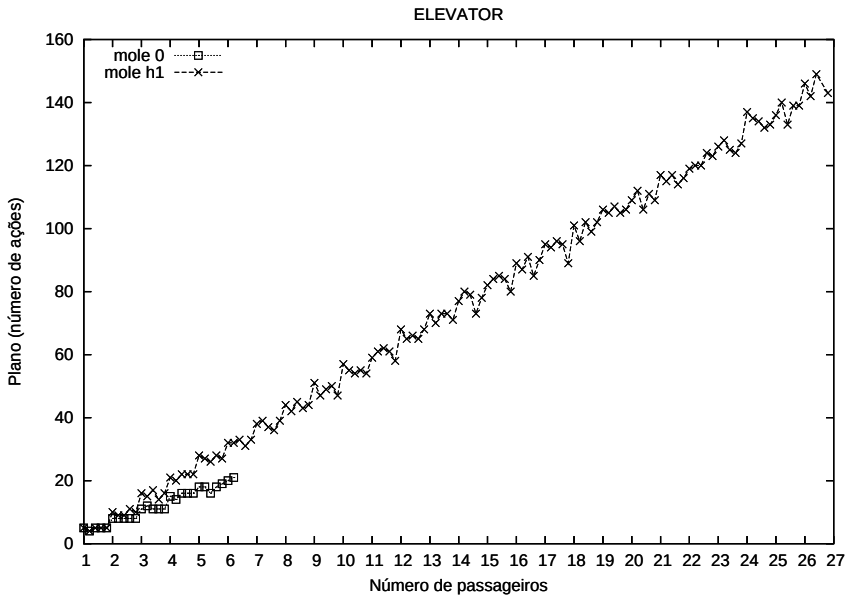


Figura 6.20: Tamanho do plano para o domínio *elevator*

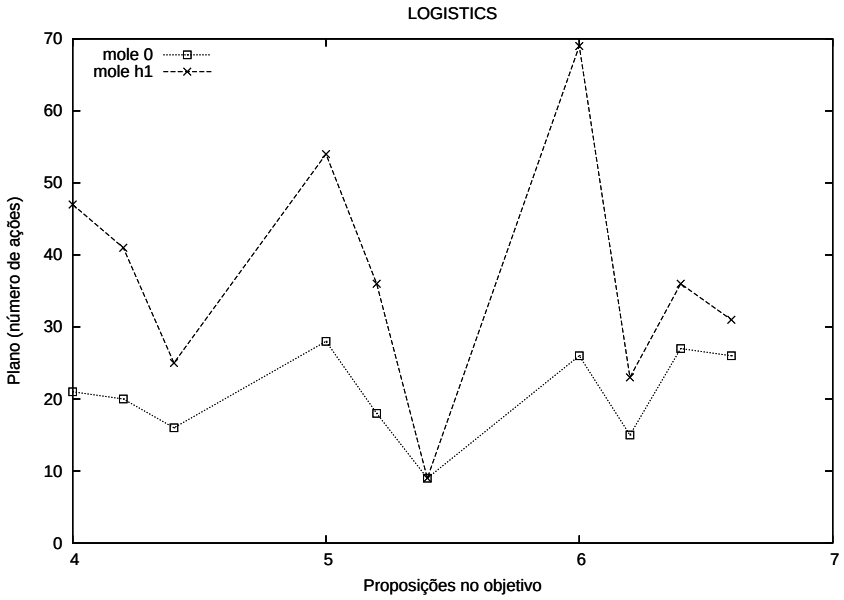


Figura 6.21: Tamanho do plano para o domínio *logistics*

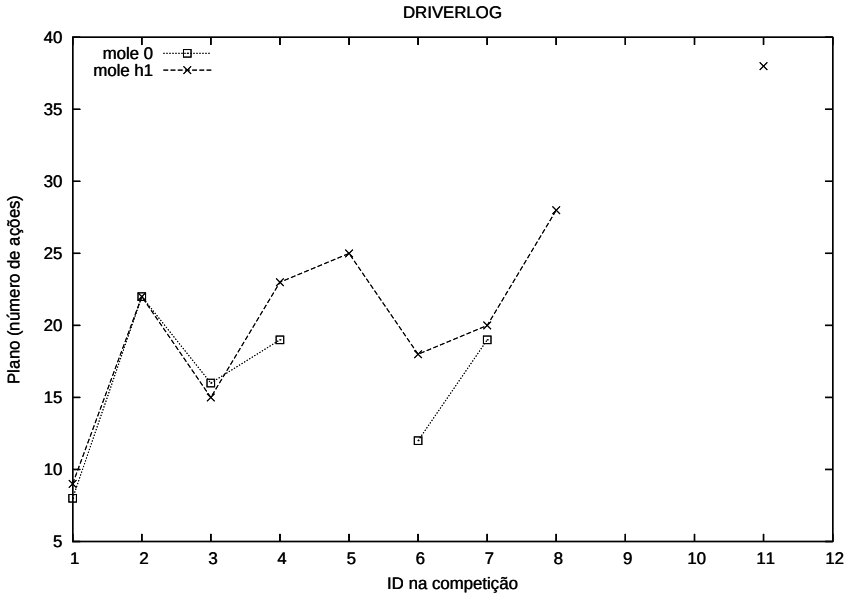


Figura 6.22: Tamanho do plano para o domínio *driverlog*

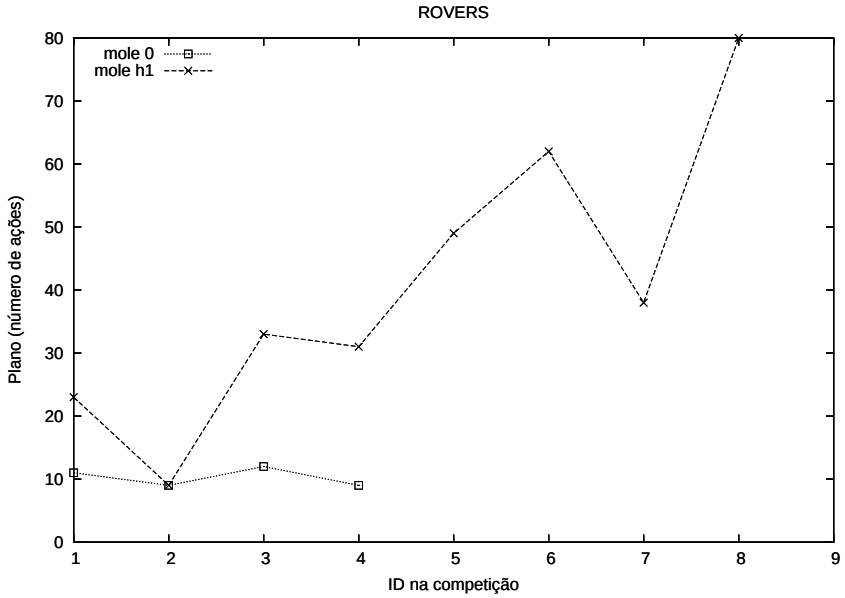


Figura 6.23: Tamanho do plano para o domínio *rovers*

