

LÚCIO MITIO SHIMADA

Volume 1 / 2



**ESTRUTURAÇÃO DO PROBLEMA DE
PLANEJAMENTO EM UMA ABORDAGEM
BASEADA EM IA E NO FORMALISMO DE REDES
DE PETRI**

Tese apresentada à Escola
Politécnica da Universidade
de São Paulo para obtenção
do título de Doutor em
Engenharia.

São Paulo

1997

LÚCIO MITIO SHIMADA

Volume 1 / 2

TESE

**ESTRUTURAÇÃO DO PROBLEMA DE
PLANEJAMENTO EM UMA ABORDAGEM
BASEADA EM IA E NO FORMALISMO DE REDES
DE PETRI**

Tese apresentada à Escola
Politécnica da Universidade
de São Paulo para obtenção
do título de Doutor em En-
genharia.

Área de Concentração:

Engenharia Mecânica

Orientador:

Prof. Dr. José Reinaldo Silva

São Paulo

1997

Universidade de São Paulo - Escola Politécnica (EPUSP)

Faculdade de Engenharia Mecânica

Laboratório de Automação de Sistemas (LAS) - *Eng. Mecatrônica*

Aluno: Lúcio Mitio Shimada N.USP: 1681108

Data: 02 de abril de 1997.

Assunto: Tese de Doutorado em
Engenharia Mecatrônica

Título: Estruturação do Problema de Planejamento em uma Abordagem
Baseada em IA e no Formalismo de Redes de Petri

Palavras-chaves: Redes de Petri, PFS / MFG, Ghenesys, HIPER, Inteligência Artificial, PROLOG, Planejamento e Sequenciação, Sistemas Dinâmicos, Roteador de Helicópteros, Planejamento Semi-Reativo, Problemas Combinatórios, Metodologia de Desenvolvimento de Sistemas, Estruturação, Modelo Conceitual.

AGRADECIMENTOS



Ao meu amigo Prof. Dr. José Reinaldo Silva, pela orientação deste trabalho. Ao Prof. Dr. Márcio Rillo, pela orientação nos estudos em AI Planning. Ao Prof. Dr. Paulo Eiji Miyagi, pelos estudos em PFS / MFG. Aos meus colegas da Mecatrônica, Prof. José Ricardo Sebastião, Prof. Diolino dos Santos Fo., Prof.a. Cristina Toshie Motohashi, Prof. Júlio Arakaki. À todos que direta ou indiretamente, contribuíram na execução deste trabalho. E, finalmente, à CNPq, FAPESP, FlexSys (Esprit 76101), AP3I e PETROBRÁS, especialmente ao Eng. Nelson Costa Cardoso, Chefe da Divisão Regional de Informática em São Paulo - DIRINF-SP, pelo patrocínio para a realização deste trabalho.

Aos meus pais,
Hiroshi e
Margarida Shimada.

SUMÁRIO

Lista de tabelas

Lista de figuras

Lista de gráficos

Resumo

Abstract

1 INTRODUÇÃO

- 1.1 Os Sistemas de “Planning” Baseados em IA
- 1.2 O “Planning” e a Automação de Fábrica
- 1.3 As Possíveis Abordagens de Solução
- 1.4 As Limitações da Hipótese do Mundo Fechado
- 1.5 O Papel da Estruturação em IA
- 1.6 A estruturação baseada em IA e Redes de Petri
- 1.7 A Reutilização de Conhecimento no "Planning"
- 1.8 O Conteúdo dos Capítulos

2 A ESTRUTURAÇÃO NOS SISTEMAS EXISTENTES EM IA

- 2.1 Sistemas Pioneiros
 - 2.1.1 GPS
 - 2.1.2 O Mundo de Blocos
 - 2.1.2.1 STRIPS
 - 2.1.2.2 Críticas ao STRIPS
 - 2.1.2.3 A Resolução Tradicional do Mundo de Blocos
- 2.2 O Planeamento Hierárquico
 - 2.2.1 O NOAH
 - 2.2.2 O SIPE
- 2.3 O Planeamento Dinâmico
 - 2.3.1 O Aprendizado e a Generalização
 - 2.3.2 O Replaneamento na Falha
 - 2.3.3 Planos Incompletos

2.4 O Aprendizado de Heurísticas de Busca no “Planning”

2.5 O Uso da Abstração Estrutural no “Planning”

2.6 O Uso de Redes de Petri em IA “Planning”

2.6.1 O Plano Representado como uma Rede de Petri

2.6.2 A Rede de Petri Aplicada ao Controle Inteligente

2.6.3 O Planejamento usando Redes de Petri

2.6.4 A Rede de Petri em Sistemas Emergenciais

3 O FORMALISMO DAS REDES DE PETRI

3.1 As Redes de Petri Elementares (RE)

3.1.1 As Relações entre as Transições

3.1.2 O Espaço de Estados dos Sistemas

3.2 A Rede de Petri “Lugar/Transição”

3.2.1 A Rede de Petri “Condição/Evento” (C/E)

3.3 As Propriedades da Rede de Petri

3.3.1 Os Invariantes de Lugar

3.3.2 Os Invariantes de Transição

3.3.3 A Rede Limitada

3.3.4 A Rede Cíclica

3.3.5 A Vivacidade

3.3.6 A Rede Reversível

3.4 A Análise das Propriedades da Rede de Petri

3.5 A rede de Petri Estendida Orientada a Objetos Ghenesys

3.5.1 A Super-Classe Box

3.5.2 A Super-Classe Atividade

3.5.3 As relações de Fluxo na rede Ghenesys

3.5.4 A Equação de Estado para a rede Ghenesys

3.5.4.1 O Disparo de uma transição

3.5.4.2 A Habilitação de uma transição

3.5.4.3 A Modularização de M para o box-capacidade

3.5.4.4 A Modularização de M para o box-de-tempo

3.5.5 O Ambiente Computacional HIPER

3.6 A Representação da RdP em Lógica

4 PROPOSTA DE UMA ABORDAGEM HÍBRIDA IA / REDES DE PETRI

4.1 O problema da Representação no “Planning”

4.1.1 O Cálculo Contextual

4.2 Crítica da representação STRIPS

4.3 Uma estrutura simples baseada em grafos

4.3.1 O Exemplo do Mundo de Blocos

4.3.2 O Construtor de Planos para o Mundo de Blocos

4.4 O Modelo Conceitual (mc)

4.5 A Descrição da Proposta

4.5.1 A Metodologia de Estruturação do Problema de “Planning”

4.6 O Uso da Análise de Propriedades da RdP

4.7 O Sistema Implementado para o Mundo de Blocos

4.7.1 Resultados Adicionais para o Mundo de Blocos

4.8 Uma Proposta de Arquitetura para o Sistema

4.8.1 O Módulo Planejador e Revisor do Plano

4.8.2 Porque a Reatividade é Importante

4.9 Diferença com outras abordagens

5 O ROTEADOR DE HELICÓPTEROS

5.1 As Componentes do Roteador de Helicópteros

5.2 O Tratamento de subredes pelo sistema

5.3 A Definição dos Níveis de Abstração para o Problema Corrente

5.4 O Roteador baseado em Rede de Petri

5.4.1 O Detalhador Baseado em Rede de Petri

5.4.2 Como Funciona o Roteador

5.5 O Otimizador baseado no PCV

5.5.1 Como Funciona o Otimizador

5.6 Resultados Computacionais

5.6.1 O Sistema de Referência

5.6.1.1 O Problema de Atribuição na PO

- 5.6.1.2 O Problema do Caixeiro Viajante na PO
- 5.6.2 O Custo e a Distância das Rotas Geradas
- 5.6.3 Os Custos Unitários das Rotas Geradas
- 5.6.4 Os Tempo de Atendimento das Rotas Geradas
- 5.6.5 O Comportamento Semi-Reativo
- 5.6.6 O Potencial de Ganho Econômico

6. CONCLUSÃO E TRABALHOS FUTUROS

- 6.1 O Uso de Modelos Conceituais
- 6.2 A Análise dos Resultados Obtidos
- 6.3 Outras Perspectivas de Fusão IA / Redes de Petri
 - 6.3.1 O Problema da Manutenção da Verdade
 - 6.3.2 A Evolução dos Estados Usando Rede de Petri
- 6.4 Trabalhos Futuros

REFERÊNCIAS BIBLIOGRÁFICAS

APÊNDICES

- Apêndice I - Planejamento e Sequenciação
- Apêndice II - Um “planning-schemata” usando Rede de Petri
- Apêndice III - Um Sistema de Planejamento Semi-Reativo de Sistemas Produtivos Dinâmicos usando Redes de Petri. Trabalho apresentado no 2º Simpósio Brasileiro de Automação Inteligente - SBAI. CEFET-PR. 13 a 15.set.1995. Curitiba.

LISTA DE TABELAS

- Tabela 1 Demandas do caso P0612AM
- Tabela 2 Casos de teste do Roteador
- Tabela 3 Características dos Helicópteros Cadastrados
- Tabela 4 Análise dos Resultados Nosso Roteador X Mainframe
- Tabela 5 A Comparação dos Custos Unitários das Rotas
- Tabela 6.a O Tempo de atendimento para problema teste n. 6 (para o nosso roteador)
- Tabela 6.b O Tempo de atendimento para problema teste n. 6 (para o mainframe)
- Tabela 7 Tempo de Atendimento dos Passageiros de Ida e de Volta

LISTA DE FIGURAS

- Fig. 1 O processo de "design" por síntese
- Fig. 2 Modelos de reutilização de "design"
- Fig. 3 Exemplo da abordagem meios-fins
- Fig. 4 O problema do Mundo de Blocos
- Fig. 5 A Anomalia de Sussman
- Fig. 6 Tipos de nós na rede procedimental NOAH
- Fig. 7 Exemplo da rede procedimental NOAH
- Fig. 8 A tabela triângulo PLAN-EX para armazenar um plano STRIPS
- Fig. 9 O problema STRIPS
- Fig. 10 A tabela triângulo PLAN-EX
- Fig. 11 A Representação do domínio do planejamento de robôs [Hayes 1975]
- Fig. 12 (a) A Árvore de submetas para o problema de se juntar duas caixas, na Fig. 12
.. (b) O Grafo de decisão do problema (na Figura 12)
- Fig. 13 O Ciclo de Vida de uma tarefa NASL
- Fig. 14 A Representação do Estado (322) da Torre de Hanói
- Fig. 15 O Grafo de Estados para o Problema da Torre de Hanói para três discos
- Fig. 16 Uma Solução Hierárquica para o Problema ((1,1,2), (3,1,2))
- Fig. 17 A Rede de Petri estendida para Mundo de Blocos

- Fig. 18 Os níveis de abstração do APE, frente ao evento fumaça na cozinha
- Fig. 19 A rede de Petri para o operador *reflexo_de_dor*
- Fig. 20 A rede de Petri para o operador *reflexo_de_dor_na_mão*
- Fig. 21 Exemplo de uma rede de Petri Elementar (RE)
- Fig. 22 A mudança-de-estados no disparo de uma transição
- Fig. 23 O Grafo de estados para a RE da Figura 21
- Fig. 24 A RdP P / T para representar uma relação produtor - consumidor
- Fig. 25 Hierarquia de classes na rede Ghenesys
- Fig. 26 Exemplo de rede Ghenesys
- Fig. 27 Editor gráfico do ambiente HIPER
- Fig. 28 Enumeração Completa dos Estados do Mundo de Blocos
- Fig. 29 “Application Model” usando RdP C/E para o Mundo de Blocos
- Fig. 30 “Planning model” usando rede Ghenesys para o Mundo de Blocos
- Fig. 31 O Construtor de Planos para o Mundo de Blocos
- Fig. 32 Estrutura do Problema de “Planning”
- Fig. 33 Implementação da RdP da Figura 30 em Petri Netze für Windows
- Fig. 34 O Menu de barras para o Mundo de Blocos
- Fig. 35 O Menu de Funções do Planejador para o Mundo de Blocos
- Fig. 36 Os Módulos do Sistema Planejador Semi-Reativo (PSR)
- Fig. 37 Os Componentes do Módulo Planejador / Revisor
- Fig. 38 O Sistema Roteador de Helicópteros
- Fig. 39 Os Componentes do Roteador de Helicópteros
- Fig. 40 A Localização de helipontos no mar
- Fig. 41 O tratamento de Subredes pelo Sistema
- Fig. 42 A Estruturação do Conhecimento do Domínio (domain-K)
- Fig. 43 Rede de Petri C/E para um “cluster” de plataformas situado no Nível 2
- Fig. 44 O Agrupamento de Helipontos e a Síntese do “Application Model”
- Fig. 45 O Roteador baseado em Rede de Petri
- Fig. 46 O Plano RdP para o Nível 2 de abstração
- Fig. 47 Os Tipos de Permutação de Rotas
- Fig. 48 A Rota 2 otimizada para o Nível 2
- Fig. 49 O Plano PCV para o Nível 0 (“ground”) de abstração

Fig. 50 Exemplo do Problema do Caixeiro Viajante (PCV)

Fig. 51 O Plano modificado para eliminar heliponto na rota 2

LISTA DE GRÁFICOS

Gráf. 1 Custo total das rotas (U\$)

Gráf. 2 Distância total das rotas (km)

Gráf. 3 Custo Unitário por Passageiro (U\$ / passag.)

Gráf. 4 Tempo Médio de Atendimento na IDA

Gráf. 5 Tempo Médio de Atendimenro na VOLTA

RESUMO

Os sistemas de Planejamento baseados em Inteligência Artificial (IA) utilizam bastante conhecimento implícito e incompleto, o que torna o desenvolvimento de sistemas para resolução deste tipo de problema muito complexo. Para se obter um melhor desempenho computacional nestes sistemas, é necessário se adotar mecanismos auxiliares para estruturar o problema. Neste trabalho, fazemos a estruturação do problema de Planejamento baseado em IA e no formalismo das Redes de Petri (RdP). A RdP promove a modelagem do problema como um Sistema de Eventos Discretos (SED) e pode capturar a característica principal de um sistema produtivo que é o seu comportamento dinâmico. A estruturação proposta permite a construção de um método para o desenvolvimento com qualidade de sistema de IA “planning”.

A nossa proposta de estruturação está baseada na construção de um *modelo conceitual (mc)* do problema, baseado no formalismo das Redes de Petri. O *mc* é constituído por dois modelos, que representam, separadamente, os tipos de conhecimento envolvidos no problema de “planning”: (i) o “application model”, que representa o conhecimento do domínio da aplicação, e (ii) o “planning self-model”, que representa o meta-conhecimento envolvido no processo de “planning”, ou seja, representa um plano para fazer planos. O *mc* é importante tanto para o processo de elaboração do plano inicial, assim como no caso ocorrer a necessidade de um replanejamento na falha, por exemplo, na execução de alguma tarefa do plano corrente.

Inicialmente, vamos mostrar, usando o exemplo do Mundo dos Blocos, como uma análise da estrutura da rede de Petri que representa o domínio do problema pode prover o meta-conhecimento necessário para dirigir o processo de “planning”. Através dos resultados desta análise da estrutura da rede que representa “application model”, podemos estabelecer uma estratégia de resolução do problema, i.e., aprender um meta-conhecimento. Utilizamos

a rede de Petri para elaborar o “planning self-model” que representa este meta-conhecimento.

As abordagem proposta apresenta diversas vantagens tais como: planos hierárquicos não lineares com diversos níveis de abstração, formalismo matemático, visualização gráfica, captura de todas as possíveis soluções, aprendizado de formas de decompor o problema e de estratégias de busca de soluções e capacidade de fazer o replanejamento na falha. Além disso, a abordagem apresenta a possibilidade de se implementar um planejador reativo puro e a possibilidade de se fazer a reutilização de planos.

Finalmente, a abordagem proposta não depende do domínio da aplicação, porque é a análise da estrutura das Redes de Petri, independente da sua interpretação para o domínio do problema, que permitem o aprendizado do meta-conhecimento para dirigir o processo de planejamento. O Construtor de Planos, é um ambiente de rede de Petri que opera sobre a rede que representa o conhecimento do domínio, e não sobre a sua interpretação para um determinado domínio da aplicação.

No capítulo 5, descrevemos uma aplicação para um sistema Roteador de Helicópteros. A motivação para o desenvolvimento desta aplicação foi para mostrar que a metodologia proposta pode tratar problemas de porte encontrados no mundo real.

ABSTRACT

Artificial Intelligence (AI) based Planning Systems relies on much implicit and incomplete knowledge, which compromise the already great complexity of the development of systems in order to search for solutions. In order to achieve a better computational performance for these systems, it is necessary to adopt other mechanisms to structure the problem. In this thesis, we've made a planning problem structuring based in AI and in the formalism of the Petri Nets (PNs). The PN makes the problem modeling as a Discrete Event System (DES) and is able to capture the productive system's main features and its dynamic behavior. The proposed structuring allows the construction of a development method for AI planning systems with quality.

In our propose, the problem structuring is based in the construction and use of an problem *conceptual model (cm)*, based in the formalism of the Petri Nets. The *cm* consists of two models, which represents, separately, the distinct knowledge types involved in a planning problem: (i) the *application model*, which represents the domain application knowledge, and (ii) the *planning self-model*, which represents the meta-knowledge involved in the planning process, i.e., it represents a plan to make plans. The *cm* is as important to the processes of elaboration of the initial plan, as in the case of occurrence of a necessity to make a replanning because of a fail, for instance, in the execution of some task in the current plan.

Initially, we showed, using the Blocks World model-problem, how a Petri net structure analysis provides the meta-knowledge necessary to guide the planning process. Through the results of this net structure analysis, of the PN which represents the "application model", we may establish a strategy for the problem solution process, i.e., we may learn a meta-knowledge. We've used the Petri Net in order to build the "planning self-model", which represents this meta-knowledge.

Abstract

The proposed approach presents several advantages such as: non linear hierarchical plans with several abstraction levels, mathematical formalism, graphical visualization, capture of all the plausible solutions, learning of ways to decompose the problem and of strategies for the search for solutions and the capability to do the replanning in case of the occurrence of a fail. Besides, this approach allows the implementation of a purely reactive planner and the possibility of doing a plan re-use.

Finally, this approach does not depend on the application domain, because is the Petri Net structure analysis, independently of its interpretation for the problem domain, which allows the learning of the meta-knowledge needed to lead the planning process. The Plan Constructor is a Petri Net Environment and operates over the net which represents the domain knowledge, instead of over its interpretation for a given application domain.

In the Chapter 5, we describe an application for a Helicopter Routing System. The motivation for the development of this application was to show that the proposed methodology can treat the large problems found in the real world.

CAPÍTULO 1

1 INTRODUÇÃO

Freqüentemente, a inteligência está associada à habilidade que os seres vivos têm de atuar num mundo dinâmico. As formas de vida superior, apresentam a capacidade de *prever* (antecipar) o futuro e de *elaborar planos de ação* para o atingimento de suas metas. Deste modo, o estudo de *ações* e de *planos* é fundamental no desenvolvimento de sistemas inteligentes que sejam capazes de lidar eficazmente com os problemas do mundo real. As formas de vida inferior apresentam (apenas) um mecanismo do tipo estímulo-resposta [Georgeff 1987].

No processo de planejamento, a decisão do planejador acerca de qual curso de ação tomar é escolhida a partir de um vasto repertório de possibilidades. Esta decisão, por sua vez pode influenciar os estados do mundo de maneiras diversas e complicadas. Tudo isto é realizado num mundo complexo e dinâmico em que o meio-ambiente está em mudança contínua.

1.1 Sistemas de “Planning”

Para se elaborar um sistema de “planning” baseado em IA, é necessário um modelo de mundo e de um modelo de ação para atuar neste modelo de mundo. A elaboração de um plano de ação eficaz (que obtenha uma solução para o problema) e eficiente (que esta solução seja obtida num tempo de processamento baixo), usando-se o computador, requer e está limitado pelos seguintes fatores:

- (a) modelo de mundo aderente à realidade,
- (b) forma da representação de eventos e ações que consiga sintetizar o domínio do problema,

- (c) no caso de existência de metas conflitantes, formas claras de como resolver estes conflitos.

O *modelo de mundo* pode ser aberto ou fechado. O modelo aberto trata do ambiente desestruturado onde o planejador não tem controle sobre todos os eventos que ocorrem no mundo. Por este motivo, este modelo é computacionalmente complexo. Para se reduzir o problema computacional associado ao modelo aberto, muitas vezes estabelece-se a hipótese do mundo fechado. Neste modelo, apenas as cláusulas cuja interpretação é conhecida podem ser verdadeiras. Quando é referenciada uma cláusula cuja interpretação não é conhecida, então, esta é imediatamente assumida como sendo falsa. Isto reduz muito o problema computacional, mas, em contrapartida, limita a aplicabilidade dos sistemas assim desenvolvidos.

A forma da representação dos eventos e ações é determinada pelo *modelo de ação*. Este define as formas possíveis para se atuar sobre o modelo de mundo. O modelo de mundo pode estar num dos seus (potencialmente) infinitos estados. Um estado é uma foto instantânea do mundo em determinado instante. Fazendo-se uma metáfora com a exibição de um filme no cinema, um estado do mundo seria um quadro isolado (ou fotograma). Um caso ("case") é conjunto de valores de propriedades de elementos fundamentais ("features") e que corresponde a um estado do modelo de mundo.

A *mudança de estados* no modelo de mundo se verifica a partir da ocorrência de um evento. A ocorrência de um *evento* pode ou não estar sob o controle do planejador. Uma *ação* é um tipo especial de evento efetuado pelo planejador de modo intencional. Por exemplo:

Queda das folhas de uma árvore (evento não controlável).

João corre três voltas no parque (ação intencional).

Uma seqüência de estados é chamado de *comportamento*. Uma seqüência de estados que inclui o estado inicial e o estado final é chamado de história. Um *plano* pode ser visto como sendo a especificação de um comportamento desejado sobre o modelo de

mundo para se atingir uma determinada meta (estado final), a partir de um estado inicial conhecido.

Os sistemas de planejamento baseados em Inteligência Artificial (IA) utilizam bastante conhecimento implícito e incompleto, o que torna o desenvolvimento de sistemas para resolução deste tipo de problema muito complexo. O problema de “planning” apresenta uma natureza não-linear e, geralmente, os espaços das possíveis soluções cresce combinatoriamente. Por este motivo, o problema de “planning” requer um tipo de raciocínio não monotônico para o processo de elaboração de soluções.

1.2 “Planning” e a Automação de Fábrica

Os Sistemas Integrados de Manufatura são uma realidade cada vez mais presente nas fábricas. Além da presença crescente da automação e de computadores de processo, observa-se a integração destes em redes. Muitas empresas promovem uma integração adicional destas redes de automação (para acionamento, controle, supervisão do sistema) com as redes administrativas da companhia. Desta forma, atualmente, tem-se a integração total de uma companhia através da automação e da informática. É importante lembrar que SIM não é um produto que se possa comprar no mercado, mas, sim um conceito para se promover a integração total de uma companhia através de uma arquitetura única.

As atividades de “planning” e de “scheduling” tem um papel essencial nesta arquitetura. Estas provêm a lógica para operacionalizar a integração das diversas camadas funcionais do SIM. O *planejamento* num sistema de manufatura discreta, consiste na definição das ações necessárias para se atingir determinados objetivos no contexto do sistema organizado. A função planejamento corresponde à camada do Sistema de Planejamento e Programação (SPP) de um Sistema Flexível de Fabricação (SFF) e estabelece as metas agregadas de produção para um determinado intervalo de tempo, por exemplo, uma semana.

Na prática, muitas vezes, estas soluções analíticas não são adequadas. Os problemas apresentados por esta abordagem são: tempo de processamento elevado, excessiva rigidez dos modelos, o fato de a solução fornecida ser pontual, e a limitada capacidade de análise de sensibilidade. Em outras palavras, pelo fato destes modelos fornecerem uma solução ótima, o tempo de processamento torna-se elevado e qualquer modificação na definição do problema requer a solução de um novo problema a partir do início -- porque a capacidade de análise de sensibilidade, levando-se em conta “perturbações” nos valores de variáveis do problema só vale para uma pequena região que está localizada na vizinhança da solução ótima corrente.

As técnicas usadas na Inteligência Artificial(IA) podem obter soluções boas, embora não necessariamente ótimas. A solução baseada em IA torna-se interessante porque:

- (i) pela natureza combinatória destes problemas, existem muitas soluções implementáveis (ou seja, soluções factíveis), muito próximas uma das outras, que não são ótimas, mas que são soluções aceitáveis,
- (ii) o tempo de processamento para se obter estas soluções “boas” pode ser muito menor comparado com o tempo para se atingir a solução matematicamente ótima, principalmente para problemas de grande porte,
- (iii) o ambiente em que o plano deve ser executado pode ser dinâmico, estando em mudança contínua,
- (iv) o planejador não quer abandonar o plano que vinha sendo executado antes da mudança do estado do mundo, sempre iniciando a confecção de um novo plano, mas sim adaptar reativamente o plano corrente.

Entretanto, existem críticas possíveis de serem feitas à esta abordagem. No capítulo 2, descrevemos os sistemas de planeamento baseados em IA. Apesar dos aspectos inovadores seguidamente propostos, como está descrito no item 2.4.1, os sistemas de planeamento baseados em IA, em sua maioria, ainda definem um plano como uma sequência de operadores do tipo STRIPS como nos sistemas clássicos (pré-1975). A questão está na falta de expressividade do operador STRIPS em contextualizar correta-

mente os efeitos da aplicação de uma determinada ação. Note que a maioria das contribuições apresentadas nos sistemas pioneiros ou clássicos -- proteção de metas já atingidas, críticas para detectar conflito, regressão de metas, hipótese do mundo fechado, ... -- estão relacionadas, de alguma forma, com formas de se corrigir este problema. Até mesmo, nos sistemas que integram planejamento e execução, são necessários os diferentes mecanismos propostos -- tabela triângulo, árvore de submetas e grafo de decisão, ... -- tratam da questão de se contextualizar os efeitos da aplicação de uma determinada ação definida no plano.

Na verdade, a origem de todos estes problemas está no escasso uso de modelos conceituais, necessários para apreender e representar as partes estática e dinâmica importantes do domínio do problema -- o modelamento funcional contextualizando o efeito da aplicação de ações que geram novos estados, é fundamental. Além disso, modelos conceituais mais completos auxiliam sobremaneira na derivação de estratégias (ou regras heurísticas) para guiar o processo tanto na busca de uma solução para o problema assim como no replanejamento em caso de falha, i.e., na derivação do "design rationale".

1.4 Limitações da Hipótese do Mundo Fechado

Apesar de serem bastante atraentes para tratar de problemas como o de planejamento, os sistemas de IA (baseados em conhecimento) apresentam algumas limitações básicas que precisam ser consideradas.

Segundo a Hipótese do Mundo Fechado, um Sistema Baseado em Conhecimentos (SBC), é constituído por uma Base de Conhecimento Inicial (BC_0) e um mecanismo de inferência (MI) -- que corresponde a um conjunto de axiomas válidos numa determinada linguagem de implementação (\mathcal{L}). Desta maneira, a Base de Conhecimentos (BC) do SBC é constituída não apenas pela BC_0 , mas sim pelo que pode ser deduzido a partir de BC_0 aplicando-se os axiomas válidos em \mathcal{L} , ou seja:

$$BC = \langle BC_0 \rangle \underset{\mathcal{L}}{\vDash}$$

Considere uma sentença lógica qualquer α . O SBC somente será capaz de determinar o valor verdade de α se:

- (a) α já estiver verdadeira na BC_0 ou,
- (b) α puder ser deduzida a partir de BC_0 e dos axiomas em \mathcal{L} .

Isto significa que um SBC não consegue deduzir o valor verdade de nenhuma sentença que já não esteja verdadeira na sua Base de Conhecimentos Inicial (BC_0) ou, então, cujo valor verdade não possa ser deduzida a partir de BC_0 e dos axiomas válidos na linguagem de implementação (\mathcal{L}), ou seja:

$$\alpha \in BC \Leftrightarrow BC_0 \underset{\mathcal{L}}{\vDash} \alpha$$

Chamamos a atenção para este fato porque, para se fazer o desenvolvimento de sistemas de planejamento que operam em ambientes dinâmicos, devemos ser capazes não só de representar as características estáticas do sistemas, mas, principalmente, as características dinâmicas dos sistemas alvo. Portanto, o tratamento de problemas de planejamento por técnicas de IA, segundo a Hipótese do Mundo Fechado apresenta boas perspectivas de resultados por um lado e algumas limitações por outro.

I.5 O papel da estruturação em IA

A literatura mostra que a elaboração de planos lineares simples construídos usando tão somente abordagens ingênuas baseado nos métodos fracos de IA (busca em largura, busca em profundidade), no estilo tentativa e erro, não logram sucesso. A resolução deste problema constitui um desafio permanente devido a sua complexidade (os algoritmos conhecidos para a sua solução são NP-completos). O que se busca é obter sistemas de planejamento mais eficazes (que consigam, efetivamente, atingir uma boa so-

lução para o problema) e mais eficientes (que esta boa solução seja obtida no tempo de resposta exigido pelo contexto da aplicação).

Nos primórdios da IA, adotou-se a teoria linear que assume que duas submetas podem ser obtidas de uma forma independente. Desta forma, uma conjunção de metas era dividida em submetas para as quais se tentava obter uma solução isoladamente. É claro que, esta hipótese não é verdadeira na maioria dos problemas de “planning”. [Sussman 1974] critica a abordagem de se tentar, nestes casos, aplicar mecanismos de busca do tipo tentativa e erro, por exemplo “backtracking”, para reverter o planejador até algum determinado ponto onde a próxima solução candidata mais plausível pudesse ser selecionada e verificada. Isto porque esta abordagem leva rapidamente à explosão combinatória. A próxima candidata mais plausível pode falhar exatamente do mesmo modo da candidata corrente e, por exemplo, apenas a milésima candidata mais plausível satisfazer a meta. Ao invés, o planejador deveria aprender com os seus erros e não gastar recursos computacionais, tentando alternativas pouco promissoras. Portanto, para se obter um melhor desempenho computacional nestes sistemas, é necessário se adotar mecanismos auxiliares para estruturar o problema.

Em aplicações do mundo real, não queremos resolver planos lineares, porque estes promovem uma ordenação total das ações no plano que, realmente, não são necessárias. As atividades, por exemplo, em aplicações de controle são altamente paralelas. No capítulo 2, descrevemos como os sistemas existentes de planejamento baseados em IA tratam da estruturação do problema, e como esta sempre foi buscada nestes sistemas.

A abordagem usando níveis hierárquicos de abstração foi proposta primeiramente por [Sacerdoti 1975] no sistema NOAH. O SIPE [Wilkins 1984], [Wilkins 1988] define o planejamento hierárquico como sendo uma eliminação de detalhes em múltiplas camadas da representação, onde cada nível de abstração tem seu próprio conjunto de predicados com seu grau específico de granularidade. Ao diminuir o grau de detalhamento, usando-se uma abstração maior, podemos representar uma região maior do espaço solução do problema em questão. O processo de planejamento, então, consiste em descer nesta hierarquia; a cada nível os detalhes faltantes são suplementados, fazendo

“backtracking”, se necessário, até os níveis de abstração mais altos, até que um plano completo seja encontrado no nível “ground”. Nota-se aí uma primeira tentativa real de estruturação do problema de planeamento.

[Mädler 1992] apresenta uma abordagem equivalente à técnica de abstração hierárquica em termos de estruturação do grafo de estados. Usando aprendizado indutivo é capaz de detectar determinados estados singulares, denominados de “needle’s eyes”. Através da análise da estrutura do grafo de estado, ele detecta a presença destes estados singulares que são utilizados, então, para estabelecer estratégias de resolução do problema. De fato, [Mädler 1992] faz uma decomposição do problema baseado numa abstração estrutural aplicada a grafos de estado.