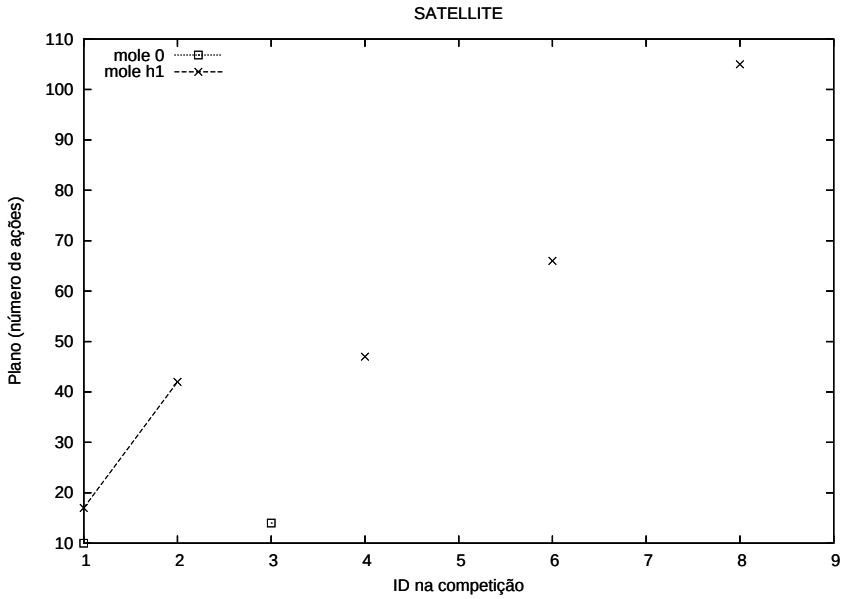


Figura 6.24: Tamanho do plano para o domínio *satellite*

Como era esperado, em todos os casos, quando o desdobramento com busca exaustiva encontra a solução, o tamanho do plano é menor que o encontrado pela heurística. Na verdade, os planos encontrados pelo *MOLE* utilizando a h^1 têm tamanhos bem próximos ao plano ótimo, porém com a vantagem de o tempo de execução ser menor.