Com o propósito de obter a estruturação do problema de “planning”, as Redes de Petri tem sido utilizadas em diversos sistemas. Como uma alternativa às redes procedimentais utilizadas no NOAH para representar planos não lineares, [Drummond 1985] usou a representação baseada numa rede de Petri C/E estendida. [Rillo 1988] desenvolveu um sistema inteligente aplicado para o controle de sistemas que utiliza uma representação baseada em diferentes tipos de redes de Petri, onde a identificação da regra mais aplicável num determinado momento é feita através de um jogador de marcas (“token player”). [Paiva 1993] descreve um sistema de planeamento aplicado para uma célula de montagem onde um plano hierárquico é elaborado como uma Rede de Petri, e as ações do plano são executadas diretamente pelo módulo de execução do plano. [Hendler 1992] apresenta uma arquitetura para o planeamento e monitoração de sistema dinâmicos que emprega redes de Petri para a representação do conhecimento, por exemplo, o comportamento reativo, que está localizado no nível mais baixo de abstração.

Neste trabalho, fazemos a estruturação do problema de planeamento baseada em IA e no formalismo das redes de Petri [Silva et Shimad 1995]. A estruturação proposta permite a construção de um método para desenvolvimento com qualidade de sistemas de planeamento.

1.6 A estruturação baseada em IA e Redes de Petri

Esta proposta está fundamentada na elaboração de um *modelo conceitual* (mc) baseado em Rede de Petri (RdP), como descreve o capítulo 4. A RdP promove a modelagem como Sistema de Eventos Discretos (SED) e pode capturar a característica principal de um sistema produtivo que é o seu comportamento dinâmico [Miyagi 1990].

A estruturação proposta faz uma separação clara dos tipos de conhecimento existentes num problema de “planning”: *o conhecimento do domínio da aplicação* e *o meta-conhecimento* para dirigir o processo de busca de soluções. Um *modelo conceitual* (mc) é construído usando-se a teoria das Redes de Petri para separar estes tipos de conhecimentos e para representar: (1) o problema de “planning” em si (o “planning self-model”), e (2) o domínio do problema (o “application model”). Estas redes de Petri constituem um “mapa rodoviário” ao alcance da mão do Planejador e descrevem todo o processo de “planning”.

O *mc* que representa o domínio do problema (o “application model”) não corresponde a nenhum plano em particular mas, sim, corresponde à combinação de todos os planos, factíveis ou não, que constituem soluções para o problema representado.

Através da análise das propriedades estruturais das Redes de Petri que representa o domínio da aplicação, podemos aprender estratégias de busca (IA) dirigir o processo de resolução do plano. Estas estratégias de busca de soluções, por sua vez, são também representadas através de uma outra rede de Petri, o “planning self-model”. O mc para o problema de “planning” pode ser visto como um representação *do meta-planejamento*, ou seja, plano para se construir o plano.

As vantagens da abordagem proposta são:

- (i) estruturação do problema em níveis hierárquicos de abstracção,
- (ii) formalismo baseado em álgebra vetorial,

- (iii) visualização gráfica do modelo conceitual,
- (iv) captura de todas as possíveis soluções do problema,
- (v) separação do conhecimento do domínio do meta-conhecimento de "planning",
- (vi) aprendizado através de análise estrutural:
 - de estratégias de busca da solução: através da análise estrutural da rede,
 - de maneiras de se fazer a decomposição do problema,
- (vii) possibilidade de validar o *mc* fazendo-se uma simulação das redes de Petri,
- (viii) aprendizado através do uso de modelos conceituais para se fazer o re-planejamento na falha,
- (ix) possibilidade de se implementar o planejador reativo,
- (x) possibilidade de reutilização de planos,

O problema do planejamento e da seqüenciação de tarefas é um problema de síntese, e pode ser visto como uma instância do problema mais geral de "design" (particularmente do CAD inteligente) e, por conseqüência, pode beneficiar-se da reutilização do "design rationale", tanto no processo de resolução de novos sistemas, bem como na modificação de um projeto existente. A questão da reutilização do "design" é discutida no item 1.7, à seguir.

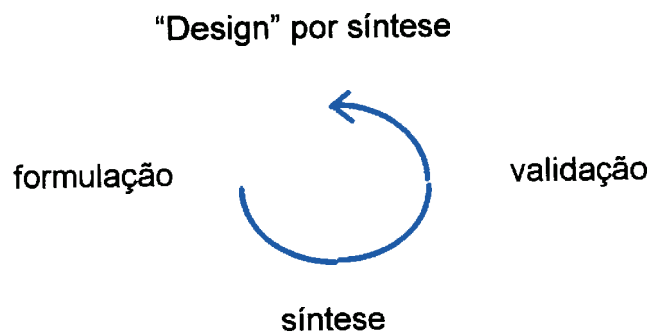


Figura 1 - O processo de "design" por síntese

1.7 A Reutilização de Conhecimento no "Planning"

"Planning" pode ser considerado uma instância do problema mais geral de "design". O problema de "design" consiste em se obter um modelo do problema, uma especificação rigorosa das metas e experiência anterior na resolução deste tipo de problema [Silva 1992], i.e.,

$$\text{DESIGN} = \text{MODELO} + \text{ESPECIFICAÇÃO} + \text{EXPERIÊNCIA.}$$

Em um modelo cognitivo proposto por [Maher 1990], o processo de "design" pode ser dividido em três subproblemas: formulação do problema, síntese da solução e validação da solução. O processo de "design" geralmente ocorre em ciclos circunscritos, chamado "design por síntese" onde cada ciclo é constituído por uma formulação seguida da sua validação, como ilustra a Figura 1, acima.

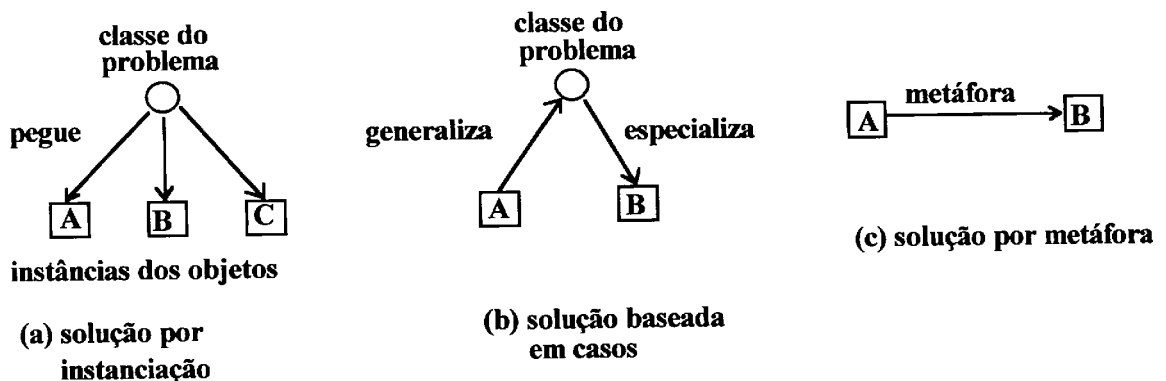


Figura 2 Modelos de reutilização de "design"

O "design" é um problema complexo caracterizado pela sua baixa produtividade. Por este motivo, é desejável se automatizar tarefas repetitivas deste processo. Todo processo de tomada de decisões no processo de "design" envolve um "design rationale" (ou seja, o que motiva a escolha de decisões). A reutilização de conhecimento no "design", envolvendo principalmente o "design rationale" pode ocorrer segundo três modelos [Silva 1992], ilustrados na Figura 2, acima.

O primeiro modelo -- solução por instanciação -- mostrado na Figura 3 (a), corresponde à aplicação de uma "solução enlatada" onde uma instância de objeto que está

em um Banco de Dados de Casos corresponde exatamente ao problema corrente. O segundo modelo -- solução baseada em casos -- mostrado na Figura 3 (b), corresponde ao caso em que o problema corrente deve ser generalizado para que seja associado à uma classe conhecida de problemas. A solução conhecida para os problemas desta classe deve então ser especializada para se adequar ao problema corrente. O terceiro modelo -- solução por metáforas -- mostrado na Figura 3 (c), foi proposto por [Silva 1992] e corresponde à aplicação do "design rationale" usado normalmente num domínio diferente do domínio do problema corrente para a sua solução.

Desta forma, a nossa proposta de estruturação do problema de "planning" pode ser classificada como uma "solução por metáforas". Os sistemas de planeamento, em geral, operam sobre o problema como foi especificado no domínio de aplicação. Ao invés, em nossa proposta, é a análise da estrutura das redes de Petri, independente da sua interpretação para o domínio do problema, o que permite o aprendizado de estratégias de busca para dirigir o processo de planeamento. Além disso, o processo de resolução opera sobre a representação do problema em redes de Petri (ou seja, sobre um esquema), e não sobre a sua interpretação para um determinado domínio da aplicação. Por este motivo, a nossa metodologia pode ser reaplicada para um outro domínio qualquer de problema.

A partir da estruturação proposta, podemos construir um método para desenvolvimento de sistemas de planeamento baseados em IA. Os métodos de desenvolvimento são importantes para a informática para assegurar a qualidade dos produtos. Ao padronizar as etapas do ciclo de vida do sistema (porque um sistema baseado em conhecimento, essencialmente, é um programa como qualquer outro), estamos tornando o processo de desenvolvimento mais científico, ou seja, obtém-se mais precisão e menos artesanato.

1.8 O Conteúdo dos Capítulos

Este trabalho está dividido em seis capítulos e possui três apêndices. No capítulo 1, apresentamos o problema de "planning" e a motivação para fazer este trabalho, desta-

cando a necessidade de estruturação do problema para se obter sistemas de planejamento mais eficazes (que consigam, efetivamente, atingir uma boa solução para o problema) e mais eficientes (que esta boa solução seja obtida no tempo de resposta exigido pelo problema).

Em nossa proposta de estruturação do problema de “planning”, utilizamos o formalismo das redes de Petri associado com regras heurísticas em IA. No capítulo 2, descrevemos como os sistemas existentes de planejamento baseados em IA tratam da estruturação do problema, e como esta sempre foi buscada nestes sistemas. Descrevemos também, no capítulo 2, item 2.6, outros sistemas que utilizaram as redes de Petri para promover a estruturação em diferentes níveis do problema de “planning”.

No Capítulo 3, descrevemos o formalismo da redes de Petri; os tipos de redes de Petri, as propriedades estruturais e as propriedades comportamentais, a análise de propriedades e as redes estendidas. A nossa proposta de estruturação, está baseada na utilização do ambiente computacional HIPER (“A High Level Integrated Petri Net Environment for the Design of FMS”) [Silva et all 1996]. O ambiente HIPER promove o modelamento e simulação de uma rede de Petri estendida Orientada a Objetos Ghenesys (“Graphical Hierarchical Extended Petri Net System”). O Ghenesys é baseado na metodologia PFS / MFG [Miyagi 1988]. A rede de Petri Ghenesys está descrita no item 3.5.

No Capítulo 4, apresentamos a nossa proposta de estruturação do problema de “planning” baseada em Inteligência Artificial (IA) e no formalismo das redes de Petri. A abordagem consiste na construção de Sistemas Baseados em Conhecimento (SBCs) utilizando o modelamento em redes de Petri do: (i) domínio do problema (o “application model”), e (ii) do problema de “planning” em si (o “planning self model”). Estas redes de Petri constituem o *modelo conceitual* (*mc*) do problema. Enquanto o “application model” descreve aspectos “salientes” do domínio para o “planning”, o “planning self-model” representa o meta-conhecimento, i.e., as estratégias, para se fazer a busca das soluções para o problema. Desta forma, com este modelo conceitual, o Planejador tem ao alcance da mão um “mapa rodoviário” que descreve todo o processo de “planning”.

Ilustramos o processo de elaboração e o uso deste *mc* para o Problema do Mundo de Blocos, para três blocos.

No capítulo 5, descrevemos uma aplicação feita com a metodologia proposta para um problema real, o sistema Roteador de Helicópteros. O Roteador de Helicópteros consiste de um sistema para elaborar a programação de vôos de helicópteros entre uma base localizada em terra (“on-shore”) e plataformas de petróleo localizadas no mar (“off-shore”). Este sistema Roteador produz planos hierárquicos, não-lineares e apresenta um comportamento semi-reativo. A partir da análise estrutural das redes de Petri, e com o uso de um mecanismo de inferência que consiste de um ambiente de Rede de Petri que opera sobre a rede que representa o conhecimento do domínio, o sistema Roteador é um SBC que é capaz de elaborar todas as rotas simultaneamente. O processo de construção das rotas leva em conta a heurística do “menor custo de inserção”, considerando-se ainda as limitações relacionadas com cada um dos aparelhos. O uso do *mc* permite o re-planejamento em caso de ocorrer uma falha na execução de uma determinada tarefa.

Finalmente, no capítulo 6, apresentamos as conclusões e os trabalhos futuros. No Apêndice I, descrevemos uma aplicação que fizemos da nossa proposta de estruturação para o problema do Scheduling Job Shop. No Apêndice II, descrevemos como a proposta pode ser aplicada como sendo um “planning schemata” para a resolução de problemas combinatórios. Ilustramos os resultados obtidos de programas que elaboramos para diversos problemas exemplo deste tipo: o problema das jarras, o problema do fazendeiro e o problema dos missionários e canibais. Além disso, descrevemos como a análise estrutural da rede de Petri para o problema do Mundo de Blocos pode ser generalizada para se resolver o problema para qualquer número de blocos. Elaboramos um programa baseado nesta generalização e mostramos os resultados obtidos para o problema do Mundo de Blocos com dez blocos.

CAPÍTULO 2

2 A ESTRUTURAÇÃO NOS SISTEMAS EXISTENTES EM IA

Neste capítulo, vamos descrever como os sistemas existentes de planejamento baseados em IA tratam da estruturação do problema, e como esta sempre foi buscada nestes sistemas. Fazemos uma comparação dos diversos mecanismos propostos para estes sistemas com a nossa proposta de estruturação baseada em IA e no formalismo de Redes de Petri. Os sistemas de planejamento baseados em IA podem ser divididos em duas fases:

- os sistemas pioneiros ou sistemas clássicos (pré-1977) e
- sistemas integrados de planejamento e execução (pós-1977).

Os sistemas pioneiros ou sistemas clássicos de planejamento baseados em Inteligência Artificial baseiam-se nas teorias da psicologia cognitiva, da pesquisa operacional e da lógica proposicional. A psicologia cognitiva está relacionada com o estudo do comportamento dos seres humanos na resolução de problemas. A pesquisa operacional está relacionada com os métodos de busca no espaço das soluções possíveis. A lógica proposicional está relacionada com a representação do conhecimento e com os mecanismos de dedução como, por exemplo, os provadores de teoremas. Os sistemas clássicos abordam problemas simples usando técnicas como provadores de teorema e sistemas especialistas. Estes implementaram idéias importantes como: planejamento hierárquico, crítica de planos, estratégia “meios-fins” e regressão de metas. Os sistemas pioneiros estão descritos nos itens 2.1 e 2.2, à seguir.

Os sistemas de planejamento pós-1975 são integrados à execução e, portanto mais próximos dos sistemas reais cujo comportamento pretendem modelar, antecipar e controlar. Os sistemas clássicos de planejamento em IA (sendo a maioria pré-1975) definem o plano como uma seqüência de operadores do tipo STRIPS. Esta seqüência de operadores representa ações que devem ser efetuadas para se transformar um determinado esta-

do inicial no estado meta desejado. Em tais sistemas a elaboração do plano não leva em conta o sucesso ou falha na execução do plano.

Para tornar os sistemas mais próximos do mundo real, a maioria dos sistemas pós-1975, trabalha com planos hierárquicos parcialmente elaborados, que definem metas parciais ou intermediárias. Uma determinada meta parcial ou intermediária é modificada para outra usando-se qualquer transformação de planejamento aplicável, tais como: a expansão de uma ação num nível de maior detalhe, a inclusão de uma ordem parcial entre ações para resolver alguma interação de dependência entre ações paralelas, etc. Os sistemas pós-1977 estão descritos no item 2.3. Estes sistemas de planejamento passam a levar em conta aspectos importantes como o replanejamento na falha, planos parciais, aprendizado e reutilização. Mais recentemente, motivado principalmente pelas necessidades do domínio da robótica, surgem os sistemas reativos em tempo real.

2.1 Sistemas Pioneiros

Os primeiros sistemas de planejamento em IA baseiam-se nas teorias da psicologia cognitiva, da pesquisa operacional e da lógica matemática. Trata-se de sistemas simples de IA (de primeira geração), não distribuídos, que tratam geralmente de uma simplificação do problema a ser resolvido. Somente mais tarde (1970) surgiu a idéia do plano como uma forma de "obter uma formulação detalhada de um programa de ação". Os sistemas mais significativos dentre os sistemas clássicos de planejamento em IA são o General Problem Solver - GPS e o STRIPS.

2.1.1 GPS

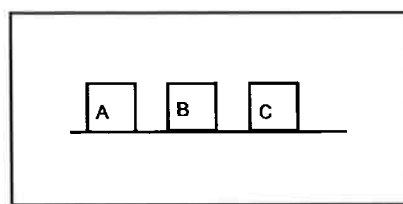
O Resolvedor Geral de Problemas ou "The General Problem Solver"- GPS [Newell et Simon 1963] baseia-se na psicologia cognitiva, na pesquisa operacional e na lógica matemática. A psicologia cognitiva contribuiu para se fazer a análise do processo de resolução de problemas por especialistas humanos através da técnica de análise de protoco-

los. A análise de protocolos foi usada na validação do sistema, comparando-se o "trace" do sistema com o comportamento de um especialista humano. GPS é um provador de teoremas baseado em lógica proposicional. GPS aplica operadores sobre objetos visando o atingimento de metas. Operadores são fórmulas bem formadas (fbf) do cálculo proposicional.

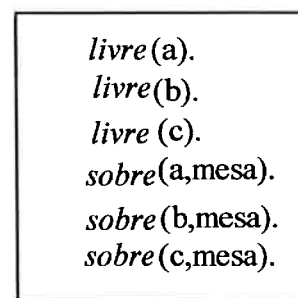
A principal contribuição do GPS está na abordagem meios-fins (ou "means-end"), que constitui uma técnica para dirigir o processo de busca de soluções. Basicamente, o "means-end" é uma forma de atingir a meta estabelecendo submetas, cujo atingimento leva ao atingimento da meta inicial. No GPS, as sentenças lógicas são chamados de *objetos* -- objetos podem ser comparados com os estados do mundo. O método meios-fins apresenta três etapas, que são resolvidas iterativamente até que a diferença δ para que a meta (o_2) a ser atingida seja nula. O algoritmo é, basicamente, o seguinte:

- 1) transformar o objeto o_1 no objeto o_2
- 2) detectar a diferença δ entre o_1 e o_2
- 3) escolher o operador ρ para ser aplicado ao estado o_1 , verificando sua relevância para reduzir esta diferença δ

Vamos ilustrar esta estratégia usando a Figura 3 (Mundo de Blocos), a seguir:



(a) Situação dos blocos que foi representada



(b) Predicados na Base de Conhecimentos

Figura 3 Exemplo da abordagem meios-fins

Na Fig. 3, deseja-se empilhar os blocos, dado que estes estão inicialmente desempilhados (o_1) e apenas colocados sobre a mesa. Suponha que se queira atingir o estado

meta (o_2) em que o bloco a está sobre o bloco b e o bloco b está sobre o bloco c . Utilizando as cláusulas $sobre(X,Y)$ e $livre(X)$, podemos representar o estado meta como sendo $(sobre(a,b) \wedge sobre(b,c) \wedge sobre(c,mesa) \wedge livre(a))$. Supondo que no estado inicial todos os blocos estão sobre a mesa e que todos os blocos estão livres, i.e., não existe nenhum outro bloco sobre eles. O estado inicial pode ser representado por $(sobre(a,mesa) \wedge sobre(b,mesa) \wedge sobre(c,mesa) \wedge livre(a) \wedge livre(b) \wedge livre(c))$.

Para modificar o estado do sistema, definem-se dois operadores:

- $mover(X,mesa,Y)$, para mover o bloco X que está sobre a mesa e colocá-lo sobre o bloco Y . A pré-condição para se aplicar este operador é de que ambos os blocos X e Y estejam livres.
- $mover(X,Y,mesa)$, para mover o bloco X que está sobre o bloco Y e colocá-lo sobre a mesa. A pré-condição para se aplicar este operador é de que o bloco X esteja livre.

Aplicando a estratégia meios-fins, tem-se para os três tipos de metas:

- 1) transformar $o_1: (sobre(a,mesa) \wedge sobre(b,mesa) \wedge sobre(c,mesa) \wedge livre(a) \wedge livre(b) \wedge livre(c))$ em $o_2: (sobre(a,b) \wedge sobre(b,c) \wedge sobre(c,mesa) \wedge livre(a))$
- 2) a diferença o_2/o_1 é $(sobre(a,b) \wedge sobre(b,c))$ que é falsa.
 $(sobre(c,mesa) \wedge livre(a))$ já são verdadeiras.
- 3) o operador $mover(a,mesa,b)$ pode ser aplicado pois $livre(a)$ e $livre(b)$ são verdadeiras e sua aplicação pode reduzir a diferença detectada no passo 2. Similarmente, o operador $mover(b,mesa,c)$ pode ser aplicado reduzindo a diferença para o estado meta.

O GPS, por ser um sistema "flat knowledge", baseado apenas na análise de protocolos de especialista humanos, não logrou sucesso na prova de teoremas genéricos do tipo:

$$(r \rightarrow \neg p) \wedge (\neg r \rightarrow q) \models \neg(\neg q \wedge p)$$

Obs: este teorema afirma que: se uma cláusula r é verdadeira e implica que não é o caso que a cláusula p seja verdadeira e se a cláusula r é falsa implica que a cláusula q é

verdadeira, então, é teorema que não é o caso que a cláusula q é falsa e a cláusula p é verdadeira. Atualmente, existem diversos sistemas provadores de teorema "deep knowledge", baseados na lógica de primeira ordem que resolvem este tipo de problema sem dificuldades.

2.1.2 O Mundo de Blocos

[Fikes et Nilsson 1971] propuseram inicialmente o problema de se elaborar um plano para que um robô empurre três blocos de modo a juntá-los. [Sussman 1974] propôs uma variante deste problema que consiste em se empilhar blocos (que ficou conhecido como o problema do Mundo de Blocos). Este problema é importante porque, apesar da sua aparente simplicidade, apresenta todas as características que dificultam a automação dos problemas de planejamento. A Figura 4, a seguir, mostra o problema do Mundo de Blocos.

META: ATINJA ($sobre(a,b) \wedge sobre(b,c) \wedge sobre(c,mesa) \wedge livre(a)$)

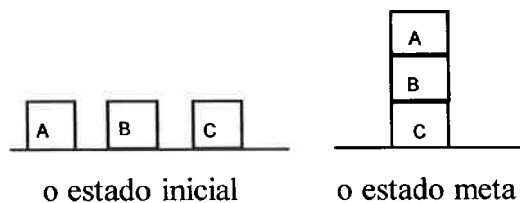


Figura 4 - O problema do Mundo de Blocos

2.1.2.1 O STRIPS

O STRIPS - STanford Research Institute Problem Solver [Fikes et Nilsson 1971] foi o primeiro sistema de planejamento em IA bem sucedido. O STRIPS tenta encontrar uma seqüência de operadores no espaço de modelos do mundo (onde o mundo é o domínio do robô de pesquisa do Stanford Research Institute -SRI), para transformar um dado modelo do mundo inicial num modelo em que uma determinada meta (fórmula bem formada) possa ser provada como sendo verdadeira. O STRIPS representa um modelo

do mundo como uma coleção arbitrária de fórmulas do cálculo de predicados de primeira ordem, empregando um provador de teoremas, que emprega o método da resolução, para responder a questões de determinados modelos e usa a análise meios-fins empregado no GPS para guiá-lo até o modelo do mundo desejado que satisfaça à meta. O STRIPS pode ser considerado como um sistema especialista (SE) no domínio do robô SRI, no sentido de ser um resolvidor genérico de problemas dentro deste domínio.

Uma questão importante nos sistemas de planejamento baseado em IA está na representação do conhecimento, destacando-se a representação do modelo do mundo e a representação do modelo de ação. O tipo de representação do conhecimento utilizado no STRIPS (em que os modelos do mundo estão representados pelos estados do problema e a ação é representada através de operadores e, através dos quais se modificam os modelos do mundo), tem sido utilizado em muitos sistemas de planejamento em IA. O STRIPS é caracterizado por uma base de fatos, um conjunto de operadores e uma meta expressa através de uma fórmula bem formada (fbf).

A evolução dos estados é determinada pelos operadores que têm como base:

- um conjunto pré-condições
- uma lista de adição
- uma lista de eliminação.

Busca-se atingir o modelo do mundo desejado (meta ou "goal") aplicando-se sucessivamente um conjunto de operadores STRIPS a partir do modelo do mundo inicial. O operador STRIPS pode ser aplicado ao estado corrente se todas as fórmulas no seu conjunto de pré-condições forem todas verdadeiras. O efeito da aplicação de um operador está definido através de listas de adição e listas de eliminação. A lista de adição define cláusulas que poderiam não ser verdadeiras no modelo original mas que são verdadeiras no novo modelo que resulta após a aplicação do operador. A lista de eliminação especifica cláusulas do modelo original que não são mais verdadeiras no novo modelo*¹.

* ¹ o valor verdade aqui está relacionado com a Hipótese do Mundo Fechado, onde a inexistência de um determinado fato é equivalente à atribuir-lhe o valor falso.

Como exemplo de um operador STRIPS, observe o operador *empurre*(k, m, n) que representa a ação do robô empurrar um objeto k a partir de um lugar m até um novo lugar n , como mostrado a seguir:

operador:	<i>empurre</i> (k, m, n)
pré-condição:	<i>robo_esta_em</i> (m) <i>esta_em</i> (k, m)
lista de eliminação:	<i>robo_esta_em</i> (m) <i>esta_em</i> (k, m)
lista de adição:	<i>robo_esta_em</i> (n) <i>esta_em</i> (k, n)

2.1.2.2 Críticas ao STRIPS

O problema do Mundo de Blocos tem sido resolvido tradicionalmente utilizando a representação STRIPS, descrita acima. Entretanto, esta representação apresenta fraquezas importantes: (1) [Lifschitz 1987] mostrou que o STRIPS enquanto um tipo de modelo lógico é inconsistente se suas regras forem aplicadas de forma arbitrária (o STRIPS pode ser visto como uma forma de modelo lógico, onde as regras STRIPS são axiomas) ; (2) [Sussman 1974] mostrou que o STRIPS não leva em conta a dependência entre metas concorrentes.

Nos primórdios da IA, adotou-se a teoria linear que assume que quaisquer duas submetas de um problema podem sempre ser obtidas de uma forma independente. Desta forma, uma conjunção de metas era dividida em submetas para as quais se tentava obter uma solução isoladamente. É claro que, esta hipótese não é verdadeira para a maioria dos problemas de “planning”. [Sussman 1974] critica a abordagem de se tentar, nestes casos, aplicar mecanismos de busca do tipo tentativa e erro, por exemplo backtracking, para reverter o planejador até algum determinado estado onde a próxima candidata mais plausível pudesse ser selecionada e verificada. Esta abordagem leva rapidamente à explosão combinatória. A próxima candidata mais plausível pode falhar exatamente do mesmo modo da candidata corrente e, eventualmente, apenas a milésima candidata mais plausível satisfazer a meta. Portanto, o planejador deveria aprender com os seus erros e não gastar recursos computacionais, tentando alternativas pouco promissoras.

A Anomalia de Sussman:

A Anomalia de Sussman está ilustrada na Figura 5, à seguir. A Anomalia do tipo “prerequisite-clobbers-brother-goal” (PCBG) ocorre quando, dada uma conjunção de metas e utilizando-se a abordagem “meios-fins”, o efeito da aplicação de uma ação para satisfazer uma submeta que foi estabelecida para satisfazer alguma meta, anula outra meta anteriormente já verdadeira. O PCBG é uma forma muito comum de não-linearidade.

Na Figura 5, seguinte, o estado meta é $(sobre(a,b) \wedge sobre(b,c) \wedge sobre(c,mesa) \wedge livre(a))$. No estado inicial todos os blocos estão sobre a mesa e todos os blocos estão livres, podendo ser representado por $(sobre(a,mesa) \wedge sobre(b,mesa) \wedge sobre(c,mesa) \wedge livre(a) \wedge livre(b) \wedge livre(c))$.

META: ATINJA $(sobre(a,b) \wedge sobre(b,c) \wedge sobre(c,mesa) \wedge livre(a))$

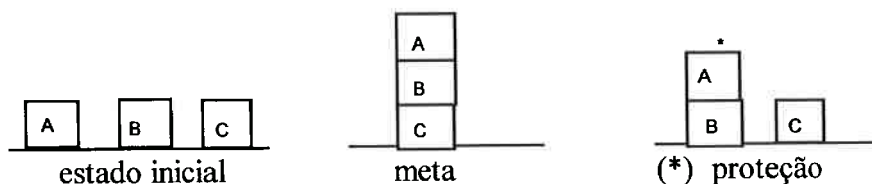


Figura 5 - A Anomalia de Sussman

Usando a abordagem meios-fins, conclui-se que o operador $mover(a,mesa,b)$ pode ser aplicado pois as pré-condições $livre(a)$ e $livre(b)$ são verdadeiras no estado inicial e sua aplicação pode reduzir a diferença entre o estado meta e o estado corrente, resultando no estado $(sobre(a,b) \wedge sobre(b,mesa) \wedge sobre(c,mesa) \wedge livre(a) \wedge livre(c))$. Porém, quando procuramos um operar para prosseguir a partir deste estado intermediário detectamos a diferença $sobre(b,c)$. Usando a abordagem meios-fins, concluímos que o operador $mover(b,mesa,c)$ reduz esta diferença. Mas, este não pode ser aplicado porque uma de suas pré-condições não é verdadeira ($livre(b)$ é falsa). Para se obter a satisfação desta pré-condição é necessário se estabelecer uma submeta $livre(b)$. Usando ainda a abordagem meios-fins, conclui-se que o operador $mover(a,b,mesa)$ é o operador rele-

vante para se estabelecer o valor verdade requerido para *livre(b)*. Paradoxalmente, este operador tem um efeito indesejado de aumentar novamente a diferença entre o estado corrente e o estado meta, indicando que o programa pode ficar num processo de "looping" sem apresentar qualquer progresso em direção ao estado meta desejado, ou que pelo menos este progresso nem sempre diminui monotonicamente a diferença até o estado meta, a partir do estado corrente (o_2/o_1).

Este fenômeno, acontece porque no modelo de ação do STRIPS, o efeito da aplicação de cada ação não está bem contextualizado. A avaliação do efeito de ações distintas no seu efeito são erroneamente avaliadas como equivalentes. A Base de Conhecimentos não tem nenhum tipo de estruturação, todos seus operadores apresentam igual oportunidade de serem escolhidos na hora de se decidir qual operador será aplicado em seguida. A abordagem meios-fins (ou "means-end"), criada no GPS, é o único critério empregado para se discriminar os operadores. Se o operador for considerado relevante para o atingimento do estado meta e se todas as suas pré-condições forem verdadeiras na BC, então, o operador é aplicado logo em seguida.

[Sussman 1974] propôs um mecanismo de "proteção" de metas já atingidas para se detectar a existência dessas situações de conflito. Na Figura 5, a meta a proteger seria *sobre(a,b)* que já é verdadeira, indicada por (*).

A resolução da Anomalia de Sussman é uma questão muito importante para se obter sistemas de planejamento baseados em IA que sejam eficientes e eficazes na busca de soluções. Mais adiante, no cap. 4, vamos discutir o papel da estruturação no desenvolvimento de sistemas para planejamento em IA e descrever proposta para a solução deste tipo de problema que está baseada no uso de redes de Petri. Vamos descrever no item 2.1.2.3, à seguir, a resolução tradicional para o problema do Mundo de Blocos. Mais tarde apresentaremos uma proposta de estruturação do problema baseado em redes de Petri.