6.4 Considerações

Os resultados obtidos apresentados neste capítulo mostram o desempenho do planejador desenvolvido, conforme descrito no capítulo 5. A comparação dos resultados ficou limitada à utilização da heurísitica no processo de desdobramento da rede de Petri.

A análise dos resultados demonstra que a abordagem proposta resolve corretamente grande parte dos problemas testados, trazendo resultados significativos para as pesquisas desenvolvidas no grupo de pesquisa.

CAPÍTULO 7

CONCLUSÃO

Este trabalho apresentou uma abordagem para resolver problemas de planejamento em inteligência artificial utilizando desdobramento de redes de Petri. A modelagem proposta tem como princípio a tradução do problema de planejamento em uma rede de Petri cíclica e segura. O problema de alcançabilidade dessa rede equivalente é resolvido por desdobramento por meio da ferramenta *MOLE*.

Para alcançar os objetivos do trabalho, foi feita uma revisão dos principais conceitos da área de planejamento, além de investigar trabalhos relacionados com a abordagem de planejamento utilizando redes de Petri.

No formalismo das redes de Petri, a técnica de desdobramento teve que ser estudada com maior atenção. O domínio completo do algoritmo de desdobramento foi necessário para avaliar a sua utilização e finalmente compreender a implementação da ferramenta *MOLE* para que pudesse ser adapatada de acordo com os critérios apresentados por Hickmott [22].

As modificações no código da ferramenta geraram algumas dificuldades em virtude da definição e uso de muitas estruturas inicialmente complexas para representar os elementos da rede e do desdobramento. A implementação das opções de parada e criação do arquivo com o plano, bem como a inclusão da heurística h^1, tiveram que ser feitas novamente pelo grupo de pesquisa, uma vez que a versão da ferramenta *MOLE*, modificada, utilizada por Hickmott não estava disponível.

A modelagem apresentada demonstrou que as quatro regras de transformação definidas para o processo de tradução permitem obter uma rede de Petri equivalente ao problema de planejamento descrito em *PDDL*. Porém, o desenvolvimento do trabalho resultou em um modelo equivalente ao apresentado por Hickmott. Embora a proposta seja equivalente em relação ao tamanho da rede, ela contribui como uma nova perspectiva às pesquisas

do grupo para os problemas de planejamento e alcançabilidade, já que esta é a primeira abordagem desenvolvida na equipe, voltada a redes de Petri cíclicas, que obtém resultados satisfatórios.

Os resultados experimentais mostram que a abordagem é promissora, como também comprovam os resultados obtidos por Töws em sua dissertação de mestrado [49], recentemente defendida no Programa de Pós-Graduação em Informática da Universidade Federal do Paraná.

É possível verificar que, pelos resultados analisados, é evidente a necessidade do uso de heurísticas na geração do desdobramento. Adquire importância fundamental, na continuação deste trabalho, a busca de novas heurísticas ou a adaptação daquelas atualmente utilizadas em planejamento.

Ainda como trabalho futuro, destacamos que é imprescindível uma implementação da técnica de desdobramento desenvolvida pelo grupo de pesquisa para que a técnica possa ser adaptada conforme nossas necessidades. Como exemplo, pode-se citar a necessidade de manter as informações de concorrência entre as ações no plano encontrado pelo desdobramento, de forma a obter planos parcialmente ordenados.

Outro caminho que pode ser explorado está na combinação dos conceitos da técnica de desdobramento com as estruturas criadas pela rede de planos. Esta proposta tem como premissa a possibilidade de gerar uma rede de Petri totalmente "desdobrada", contendo todas as marcações alcançáveis, eliminando a etapa de geração da rede cíclica e seu posterior desdobramento.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Arteiro, R. D., Souza, F., Rosa, N. S. e Maciel, P. Utilizando redes de Petri para modelagem de desempenho de middleware orientado a mensagem. *In WPerformance*, páginas 1–21, Rio de Janeiro, RJ, 2007.