Comparando com a nossa proposta de representação baseada em Rede de Petri, podemos observar que, na representação do operador STRIPS, nada é dito a respeito de pós-condições, para verificar se um dado operador pode ou não ser aplicado. Seja o exemplo do operador *empurre*(k,m,n), acima. Agora, suponha a ação *empurre*($a,sala1,sala2$) de o robô empurrar o bloco a (a partir) da *sala1* até *sala2*. A aplicação do operador é feita simplesmente se suas pré-condições forem verdadeiras, i.e., *robo_esta_em*(*sala1*) e *esta_em*($a,sala1$). Na representação em Rede de Petri, é sempre necessária a verificação simultânea das pré-condições e das pós-condições *². Um exemplo de pós-condição para a ação *empurre*($a,sala1,sala2$) seria, supondo que o robô é único, verificar se existe espaço suficiente na *sala2* para o bloco a , ou seja, não existe nenhum outro bloco ocupando a *sala2*, i.e., $\neg esta_em(X,sala2)$ (ou, em termos genéricos, $\neg esta_em(x,n)$).

2.1.2.3 A Resolução Tradicional do Mundo de Blocos


Para a solução da inconsistência do STRIPS (apresentada na seção anterior), [Lifschitz 1987] propôs adotar a hipótese do mundo fechado onde apenas um conjunto específico de fórmulas é permitido. Um operador STRIPS é definido pela 3-upla (P,D,A), onde P é um conjunto de sentenças que definem a pré-condição, D é um conjunto de sentenças na lista de eliminação e A é um conjunto de sentenças na lista de adição. Um operador descrito por (P,D,A) é consistente (“sound”) com relação a uma ação a , se para cada estado do mundo s , cada sentença da pré-condição P é verdadeira no estado s , e

- a é aplicável no estado s
- cada sentença na lista de adição A é verdadeira no estado resultante $a(s)$.
- cada sentença que é verdadeira no estado inicial s e não pertencente à lista de eliminação D continuará sendo satisfeita no estado resultante $a(s)$.

*² condição de disparo de transição estrita.

Um sistema STRIPS Σ é consistente se o modelo inicial do mundo é consistente e a descrição de cada operador (P_j, D_j, A_j) é consistente com relação à ação a_j .

Por exemplo, suponha que o operador *empurre*($a, sala1, sala2$), descrito anteriormente, é aplicado para representar que o robô empurra um bloco a que está na $sala1$ para $sala2$. As fórmulas *robo_esta_em*($sala2$) e *esta_em*($a, sala2$) na lista de adição do operador são verdadeiras após a aplicação deste. As fórmulas *robo_esta_em*($sala1$) e *esta_em*($a, sala1$) na lista de eliminação do operador não serão mais verdadeiras após a aplicação deste. Por outro lado, o fato de a ser um bloco expresso por *bloco*(a), por exemplo, não é alterado pelo fato dele ser empurrado da $sala1$ para a $sala2$. Portanto, este fato não precisa ser incluído na lista de eliminação do operador.



 META: ATINJA $(sobre(a, b) \wedge sobre(b, c) \wedge sobre(c, mesa) \wedge livre(a))$

Para a solução da dependência entre metas concorrentes, [Waldinger, 1975] propôs o método de regressão de metas que se tornou um método clássico para a solução de problemas com metas concorrentes, como ilustramos acima.

- apenas uma meta é tratada por vez.
- se uma meta mais recente interferir com o plano existente para metas já resolvidas, a meta conflitante é "regredida" através do plano, i.e., para uma posição anterior onde não exista mais interferência. Em outras palavras, a meta conflitante *sobre*(b, c) deve ser obtida antes da meta anteriormente "já resolvida" que foi "protegida", *sobre*(a, b).

Observe que estas soluções tradicionais (hipótese do mundo fechado, regressão de metas), estão relacionadas com a idéia de planos lineares construídos incrementalmente usando estratégias que diminuem monoticamente a diferença do estado corrente em relação ao estado meta. Observe que, enquanto que a hipótese do mundo fechado contorna o problema da inconsistência, a regressão de metas consiste simplesmente de um re-

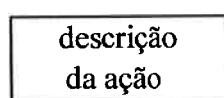
ordenamento da conjunção de metas (no estado meta) quando ocorre uma falha no planeamento usando-se a ordenação corrente da conjunção de metas (nada é feito com relação ao modelo de mundo). Além disso, estes são sistemas “flat knowledge”, sem nenhum tipo de estruturação. Os operadores representam ações primitivas e têm uma mesma oportunidade de serem aplicados para qualquer estado factível.

As primeiras tentativas de estruturação do problema de “planning” apontaram para a hierarquização dos planos.

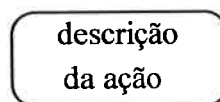
2.2 O Planeamento Hierárquico

A idéia de se fazer o planeamento usando níveis hierárquicos de abstração foi proposto pela primeira vez no sistema Nets of Action Hierarchies - NOAH [Sacerdoti 1975]. Neste sistema o plano não é mais linear. Ao invés, é feita uma decomposição das ações em sub-ações objetivando obter um melhor desempenho na busca das soluções. O SIPE [Wilkins 1984], [Wilkins 1988] é um sistema de planeamento independente do domínio que suporta a geração, automática ou interativa, de planos hierárquicos, parcialmente ordenados. No SIPE, os planos são representados por redes procedimentais, como no NOAH. Além disso, o SIPE usa dois tipos de representação do conhecimento para representar o domínio do problema: “frames” e cálculo de predicados.

predecessor \Rightarrow nó \Leftarrow sucessor



(a) meta



(b) phantom



(c) split



(d) join

onde,

meta: meta a ser atingida

phantom: metas supostas como verdadeiras

disjunção (split): submetas concorrentes (conflitos)

conjunção (join): síntese de sub-planos das submetas

Figura 6 Tipos de nós na rede procedimental NOAH

2.2.1 O NOAH

[Sacerdoti 1975] observa que, apesar de a execução do plano ser, geralmente, efetuada passo-a-passo, de uma forma linear, o plano em si não é linear. Ao invés, este consiste de uma seqüência de ordens parciais de ações no tempo. O NOAH (Nets of Action Hierarchies) [Sacerdoti 1975] usa um grafo procedimental para fazer a representação do plano. Cada nó do grafo procedimental representa uma ação. O autor afirma que o grafo procedimental dispensa informações adicionais como listas de adição e listas de eliminação. Existem quatro tipos de nós no grafo. A Figura 6 acima ilustra estes tipos de nó.

No grafo procedimental, os planos são seqüências de ações parcialmente ordenados no tempo e podem ser representados por grafos distintos para cada nível de abstração. Deste modo, para cada nível da hierarquia, os nós do grafo formam seqüências que constituem o plano naquele nível de detalhe. No NOAH utiliza-se uma hierarquia de submetas. Por exemplo, seja a rede procedimental que representa um plano para o problema de se pintar o telhado e os degraus de uma escada, ilustrada nas Figuras 7 (a)-(d), a seguir.

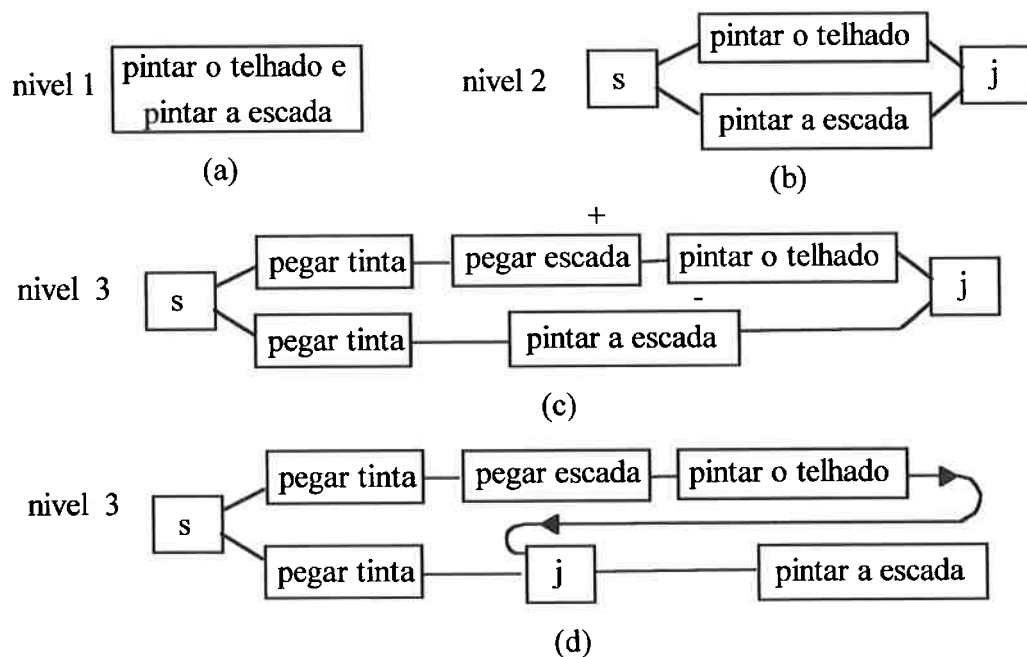


Figura 7 Exemplo da rede procedimental NOAH

O plano para esta tarefa no nível mais abstrato possível (nível 1) pode ser representado por um único nó, como mostra a Figura 7 (a). Detalhando este plano (nível 2), podemos ver que se trata de uma conjunção de duas tarefas *paralelas*, como mostra a Figura 7 (b). Decompondo estas tarefas em sub-tarefas com ainda maior detalhe (nível 3), podemos ter "pegue a tinta, pegue a escada e então aplique a tinta no telhado" e "pegue a tinta e aplique a tinta na escada" como mostra a Figura 7 (c).

Um procedimento hierárquico é usado para se fazer o planejamento. Ele inicia no nível mais alto de abstração e prossegue gerando planos cada vez mais detalhados para os demais níveis, de maneira "top-down". À medida que este processo de detalhamento é feito, diversos problemas podem ocorrer como: (1) conflito entre tarefas paralelas, (2) raciocínio não-monotônico devido a falha -- a não-monotonicidade deve-se à natureza não linear do problema de planejamento -- e (3) geração de pré-condições redundantes. O novo plano que resulta da expansão global pode apresentar problemas que tornam o plano global inactivel. Estes são sanados por meio de críticas aos planos gerados.

O algoritmo do processo de planejamento NOAH é o seguinte:

- 1) simular o plano mais detalhado na rede procedimental -- isto irá produzir um novo plano, mais detalhado.
- 2) criticar (e corrigir) o plano obtido no PASSO1, efetuando toda reordenação necessária e eliminando tarefas redundantes.
- 3) voltar ao passo 1.

[Sacerdoti 1975] apresenta três tipos de críticas genéricas:

- a) Resolução de conflitos: Tarefas paralelas no plano antecedente devem ser ordenados de modo a não tornar o objetivo global inatingível. O NOAH usa tabela de múltiplos efeitos que tem toda expressão afirmada ou negada por mais de um nó no plano corrente para detectar interferências.
- b) Uso de objetos existentes: Com o objetivo de evitar "backtracking" e soluções parciais, o NOAH usa uma abordagem do "menor comprometimento", i.e., durante o

planejamento evita-se associar objetos muito específicos às variáveis. Ao invés, usa-se um objeto formal que tem o papel de "place holder", i.e., uma entidade ainda não instanciada. Esta crítica substitui objetos formais por objetos reais sempre que possível.

- c) Eliminar pré-condições redundantes: As críticas anteriores podem tornar redundantes algumas das pré-condições estabelecidas no plano antecedente.

No caso do exemplo da Figura 7 (c) existe um conflito pois "pintar a escada" irá efetivamente tornar falsa a proposição "pegar escada". Em tal situação, o conflito irá ocorrer porque "pegar escada" é uma pré-condição para "pintar o telhado". Na forma gráfica, o conflito é denotado por um "+" sobre a pré-condição violada e um "-" na causadora da violação. O conflito é resolvido ao se definir que operador correspondente à pré-condição ameaçada ("pintar o telhado") seja realizada antes da ação causadora da violação ("pintar a escada"). Esta crítica está ilustrada na Figura 7 (d).

A contribuição importante do sistema NOAH está no uso de planos não lineares hierárquicos, construídos de uma forma "top-down", ou seja, primeiro as primeiras coisas. As decisões de mais alto nível são decididas antes, sem detalhar como estes são efetuados pelos níveis de abstração de nível mais baixo. Entretanto, no caso do plano para algum nível não poder ser executado, a correção do plano elaborado é feita de uma forma "bottom-up".

No sistema NOAH, podemos identificar, pela primeira vez a tentativa do uso de grafos para estruturar o problema de planejamento em IA. Esta estruturação que foi feita sobre o sistema STRIPS ainda é muito limitada porque foi feita apenas para o plano e não sobre o domínio do problema. Mesmo os mecanismos de crítica propostos para a correção das inconsistências são relativamente simples. Apesar de o sistema pretender elaborar planos não lineares, estes são lineares dentro de cada nível de abstração. Um plano contendo inconsistências deve ser elaborado inicialmente para que, então, se faça a sua correção através do mecanismo da crítica.

2.2.2 O SIPE

O SIPE (System for Interactive Planning and Execution Monitoring) [Wilkins 1984], [Wilkins 1988] é um sistema de planejamento independente do domínio que suporta a geração, automática ou interativa, de planos hierárquicos, parcialmente ordenados.

Um plano SIPE é um conjunto de metas e ações parcialmente ordenados elaborado pelo sistema a partir dos operadores que descrevem as ações. Eventualmente, este pode gerar um plano que não atinge a meta desejada. Neste caso, o sistema efetua críticas para detectar problemas potenciais e tentar evitá-los. No SIPE, o plano é representado na forma de redes procedimentais como em [Sacerdoti 1975]. O SIPE usa dois tipos de representação do conhecimento: "frames" e cálculo de predicados. As propriedades invariantes dos objetos estão representadas em "frames", que permitem herança de propriedades e imposição de limites nos valores dos atributos destes objetos. As demais propriedades estão representadas usando o cálculo de predicados de primeira ordem, que interage com o conhecimento representado em "frames".

O SIPE apresenta:

- formalismo para representação de ações;
- representação de propriedades de objetos que não se modificam (invariantes) ao longo do tempo;
- descrição parcial de objetos através da limitação de valores de seus atributos;
- mecanismos para a exploração concorrente de planos alternativos;
- heurísticas para raciocínio acerca de recursos;
- mecanismos para efetuar dedução dos efeitos das ações e
- habilidade para raciocinar acerca da interação entre ações paralelas.

O SIPE assume tempo discreto, estados discretos e operadores discretos. O tempo não é descrito explicitamente porque a ordem parcial dada na rede procedimental provê a informação de ordenação necessária.

Um operador SIPE apresenta atributos para especificar suas pré-condições e metas: (1) representa apenas o conhecimento de como atingir metas (e não como atingir pré-condições); (2) a pré-condição é útil para se fazer a conexão entre diferentes níveis de detalhe na hierarquia. O SIPE constrói ligações entre os níveis do plano, e dentro de cada nível para auxiliar na representação da lógica de planejamento por trás de cada ação. A pré-condição de um operador pode especificar que uma determinada condição de mais-alto-nível deve ser verdadeira, enquanto que o mesmo operador pode especificar que metas devem ser atingidas num nível de hierarquia de maior detalhe.

A capacidade do SIPE em interligar níveis de hierarquia é aumentada pelo atributo *propósito* do operador. Este atributo especifica o que se está tentando atingir no nível de menor detalhe e é usado para se determinar a lógica do planejamento. O atributo meta do operador especifica o que o operador está tentando atingir no nível de maior detalhe. Operadores SIPE contêm sub-planos ("plots"), representados como redes procedimentais, que especificam como as ações devem ser efetuadas em termos de ações e metas. Operadores SIPE permitem a imposição de limites para valores dos objetos, especificação de recursos, representação explícita da lógica por trás de cada ação, e o uso de dedução para se determinar o efeito das ações.

A abordagem hierárquica nada mais é que a técnica de decomposição "top-down", importante para o "design" estruturado de sistemas. Esta faz a decomposição do problema em subproblemas para diminuir a sua complexidade. Entretanto, no mundo real, freqüentemente os subproblemas interagem entre si. Se não existir uma forma de se tratar estas interações entre os subproblemas, os sistemas de planejamento hierárquicos deste tipo somente são úteis nos poucos casos teóricos em que os subproblemas são completamente independentes e podem ser resolvidos em paralelo. Os subproblemas interagem entre si, principalmente nos problemas de planejamento e de "design".

Para se tratar as interações entre os subproblemas é feita uma imposição de restrições que são depois propagadas. O planejamento hierárquico cria descrições dos estados abstratos e divide a tarefa do planejamento em subproblemas de refinamentos suces-

sivos dos estados abstratos. O espaço abstrato permite focar a atenção nas considerações importantes, evitando a confusão que surge ao se tentar resolver tudo de uma só vez.

O SIPE constrói descrições parciais de objetos ainda não-instanciados, i.e., restringe os possíveis valores que as variáveis de planejamento podem assumir. Estas descrições parciais introduzem complexidade ao sistema porque interagem com todas as suas partes. Durante o planejamento, SIPE assume que é possível satisfazer qualquer limitação que não possa ser provada insatisfável, adiando a decisão sobre satisfabilidade o máximo possível. A verificação de satisfabilidade, executando a rotina de satisfação destas limitações de valores, só é feita uma única vez em cada nível da hierarquia através da busca automática.

A principal contribuição do SIPE está no uso de “frames” para fazer a especificação parcial (de invariantes) de objetos e para raciocinar acerca dos recursos. O SIPE provê uma linguagem geral para se fazer a programação por restrições.

O NOAH não é realmente top-down” porque não admite a verificação do plano em cada nível de abstração, mas sim “bottom-up” depois que o plano foi elaborado. Neste aspecto, houve um avanço no SIPE, porque a especificação parcial de objetos usando a representação em “frames” permite uma verificação “top-down”. Porém, a estruturação baseada em “frames” é feita sobre o plano e não sobre o contexto do problema de “planning”.

2.3 O PLANEJAMENTO DINÂMICO

O problema do planejamento tem por objetivo decidir, por exemplo, se um conjunto de tarefas deve ser executado de acordo com o contexto do problema. O problema da seqüenciação tem por objetivo fazer a alocação dos recursos às operações que foram planejadas e decidir exatamente quando estas ações devem ser iniciadas. Em outras palavras, o planejamento define o que (e quanto) deve ser feito, enquanto a seqüenciação define quando (e onde -- ou seja como) o que foi planejado deve ser feito. A forma

como uma determinada tarefa é executada pode determinar o grau de dependência entre estes dois problemas. Se a dependência é forte, a forma como a execução do plano é feita impõe limitações ao que (e a quanto) se pode ser feito e a quando pode ser feito.

O problema da seqüenciação pode ser visto como um problema combinatório que, para ser tratável, deve satisfazer a um conjunto de condições. O plano define uma seqüência de ordem parcial de tarefas objetivando atingimento da meta desejada. Cada passo da tarefa é constituído por diversas operações. A cada operação estão associadas dois tipos de condições: tecnológica e temporal. As condições tecnológicas descrevem os recursos necessários para se efetuar a operação. As condições temporais descrevem a ordem parcial entre as operações e os prazos de entrega a serem atendidos. O problema da seqüenciação tem a característica de ser dinâmico e aberto. Eventualmente, condições tecnológicas ou temporais podem ser impostas ou eliminadas sem o controle do planejador. A complexidade do problema aumenta no planejamento de tarefas no domínio dos sistemas dinâmicos, tais como no domínio da robótica, em que o planejamento deve ser feito simultaneamente com a execução do plano, em tempo real.

Uma abordagem tradicional para este problema é a da otimização. Entretanto, devido ao comportamento dinâmico que geralmente caracteriza o domínio destes problemas, um plano ótimo é facilmente descumprido. Nestes casos, é preferível se produzir planos que são satisfatórios, factíveis de executar, embora não sendo ótimos, e manter a factibilidade destes planos dentro do ambiente dinâmico. Existem duas abordagens possíveis para se obter um plano satisfatório, mas não ótimo: através de agentes de planejamento cognitivos e através de agentes reativos. Um Planejador que atua como agente cognitivo assume que o mundo é estático e que todos os eventos são inteiramente previsíveis. Este constrói incrementalmente um plano completo. Uma variante desta abordagem consiste em se elaborar tantos planos de contingência quanto as alternativas de execução previstas com antecedência, armazená-los e invocar um deles dependendo da circunstância atual. Caso as circunstâncias sejam ligeiramente diferentes, este deve ser adequado à situação corrente. Um Planejador que atua como agente reativo atua num estilo estímulo-resposta, reagindo à percepção sensorial que tem do ambiente em que atua. Este responde apropriadamente aos estímulos do ambiente no sentido de manter a

factibilidade de um plano. A idéia geral é de que a ação é mais explicada com base nas circunstâncias e menos pelas metas que se espera que o sistema vá atingir.

Objetivando a obtenção de sistemas para a resolução do problema do planejamento de tarefas no domínio dos sistemas dinâmicos, foram propostas inicialmente abordagens como: planejamento total com replanejamento na falha [Hayes 1975], aprendizado e generalização STRIPS+PLAN-EX [Fikes, Hart et Nilsson 1972] e planos incompletos [Mc Dermott 1978].

Entretanto, é importante observar que estes planejadores fazem determinadas hipóteses acerca da representação do mundo, e o mundo é atualizado usando-se as mesmas operações de adição / eliminação definidas com as ações usadas na representação STRIPS.

2.3.1 O Aprendizado e a Generalização

[Fikes, Hart et Nilsson 1972] propuseram algumas extensões ao STRIPS para que o sistema fosse capaz de aprender com o processo de planejamento. A idéia é a de se reutilizar o "planning rationale" no replanejamento subsequente que se tornar necessário, por exemplo, devido a ocorrência de uma falha qualquer na execução do plano. A primeira extensão é um processo de generalização do plano produzido pelo STRIPS de uma forma que as constantes específicas do problema que aparecem no plano são substituídas por variáveis independentes do problema. O plano generalizado é armazenado numa forma específica chamada tabela triângulo. O plano generalizado pode ser empregado de duas maneiras: (1) como macro-ações (MACROPS) que podem ser reutilizadas pelo STRIPS na elaboração de planos subsequentes; (2) na tomada de ação reativa inteligente pelo robô quando da execução do plano, provendo maior flexibilidade. A segunda extensão é um processo de especialização que permite o emprego do plano generalizado destas duas maneiras.

STRIPS+PLAN-EX são dois componentes de um sistema que faz a elaboração de planejamento (STRIPS) e a monitoração da execução (PLAN-EX). Este sistema é capaz de generalizar e guardar planos genéricos a partir de um plano para solucionar um problema específico. PLAN-EX é um sistema supervisor da execução do plano e deve responder a cada passo:

- a) a parte executada do plano atingiu os resultados desejados ?
- b) que parte do plano original que deve ser executada a seguir, de modo que a tarefa seja concluída com sucesso ?
- c) esta parte do plano pode ser realizada no estado corrente ?

O plano gerado pelo STRIPS com n operadores é uma seqüência de operadores Op_1, Op_2, \dots, Op_n . Os operadores mudam o modelo de mundo*³, do mesmo modo que as ações que elas representam mudam o estado do domínio do problema. PLAN-EX armazena este plano numa estrutura específica chamada tabela triangular, cuja ilustração está na Figura 8. Portanto, a tabela triangular serve para armazenar o plano e serve também para fazer a monitoração da execução do plano.

A tabela triangular é uma matriz triangular inferior onde as linhas e colunas da matriz correspondem à operadores do plano. O conteúdo da i -ésima linha na tabela triangular, excluindo-se a coluna mais à esquerda, especifica cláusulas inseridas pela aplicação de $(i-1)$ operadores na seqüência $Op_1, Op_2, \dots, Op_{i-1}$. As colunas da tabela, com exceção da Coluna zero que corresponde às pré-condições (PC_i) do operador Op_i , são rotuladas com os nomes dos operadores no plano, por exemplo, Op_1, Op_2, \dots, Op_n . Para cada coluna $i, i = 1, 2, 3, \dots$ colocamos a lista de adição A_i do operador Op_i na célula mais alta. Descendo pela i -ésima coluna, colocamos a porção de A_i que sobrevive à aplicação dos operadores subsequentes nas células consecutivas. Deste modo, $A_{1/2}$ denota cláusulas em A_1 não eliminadas por Op_2 ; $A_{1/2,3}$ denota cláusulas em A_1 não eliminadas por Op_2 e por Op_3 ; $A_{2/3}$ denota cláusulas em A_2 não eliminadas por Op_3 e

*³ O uso da palavra "modelo" aqui é consistente com o uso costumeiro da palavra em IA. Não fazer confusão com a definição técnica usada em lógica, i.e., uma interpretação para um determinado conjunto de fórmulas.

assim por diante. Deste modo a ij -ésima célula da matriz contém fórmulas bem formadas (fbf) adicionadas pelo j -ésimo operador e ainda verdadeiras (sobreviventes) depois da aplicação do i -ésimo operador. A Figura 8, ilustra a estrutura da tabela triângulo para $n = 3$ operadores.

1	PC ₁	OP ₁		
2	PC ₂	A ₁	OP ₂	kernel(3)
3	PC ₃	A _{1/2}	A ₂	OP ₃
4		A _{1/2,3}	A _{2/3}	A ₃
	0	1	2	3

Figura 8 A tabela triângulo PLAN-EX para armazenar um plano STRIPS

Define-se o conjunto de cláusulas utilizadas para a prova de uma fórmula como o conjunto suporte daquela fórmula. Com o uso da tabela triangular busca-se assegurar que o conjunto suporte das pré-condições do operador Op_i está presente na i -ésima linha da matriz. As cláusulas no conjunto suporte podem ser adicionadas através da aplicação dos $(i-1)$ operadores ou, ainda, serem verdadeiras no estado inicial e não terem sofrido nenhuma modificação até aquele instante. Estas são denotadas por PC_i , i.e., as pré-condições do operador Op_i , e são exatamente as cláusulas inseridas na coluna mais à esquerda da tabela triângulo (coluna zero).

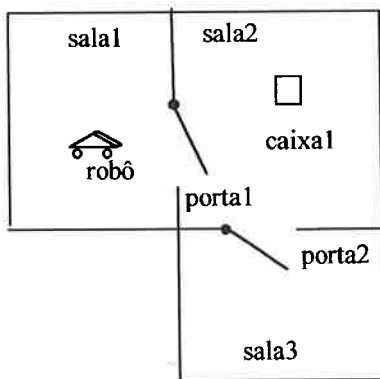


Fig. 9 O Problema STRIPS

			Obs: (*) cláusulas marcadas
1	* esta_em(robô,S1) * conecta(P1,S1,S2)	passa_atraves(P1,S1,S2)	
2	* esta_em(C,S2) * conecta(P1,S1,S2) * conecta(X,Y,Z) → conecta(X,Z,Y)	* esta_em(robô,S2)	empurre_atraves(C,P,S1,S2)
3			esta_em(robô,S1) esta_em(C,S1)
	0	1	2

Fig. 10 A tabela triângulo PLAN-EX

Deseja-se assegurar que a i -ésima linha da tabela triângulo contém todas as fbfs no conjunto suporte das pré-condições para o operador Op_i . Todas as cláusulas no conjunto suporte de um operador Op_i são marcadas na tabela triângulo. Por construção todas as cláusulas da coluna zero são marcadas. Observe na Figura 10 que as cláusulas marcadas com "*" foram todas usadas para a prova de pré-condições dos operadores.

Na monitoração da execução do plano, PLAN-EX usa a noção de "kernel" para garantir a execução do restante de plano. O kernel para o operador Op_i , ou seja o $kernel(i)$, corresponde ao retângulo na tabela triangular que contém a célula do canto inferior esquerdo da tabela triângulo e a linha i . A Figura 8, ilustra o kernel para a linha 3, i.e., o $kernel(3)$. O restante do plano ("tail"), i.e., a seqüência $Op_j, Op_{j+1}, \dots, Op_n$ é aplicável se todas as cláusulas marcadas no i -ésimo kernel forem verdadeiras naquele modelo. Em outras palavras, o restante do plano é aplicável para um modelo (ou estado do mundo) se o modelo já contém a parte do conjunto suporte de cada operador no restante do plano que não é suprida pelo próprio restante do plano. Isto porque se parte das sentenças definidas como pré-condições para os operadores no $kernel(i)$ que não é suprida pelo próprio restante do plano já são verdadeiras, o restante das sentenças definidas como pré-condições serão supridas pelo restante do plano.

A Figura 9, na página anterior, mostra o domínio de um problema exemplo STRIPS. Este problema envolve uma caixa, três salas conectadas através de duas portas e um robô. A Figura 10 mostra o posicionamento de cláusulas na tabela triângulo PLAN-EX que guarda o plano para este exemplo. O conjunto de cláusulas que definem o estado inicial do modelo está mostrado a seguir.

Modelo inicial: esta_em(ROBO,SALA1)
 conecta(PORTA1,SALA1,SALA2)
 conecta(PORTA2,SALA2,SALA3)
 caixa(CAIXA1)
 esta_em(CAIXA1,SALA2)

fbf meta: FAÇA: $(\exists (x) [CAIXA(x) \wedge esta_em(x,SALA1)]$

A meta, mostrada acima, é a de se posicionar o objeto x onde x é caixa na *SALA1*. Os dois operadores que definem as ações do robô estão mostrados a seguir: *passa_através* ($P, S1, S2$) e *empurre_atraves* ($C, P, S1, S2$).

Operador: $passa_através(P, S1, S2)$
Pré-condição: $esta_em(robô, S1) \wedge conecta(P, S1, S2)$
Lista de adição: $esta_em(robô, S2)$
Lista de eliminação: $esta_em(robô, S1)$

Operador: $empurre_atraves(C, P, S1, S2)$
Pré-condição: $esta_em(C, S2) \wedge esta_em(robô, S2) \wedge conecta(P, S1, S2)$
Lista de adição: $esta_em(C, S1) \wedge esta_em(robô, S1)$
Lista de eliminação: $esta_em(C, S2) \wedge esta_em(robô, S2)$

A generalização do plano no PLAN-EX consiste de dois passos: (1) mapeamento (que o autor chama de "lifting") da tabela triângulo para numa forma mais geral, através da inserção de variáveis ou parâmetros. (2) restrição dos valores para as variáveis inseridas no passo 1. No passo 1, de mapeamento (ou "lifting"), troca-se inicialmente cada ocorrência de constantes na coluna zero por variáveis distintas (ocorrências múltiplas de mesma constante criam variáveis distintas). Depois, preenche-se as listas de adição usando-se variáveis ainda não instanciadas. No passo 2, restrições são obtidas refazendo-se a prova da pré-condição para cada novo operador. Usa-se cláusulas marcadas na tabela mapeada ("lifted") como axiomas e as fórmulas originais na pré-condição dos operadores como teoremas a provar. Este processo de prova por teoremas, assegura que cada linha na tabela original é uma instância da linha correspondente na tabela mapeada. No refinamento após a generalização, antes de armazenar o plano generalizado como macro operador (MACROP), o sistema deve remover:

- *super_generalização:* Ex. duas variáveis criadas a partir de uma única ocorrência de constantes numa cláusula.
- *inconsistência:* Ex. cláusulas sobreviventes na tabela triangular mapeada podem não ser verdadeiras em função de como unificam as variáveis.

Já vimos que o plano generalizado pode ser empregado de duas maneiras:

- (1) para prover maior flexibilidade durante a execução do plano, e
- (2) como macro-ações (MACROPS) reutilizáveis pelo STRIPS no replanejamento.

Ao invés de trabalhar com uma instância específica do plano, STRIPS+PLAN-EX trabalha com uma MACROP generalizada, para tornar a execução do plano mais flexível. Na preparação de uma MACROP para execução, as variáveis do MACROP são parcialmente instanciados para objetos específicos, i.e., MACROP é especializado para a tarefa que está sendo feita. A cada estado da execução, PLAN-EX deve ter pelo menos um kernel verdadeiro para poder prosseguir com a execução do plano. Assim, PLAN-EX verifica a partir do último kernel no sentido regressivo, até encontrar um kernel verdadeiro. Se o último kernel é verdadeiro, então FIM da execução. Caso contrário, executa o operador Op_i correspondente ao kernel(i) e verifica o resultado, e assim por diante. Quando nenhum kernel está verdadeiro é feito replanejamento com STRIPS.