[2] Bacchus, F. AIPS'00 planning competition: The fifth international conference on artificial intelligence planning and scheduling systems. *AI Magazine*, 22(3):47–56, 2001.

[3] Beek, P. e Chen, X. CPLAN: A constraint programming approach to planning. *Proceedings of the 16th National Conference on AI - AAAI'99*, páginas 585–590, Orlando, Florida, USA, 1999.

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

[5] Bonet, B. e Geffner, H. HSP: Planning as heuristic search. *In Entry at the AIPS-98 Planning Competition*, Pittsburgh, Pennsylvania, USA, 1998.

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

[7] Cardoso, J. e Valette, R. *Redes de Petri*. Editora da UFSC, 1997.

[8] Carvalho, C. S. Algoritmos genéticos para solução de problemas de alcançabilidade em uma determinada classe de redes de Petri acíclicas. Dissertação de Mestrado, Universidade Federal do Paraná - UFPR, Curitiba, PR, Brasil, fevereiro de 2007.

[9] Castilho, M. A., Künzle, L. A. e Silva, F. A Petri net based representation for planning problems. *V International Conference on Knowledge Based Computer Systems - KBCS'04*, Hyderabad, India, 2004.

[10] Cormen, T., Leiserson, C., Rivest, R. e Stein, C. *Algoritmos: Teoria e Prática*. Elsevier, Rio de Janeiro, 2nd edition, 2002.

[11] Esparza, J., Römer, S. e Vogler, W. An improvement of McMillan's unfolding algorithm. *Tools and Algorithms for Construction and Analysis of Systems*, páginas 87–106, 1996.

[12] Esparza, J., Römer, S. e Vogler, W. An improvement of McMillan's unfolding algorithm. *Formal Methods in System Design*, 20(3):285–310, 2002.

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

[14] Gerevini, A. e Long, D. Plan constraints and preferences in PDDL3. Technical report, Department of Electronics for Automation, University of Brescia, Italy, 2005.

[15] Gerevini, A., Saetti, A. e Serina, I. Planning through stochastic local search and temporal action graphs. *Journal of Artificial Intelligence*, (20):291–341, 2003.

[16] Ghallab, M., Nau, D. e Traverso, P. *Automated Planning: Theory and Practice*. Morgan Kaufmann, San Francisco, 2004.

[17] Grahlmann, B. e Best, E. Format descriptions. Appendix of: PEP Documentation and User Guide 1.8, 1998.

[18] Green, C. C. Application of theorem proving to problem solving. *Proceedings of the 1st International Joint Conference on Artificial Intelligence - IJCAI*, Washington, D.C., 1969.

[19] Gustin, G. D. B. Aplicação de redes de Petri interpretadas na modelagem de sistemas de elevadores em edifícios inteligentes. Dissertação de Mestrado, Escola Politécnica da Universidade de São Paulo, São Paulo, SP, Brasil, 1999.

[20] Haslum, P. e Geffner, H. Admissible heuristics for optimal planning. páginas 140–149. AAAI Press, 2000.

[21] Hickmott, S. L. Concurrent planning using Petri net unfoldings. *International Conference on Automated Planning and Scheduling - ICAPS'06*, Cumbria, Inglaterra, 2006.

[22] Hickmott, S. L., Rintanen, J., Thiebaux, S. e White, L. B. Planning via Petri net unfolding. *20th International Joint Conference on Artificial Intelligence - IJCAI'07*, Hyderabad, India, 2007.

[23] Hoffmann, J. e Nebel, B. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence*, (14):253–302, 2001.

[24] Howell, R. R. The complexity of problems involving structurally bounded and conservative Petri nets. *Information Processing Letters*, (39):309–315, 1991.

[25] Kautz, H. e Selman, B. Planning as satisfiability. *Proceedings of the 10th Eur. Conf. AI*, páginas 359–363, Vienna, Austria, 1992.

[26] Kautz, H. e Selman, B. Unifying SAT-based and graph-based planning. *Proceedings of the 16th International Joint Conference on Artificial Intelligence - IJCAI'99*, páginas 318–325, Stockholm, Sweden, 1999.

[27] Khomenko, V. *Model Checking Based on Prefixes of Petri Net Unfoldings*. Tese de Doutorado, University of Newcastle upon Tyne, Newcastle upon Tyne, UK, fevereiro de 2003.