O STRIPS+PLAN-EX emprega MACROPS na elaboração de um novo plano ou no replanejamento. Isto constitui um tipo de aprendizado, onde um algoritmo é usado para extrair subseqüências relevantes de operadores a partir de MACROPS arquivadas.

As subseqüências de operadores são escolhidas de modo que listas de adição promovem a maior redução de diferenças. Então, o sistema inclui estas subseqüências no novo plano em elaboração. O novo plano gera novo MACROP que poderá subsidiar em futuras tarefas de planejamento. Este pode conter subseqüências já existentes ou não. As subseqüências já existentes são eliminadas.

O STRIPS+PLAN-EX apresenta uma estruturação do plano, que é armazenado na forma sentenças lógicas numa estrutura chamada de tabela triangular, para efetuar a recuperação do sistema em caso de falha na execução do plano corrente. A monitoração da execução do plano leva em conta a idéia de kernel da tabela triângulo. Note que, ao contrário da nossa proposta de estruturação, descrita mais adiante, STRIPS + PLAN-EX faz apenas a estruturação do plano e não do domínio do problema. Além disso, a representação do plano é feita através de uma extensão da representação STRIPS.

2.3.2 O Replanejamento na Falha

Nos problemas do mundo real, ocorrem eventos ocasionados por outros agentes e sobre os quais o sistema planejador não tem nenhum tipo de controle. Dentro da abordagem de agentes cognitivos de planejamento, alguns sistemas de planejamento baseado em IA fazem o replanejamento sempre que o sistema planejador reconhece uma discrepância entre o estado esperado e o estado real do mundo. [Hayes 1975] propôs que árvores de submetas e grafos de decisão utilizados na elaboração do plano fossem usados para guiar o replanejamento. Este sistema usa o grafo de decisão para anotar as dependências entre componentes num exemplo de plano de viagem através da Europa usando trem, avião e navio.

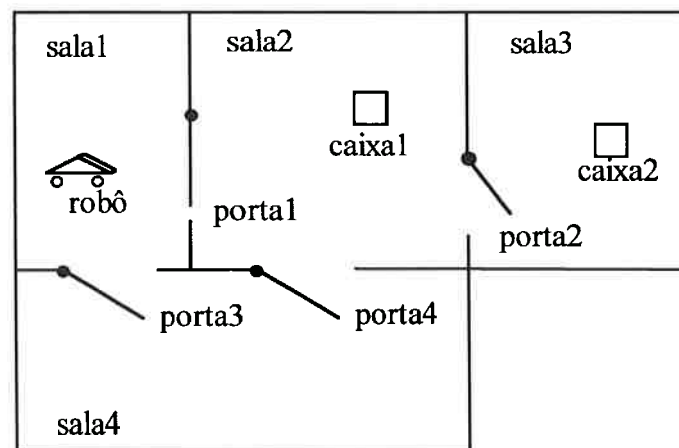


Figura 11 A Representação do domínio do planejamento de robôs [Hayes 1975]

[Hayes 1975] propôs uma estrutura que representa e torna disponível todas as informações acerca de todos os passos do processo de elaboração do plano. Esta informação é usada tanto no processo de elaboração do plano original assim como para a sua modificação quando ocorrer eventos inesperados que causam uma falha na sua execução.

A estrutura da representação envolve duas estruturas de dados: uma árvore de submetas (j-nodes) e um grafo de dependência de decisões (d-nodes), veja as Figuras 12(a) e 12(b), a seguir.

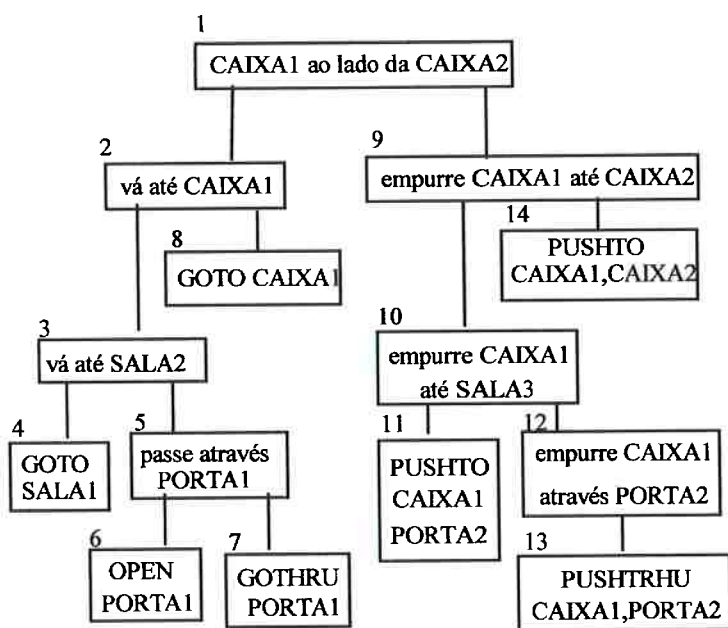


Fig. 12 (a) A Árvore de submetas para o problema de se juntar duas caixas, na Fig. 11

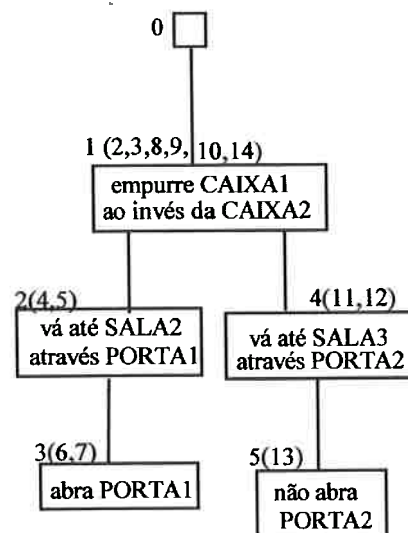


Fig.12 (b) O Grafo de decisão do problema

Na árvore de submetas, cada nó (j-node) corresponde a uma meta e as folhas às ações necessárias para o seu atingimento. Submetas estão representadas por uma ramificação na árvore. Deste modo, o nó raiz da árvore corresponde à meta de maior nível do plano, enquanto que as folhas da árvore correspondem às ações primitivas.

Cada nó (d-node) do grafo de dependência de decisões corresponde a uma decisão. A relação hierárquica entre nós indica dependência lógica entre as decisões. Cada nó (d-node) aponta para nós da árvore de submetas (j-nodes) que correspondem às metas geradas como consequência desta decisão. Estes ponteiros são usados em conjunção com a estrutura do grafo (de decisões) para identificar precisamente todos os efeitos diretos e indiretos de uma decisão no desenvolvimento do plano. O processo de se desfazer os efeitos (assim identificados) de uma decisão para um plano é chamado de "undoing" e consiste dos seguintes passos:

- 1) remover o nó correspondente à decisão do grafo de decisões (d-node) e remover todos os nós na árvore de submetas (j-nodes) que estejam apontados por aquele d-node.
- 2) eliminar nós descendentes do nó removido do grafo de decisões (d-nodes).

O grafo de dependência de decisões (d-nodes) é usado para se eliminar seletivamente partes do plano quando ocorre falha durante o planejamento (elaboração do plano) ou durante sua execução. Em caso de falha na execução, para se fazer o replanejamento, o monitor marca um conjunto de submetas (j-nodes) como não executáveis. Então, para tais submetas não executáveis de maior hierarquia são executados os seguintes passos:

- 1) toda porção da árvore de submetas (j-nodes) efetuada com sucesso e nós (d-nodes) no grafo de decisões diretamente responsáveis por elas são descartadas.
- 2) toda informação da árvore de submetas (j-nodes), falsificada como resultado da falha ocorrida, é atualizada.
- 3) para cada nó (j-node), A, do conjunto de submetas marcado como não executável:
 - 3.a) se o nó A já foi eliminado, então não há nada a fazer. Caso contrário, passo 3.b.
 - 3.b) o nó (d-node) D diretamente relacionado com A é desfeito ("undone").
 - 3.c) o processo de elaboração do plano é retomado a partir do ponto de retomada D.
 - 3.d) se a elaboração retomada do plano termina com sucesso, então nada mais é feito (FIM). Caso contrário, passo 3.e.
 - 3.e) se o nó A não é o nó raiz da árvore, ele é substituído pelo seu nó pai e volta ao passo 3.b. Caso contrário, passo 3.f.
 - 3.f) o replanejamento em questão é impossível.

Seja o exemplo do problema envolvendo um robô, duas caixas, quatro salas conectadas através de quatro portas, como mostra a Figura 11, na página 41. As Figuras 12 (a) e 12 (b), na página 42, ilustram, respectivamente, a árvore de submetas e o grafo de decisões para o problema da Figura 11.

Vamos ilustrar o processo de replanejamento usando a árvore de submetas e o grafo de decisões. Por exemplo, suponha que na execução do plano elaborado, o robô descobre que a PORTA1 não apenas está fechada mas trancada. O sistema aplicaria o procedimento de replanejamento para o j-node 5 na árvore de submetas, mostrada na Fig. 12 (a), porque esta é a submeta de maior nível hierárquico não executável ancestral

do j-node 6, que é a ação primitiva não executável de abrir a PORTA1. Inicialmente, o j-node 4 será descartado por já ter sido efetuada com sucesso. Então, o banco de dados global é atualizado com a posição presente do robô acrescido do fato de que a PORTA1 está trancada.

O d-node 2, a escolha da rota do robô a partir da SALA1 para SALA2 é a decisão diretamente responsável pelo j-node 5 e, deste modo, deve ser desfeito ("undone"), removendo-se os j-nodes 5,6 e 7 a partir da árvore de submetas e os d-nodes 2 e 3 do grafo de decisão. Com o conhecimento agora disponível de que a PORTA1 está trancada, a retomada do processo de elaboração do plano deverá resultar numa rota a partir da SALA1 até a SALA2 passando através da SALA4. No novo plano, assim refeito, a árvore de submetas representando a ação do robô deslocar da SALA1 até a SALA2 será detalhada a partir do j-node 3. Todo o resto do plano original acerca de empurrar a CAIXA1 até a SALA3 através da PORTA2 permanece evidentemente intacto neste processo.

[Hayes 1975] utiliza duas estruturas: (i) uma árvore de submetas para armazenar simultaneamente o plano e (ii) um grafo de decisão para armazenar como foi o processo de elaboração deste plano. Além disso, é guardado o relacionamento entre estas estruturas. Esta proposta é um avanço em relação a estrutura da tabela triângulo usada para armazenar o plano em STRIPS+PLAN-EX, porque consegue-se fazer uma representação mais expressiva do "planning rationale" tanto na elaboração do plano original assim como para se fazer uma modificação deste.

Tanto a árvore de decisão quanto o grafo de decisão apresentam a vantagem da representação gráfica inerente à teoria de grafos. A desvantagem está na necessidade de se utilizar diversas estruturas, uma para representar o plano e outra representar o "design rationale" que foi usado no processo de elaboração do plano. Além disso, o domínio da aplicação continua sem qualquer tipo de estruturação.

2.3.3 Planos Incompletos

[Mc Dermott 1978] critica a abordagem que precisa desenvolver todo o plano completo, argumentando que, geralmente, a elaboração de um plano minimamente confiável depende da antecipação correta dos estados do mundo no momento da execução do plano, a qual é muito difícil de ser feita. O autor propôs uma nova teoria que associa planejamento e execução. Para isto, propõe uma abordagem onde os passos da elaboração de um plano incompleto são obtidos "em pleno vôo". O plano incompleto é executado tão logo as regras de escalonamento o permitam. O sistema expande partes do plano enquanto este vai sendo executado. A noção de tarefa é vista como qualquer ação que o sistema está efetuando ou deve efetuar. Um problema é uma tarefa que não pode ser executada imediatamente na forma de uma tarefa primitiva. Redução é o processo de se subdividir uma tarefa em sub-tarefas. Um plano para a solução de um problema é o conjunto de sub-tarefas necessárias para que a tarefa original seja efetuada com sucesso. Executar uma tarefa primitiva significa simplesmente invocar um código escrito, por exemplo, em LISP.

estado da tarefa:	PENDENTE		ATIVA		TERMINADA
estado da habilitação:	Bloqueada	Habilitada	Sub-tarefas Habilitadas	Sucessoras Habilitadas	

Figura 13 - O Ciclo de Vida de uma tarefa NASL

[Mc Dermott 1978] fez a implementação desta teoria em STP + NASL. STP é um sistema de planejamento baseado em provador de teoremas. NASL o sistema de monitoração da execução do plano. NASL é um interpretador que pode executar tarefas de um plano incompleto assim que regras de escalonamento o permitam. NASL utiliza inferência lógica para retornar condicionalmente comandos de adição e de eliminação usados pelos operadores. Deste modo, o comportamento do NASL pode ser modificado pelas contexto do ambiente, por exemplo, poderia existir dois comandos distintos de adição de cláusulas para uma ação de *ir para* ("goto") dependendo ou não desta ação significar ou não o atingimento do destino final. Este uso do retorno dedutivo permite ao NASL prover um tratamento uniforme dos operadores de planejamento, não importando

se eles obtêm sucesso ou falham durante a execução do plano. NASL não permite divisão de sub-tarefas na redução.

A Figura 13 acima, ilustra o ciclo de vida de uma tarefa NASL. Quando uma tarefa NASL é criada, ela recebe dois tipos de estados ("status") que se modificam através das várias transições. O estado da tarefa ("task status") indica se o trabalho nesta tarefa está: pendente, ativa ou terminada. O estado de habilitação ("task enablement") expressa seu relacionamento com outras tarefas que pode estar: bloqueada, habilitada, sub-tarefas habilitadas e sucessoras habilitadas. Por exemplo, uma determinada tarefa está habilitada se todas suas tarefas antecessoras estiverem terminadas ou uma super-tarefa a habilita enquanto sub-tarefa.

O laço ("loop") principal do interpretador NASL é o seguinte:

```
Pegue a tarefa da vez
SE ação primitiva ENTÃO execute
CASO CONTRÁRIO, reduza
Repita até não existir mais tarefas
```

No primeiro passo, NASL pega uma tarefa qualquer habilitada. No NASL uma ação primitiva pode ser uma função LISP descrita por: nome da ação, descrição da ação, lista de adição e lista de eliminação. Executar uma ação primitiva consiste em atualizar base de conhecimentos acreditada pelo NASL. Uma ação primitiva é executada. Observe que NASL+STP utiliza, portanto, uma representação derivada do sistema STRIPS.

Um problema é reduzido invocando o provador de teoremas STP. Se STP retorna exatamente uma ação então uma nova tarefa é criada, habilitada e considerada sub-tarefa principal ("main") da tarefa corrente. Se o STP retornar mais de uma ação, então, NASL dispara a função escolha ("choice"). Quando ocorre erro no planejamento, i.e., quando nenhuma ação é retornada pelo STP, é feita a re-especificação do problema ("re-phrasing"). O problema não resolvido é re-submetido após redefinição do problema e re-

especificação das metas. Se existe uma solução armazenada para o problema, então esta rotina é desviada ("bypassed"). Neste caso, sub-tarefas do plano guardado são habilitadas. O autor considera este re-especificação como sendo uma falha do sistema, porque este problema deriva, provavelmente, de uma escolha anteriormente mal feita. Mas, o sistema NASL+STP não implementa nenhuma forma de backtracking.

Finalmente, [Mc Dermott 1978] observa que a teoria proposta não é completa pois carece de uma teoria formal acerca do tempo, de ações e de eventos. Destaca o papel de eventos e transições principalmente na antecipação de estados futuros do mundo e na análise de todos os possíveis cursos de ação, mesmo que alguns deles nunca venham a se realizar. Além disso, ressalta a importância de se definir concretamente o atingimento com sucesso ou falha de uma tarefa, assim como a necessidade de uma teoria para recuperação de erros.

Comparando com a nossa proposta de estruturação, podemos verificar que o NASL+STP utiliza uma forma de representar o estado de execução da tarefa e também o estado de habilitação da tarefa. O que caracteriza as Redes de Petri é justamente a sua capacidade de representar a dinâmica dos sistemas como um Sistema de Eventos Discretos (SED), de maneira que o módulo de monitoração da execução do plano pode ser representado em Rede de Petri, com vantagens. No NASL+STP, o detalhamento de um plano incompleto é feito durante o processo de execução do mesmo. Isto porque este sistema tem a preocupação com o tempo de resposta e está direcionado para aplicações em tempo real. A nossa a proposta de metodologia pode ser estendida para suportar a representação de problemas de "planning" e do domínio da aplicação, usando Redes de Petri de alto nível, para o desenvolvimento de sistema de planejamento que usam planos parciais que são detalhados em tempo de execução.