[28] Khomenko, V. e Koutny, M. Towards an efficient algorithm for unfolding Petri nets. *Lecture Notes in Computer Science*, 2154:366–380, 2001.

[29] Lipton, R. The reachability problem requires exponential space. Technical Report 62, Yale University, 1976.

[30] Long, D. e Fox, M. The 3rd international planning competition: results and analysis. *Journal of Artificial Intelligence Research*, 20:1–59, 2003.

[31] Marynowski, J. E. Ambiente de planejamento IPE. Dissertação de Mestrado, Universidade Federal do Paraná - UFPR, Curitiba, PR, Brasil, novembro de 2004.

[32] McDermott, D. PDDL - the planning domain definition language. *AIPS-98 - Planning Competition Comittee*, 1998.

[33] McMillan, K. L. A technique of state space search based on unfolding. *Formal Methods in System Design*, 6(1):45–65, 1995.

[34] Montaño, R. Aplicação de fórmulas não-clausais em planejamento com redes de Petri. Dissertação de Mestrado, Universidade Federal do Paraná - UFPR, Curitiba, PR, Brasil, setembro de 2006.

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

[36] Pádua, S. I. D., Silva, A. R. Y., Porto, A. J. V. e Inamasu, R. Y. O potencial das redes de Petri em modelagem e análise de processos de negócio. *Gestão & Produção*, 11(1):109–119, 2004.

[37] Pednault, E. P. D. ADL: Exploring the middle ground between STRIPS and the situation calculus. *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning - KR'89*, Toronto, Canada, 1989.

[38] Penberthy, J. S. e Weld, D. S. UCPOP: A sound, complete, partial order planner for ADL. *Proceedings of the Third International Conference of Knowledge Representation and Reasoning - KR'92*, San Mateo, California, USA, 1992.

[39] Petri, C. A. Kommunikation mit automaten. *New York: Griffiss Air Force Base, Technical Report RADC-TR-65–377*, 1:1–Suppl. 1, 1966. English translation.

[40] Reisig, W. *Petri Nets: An Introduction*. Springer-Verlag, New York, 1985.

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

[42] Russell, S. e Norvig, P. *Artificial Intelligence: A Modern Approach*. Prentice Hall, New Jersey, 2nd edition, 2003.

[43] Silva, A. R. Y. da. Modelagem de custos em sistemas de manufatura utilizando redes de Petri. Dissertação de Mestrado, Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, SP, Brasil, 2002.

[44] Silva, F. Algoritmo para planificação baseada em STRIPS. Dissertação de Mestrado, Universidade Federal do Paraná - UFPR, Curitiba, PR, Brasil, outubro de 2000.

[45] Silva, F. *Rede de Planos: Uma proposta para a Solução de Problemas de Planejamento em Inteligência Artificial usando Redes de Petri*. Tese de Doutorado, Centro Federal de Educação Tecnológica do Paraná - CEFET, Curitiba, PR, Brasil, fevereiro de 2005.

[46] Silva, F., Castilho, M. A. e Künzle, L. A. Petriplan: a new algorithm for plan generation. *The International Joint Conference IBERAMIA'2000 - SBIA'2000*, Atibaia, SP, Brasil, 2000.

[47] Stehno, C. PEP version 2.0. *Tool demonstration ICATPN 2001*, Newcastle upon Tyne, UK, 2001.

[48] Stewart, I. A. On the reachability problem for some classes of Petri nets. Technical report, University of Newcastle upon Tyne, 1992.

[49] Töws, G. S. Petrigraph - um algoritmo para planejamento por desdobramento de redes de Petri. Dissertação de Mestrado, Universidade Federal do Paraná - UFPR, Curitiba, PR, Brasil, maio de 2008.

[50] Vossen, T., Ball, M., Lotem, A. e Nau, D. On the use of integer programming models in AI planning. *Proceedings of 16th the International Joint Conference on Artificial Intelligence - IJCAI'99*, páginas 304–309, 1999.

[51] Weld, D. S. Recent advances in AI planning. *AI Magazine*, 20(2):93–123, 1999.

[52] Yamada, M. C., Porto, A. J. V. e Inamasu, R. Y. Aplicação dos conceitos de modelagem e de redes de Petri na análise do processo produtivo da indústria sucroalcooleira. *Pesquisa Agropecuária Brasileira*, 37(6):809–820, 2002.