2.4) O Aprendizado de Heurísticas de Busca no “Planning”

Em nossa proposta, como veremos mais adiante, a estruturação do domínio da aplicação e do problema do “planning” usando redes de Petri constituem um modelo conceitual. A análise da estrutura destas redes permitem o aprendizado de heurística de busca para dirigir o processo de planejamento.

O aprendizado de heurísticas para fazer o “planning” e “scheduling” foi usando em ART [Day 1992]. ART é um sistema de planejamento baseado na satisfação de restrições, e aplicado para problemas de planejamento e seqüenciação em transportes. A satisfação de restrições é uma abordagem genérica para a resolução de problemas em que o domínio do problema é representado por um conjunto de variáveis e por um conjunto de condições que expressam os requisitos para os valores atribuídos a estas variáveis. Uma solução válida consiste de um conjunto de atribuições para todas as variáveis do problema de uma maneira tal que todas as condições são satisfeitas. A estratégia de busca padrão para este tipo de problema é a busca em árvore. Nesta, cada variável ainda não-atribuída é selecionada iterativamente e atribuído um valor para ela dentre os valores do domínio que satisfazem a todas as condições pertinentes até que todas as variáveis tenham sido atribuídas. Se todos os valores plausíveis para uma determinada variável causa a violação de condições, então invoca-se o backtracking ao longo da última atribuição de valor feita para uma variável.

Está implícito, na programação por restrições, a hipótese linear do plano, já discutido anteriormente para o sistema STRIPS. Por este motivo, a complexidade do processo de busca da solução pode variar de acordo com: a ordem em que a variável é atribuída e a ordem em que os valores para cada variável individual são atribuídos.

[Day 1992] enfatiza a obtenção automática de heurísticas para a ordenação de valores (candidatos a serem atribuídos para as variáveis). Este afirma que é possível se evitar a atribuição de valores não apropriados para as variáveis (porque estas irão levar a uma falha e, em conseqüência, ao backtracking). Isto pode ser feito usando-se o backtracking orientado por dependência, formando e estabelecendo condições que irão

excluir a atribuição de um conjunto de valores que “causam” o backtracking no futuro. Este processo é considerado como sendo um tipo de aprendizado.

O principal problema deste tipo de aprendizado é que ele não pode ser reutilizado para auxiliar o processo de solução de outros problemas semelhantes de satisfação de restrições. As condições induzidas para podar o espaço de busca para a solução do problema só podem ser usadas para outros problemas que apresentem uma rede de restrições idêntica. Para contornar este problema, as condições sobre as variáveis foram transformadas em restrições “soft” (ponderadas), cujo efeito nos problemas subsequentes é o de modificar a ordem sob a qual os valores são atribuídos às variáveis, mas, não elimina completamente estes do conjuntos de atribuições plausíveis para as variáveis. Este mecanismo é chamado de generalização de restrições. Uma restrição generalizada é imposta de forma empírica, assegurando que sobrevivem sempre um percentual de instâncias do problema acima de 20%, após a sua aplicação.

Entretanto, isto não é suficiente para permitir que o conhecimento heurístico seja aplicado eficazmente nos problemas subsequentes, por que não está claro para quais variáveis do novo problema se aplicam o re-uso destas restrições. Estritamente, a identidade de uma variável num problema de satisfação de restrições é definida pelo seu domínio (i.e., que valores estão no seu domínio) e pelas relações com outras variáveis através da rede de restrições. Isto quer dizer que uma variável numa rede de restrições é idêntica a uma outra variável em outra rede de restrições somente se estas redes forem completamente isomórficas.

Para identificar sobre quais variáveis aplicar as heurísticas para a ordenação de valores plausíveis para uma variável obtidas para um problema, ART usa uma *classificação hierárquica de variáveis*. Esta classificação é feita usando-se o método de agrupamento conceitual (“conceptual clustering technique”). Cada variável encontrada num problema é usado como uma instância de treino no algoritmo de agrupamento conceitual. O resultado final é uma hierarquia de classificação de variáveis. Na resolução de um novo problema, as semelhanças das variáveis com as variáveis encontradas no problema anterior podem ser medidas classificando as novas variáveis na hierarquia de variáveis

umentada, e assim o conhecimento acerca de uma variável pode ser associada com outras variáveis semelhantes.

Comparando com a nossa proposta de estruturação, podemos observar que este aprendizado de heurísticas para auxiliar o processo de busca de soluções está limitado pela falta de estruturação do domínio do problema. As restrições são impostas sobre variáveis e, até mesmo, a reutilização destas heurísticas “aprendidas” só pode ser feito quando a nova variável é semelhante à variável anterior, através da classificação hierárquica de variáveis. Por este motivo, esta abordagem só pode ser feita em problemas localizados num mesmo domínio de aplicação. Ao contrário, em nossa proposta, é a análise da estrutura das redes de Petri que permitem o aprendizado de heurística de busca para dirigir o processo de planejamento. Por este motivo, a metodologia pode ser reaplicada para um outro domínio qualquer de problema. Isto ocorre porque o processo de resolução opera sobre a representação do problema em redes de Petri, e não sobre a sua interpretação para um determinado domínio da aplicação.

2.5) O Uso da Abstração Estrutural no “Planning”

Em aplicações do mundo real, não queremos resolver planos lineares, porque estes promovem uma ordenação total das ações no plano que, realmente, não são necessárias. As atividades, por exemplo, em aplicações de controle são altamente paralelas. A abordagem usando níveis hierárquicos de abstração foi proposta por [Sacerdoti 1975] no sistema NOAH, descrito no item 2.2.1. Em SIPE, descrito no item 2.2.2, [Wilkins 1984], [Wilkins 1988] define o planejamento hierárquico como sendo uma eliminação de detalhes em múltiplas camadas da representação, onde cada nível de abstração tem seu próprio conjunto de predicados com seu grau específico de granularidade. Ao diminuir o grau de detalhamento, usando-se uma abstração maior, podemos representar uma região maior do espaço solução do problema em questão. O processo de planejamento, então, consiste em descer nesta hierarquia; a cada nível os detalhes faltantes são suplementados, fazendo “backtracking”, se necessário, até os níveis de abstração mais altos, até que um plano completo seja encontrado no nível “ground”.

[Mädler 1992] apresenta uma abordagem equivalente à técnica de abstração hierárquica em termos de estruturação do grafo de estados. Usando-se aprendizado indutivo é possível detectar determinados estados singulares, denominados de “needle’s eyes”. Nesta abordagem, obtém-se um agrupamento de problemas de planejamento (“planning problems cluster”) que podem ser resolvidos através de decomposição em dois subproblemas, estabelecendo-se o estado “needle’s eye” como sendo a submeta do primeiro subproblema e como sendo o estado inicial para o segundo subproblema. Esta abordagem é ilustrada usando-se o problema da Torre de Hanói para três discos, onde os tamanhos dos discos são tais que $a < b < c$.

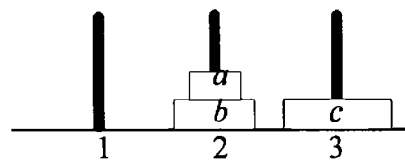


Figura 14 - A Representação do Estado (322) da Torre de Hanói

A representação dos estados do sistema é feita segundo a notação (C B A), indicando a localização dos discos c , b e a , em termos do pino 1, pino 2 ou pino 3. Se mais de um disco estão localizados num mesmo pino, assume-se que os discos estão dispostos numa ordem de acordo com o seu tamanho. Por exemplo, A Figura 14, acima, ilustra o estado (322) para o problema da Torre de Hanói.

Desde que o problema para três discos é finito, o conjunto de estados do sistema é finito e pode ser representado através do grafo de estados, ilustrado na Figura 15, seguinte. [Mädler 1992] opera sobre uma seção parcialmente ordenada do espaço de estados para o problema da Torre de Hanói. Esta região está delineada na Figura 15 pela linha tracejada e identificada por Z_{opt} . Apenas as transições de estado que estão indicadas por setas são as transições admissíveis. A aresta localizada no lado direito do triângulo ilustra o plano ótimo para o problema de mover completamente todos os discos a partir do pino 1 para o pino 3, ou seja, para o problema ((111),(333)). Entretanto, não se deseja um sistema capaz de resolver apenas este problema singular, mas, um sistema mais flexível capaz de resolver o problema a partir de qualquer estado inicial até diferentes estados metas.

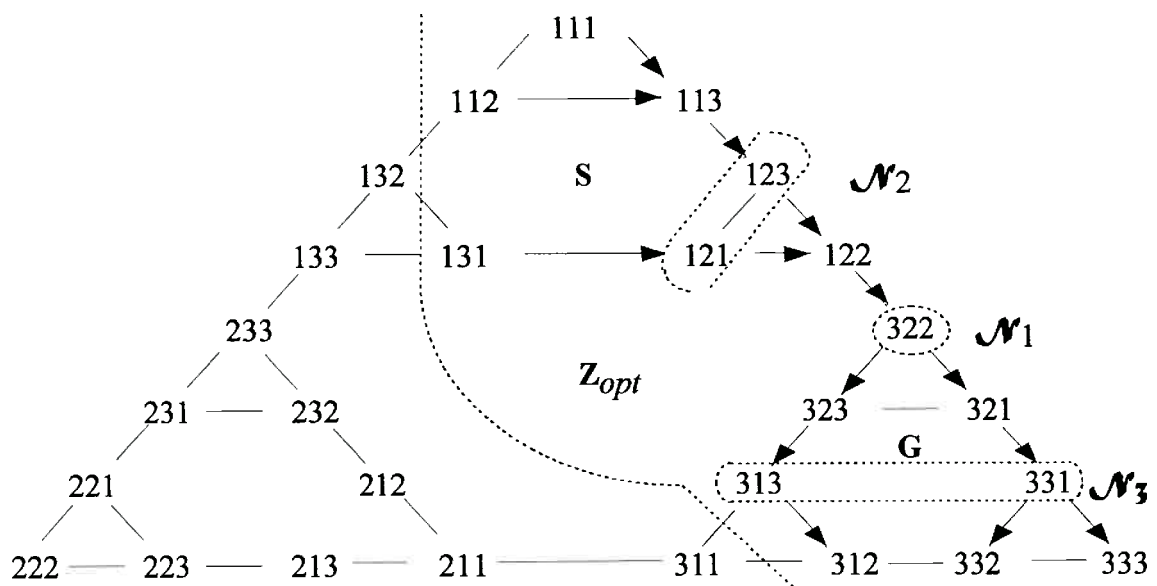


Figura 15 O Grafo de Estados para o Problema da Torre de Hanói para três discos

Para obter um sistema mais flexível, [Mädler 1992] usa a noção de “needle’s eyes” para promover a decomposição do problema em subproblemas que são resolvidos de uma maneira independente, onde a diferença para se atingir o estado meta diminui monotonicamente assim que o sistema elabora uma solução para o problema. Por exemplo, o problema $((112), (312))$ pode ser decomposto nos subproblemas $((112), (322))$ e $((322), (312))$, de maneira que o plano final é obtido pela concatenação dos planos parciais para estes dois subproblemas. Portanto, o estado (322) , indicado na Figura 15 como \mathcal{N}_1 , constitui um “needle’s eye” para este problema $((112), (322))$. Entretanto, este mesmo estado (322) não é um bom candidato para decompor o problema $((133), (311))$. Por este motivo, define-se a idéia de agrupamento maximal de problemas (“maximal cluster of problems”), ilustrado como sendo a região contendo todos estados à direita da linha tracejada na Figura 15, que podem ser decompostos usando-se o estado (322) como “needle’s eye”. Desta forma, todos os problemas (E_{Inicio}, E_{Meta}) localizados neste agrupamento maximal com o estado inicial localizado acima do estado (322) , ou seja, $E_{Inicio} \in \mathcal{S}$, e com o estado meta localizado abaixo do estado (322) , ou seja, $E_{Meta} \in \mathcal{G}$, tem as suas soluções ótimas indicadas por setas na Figura 15.

O agrupamento de problemas escolhido tem uma propriedade notável: o conjunto Z_{opt} de todos os estados que ocorrem em quaisquer soluções ótimas estão ordenados parcialmente como mostram as setas na Figura 15. Esta ordem parcial, denotada a partir deste momento por \prec , sugere uma nova maneira de gerar abstrações de operadores que preservam a otimalidade a partir de subgrafos do espaço de estados do sistema.

O Operador *decomp*(*EI,EM*) em \mathcal{N}_1 :

Para definir o operador abstrato genérico a partir do grafo estrutural de estados no estado (322), são introduzidos dois predicados: P_S para descrever o conjunto inicial $S := \{ EInicio \mid EInicio \in Z_{opt}, EInicio \prec (322) \}$, e o predicado P_G para descrever o conjunto meta $G := \{ EMeta \mid EMeta \in Z_{opt}, (322) \preceq EMeta \}$. Então, o operador seguinte decompõe cada problema ($EInicio, EMeta$) com $EInicio \in S$ e $EMeta \in G$ nos subproblemas ($EInicio, (322)$) e $((322), EMeta)$.

decomp(*EInicio,EMeta*) em \mathcal{N}_1 :

pré-condição: P_S
 pós-condição: P_G
 decomp_em: (322)

Para obter as soluções desejadas, basta resolver os dois subproblemas acima no nível abaixo de abstração e, em seguida, concatenar os seus resultados.

Vamos ilustrar o processo de planejamento hierárquico, usando este processo de decomposição de problemas usando “needles’s eyes”. Seja o problema $((1,1,2),(3,1,2))$. Podemos observar que este problema pertence à *constelação estrutural de estados* (“structural constellation of states”) Z_{opt} , ou seja, o estado inicial $(1,1,2) \in S$ e o estado meta $(3,1,2) \in G$, portanto, este problema pode ser decomposto usando o “needle eye” $(3,2,2)$.

Este problema pode apresenta três níveis de abstração. O nível 0 ou (“ground”) que define a localização de todos os discos (C B A). O nível 1 de abstração, que define

apenas a localização dos discos c e b , i.e. $(C B _)$. O nível 2 de abstração, que define apenas a localização do disco c . Em cada um destes níveis, são estabelecidas as metas dos subproblemas, considerando-se o estado que atua como “needle’s eye” e o nível de detalhamento daquele nível de abstração.

Considere o problema $((1,1,2),(3,1,2))$, cujo processo de resolução hierárquica está ilustrado na Figura 16, seguinte.

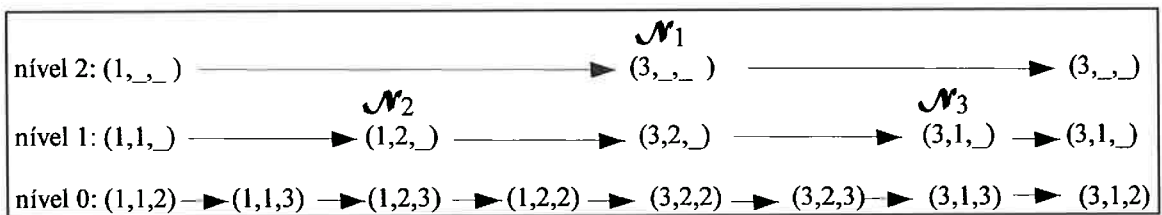


Figura 16 - Uma Solução Hierárquica para o Problema $((1,1,2), (3,1,2))$

No nível 2 de abstração o sistema irá decompor o problema em dois subproblemas, estabelecendo o “needle’s eye” $(3,2,2)$, indicado na Figura 15 por \mathcal{N}_1 , como meta intermediária. Mas, para não sobre-especificar o problema, este é definido apenas como $(3, _ , _)$ porque o sistema tem interesse apenas na localização do disco c . No nível de abstração 1, o sistema tem de estabelecer metas para obter o detalhamento da localização dos discos c e b . Para isto, o \mathcal{N}_1 é definido como $(3,2, _)$. Além disso, é preciso definir \mathcal{N}_2 como $(1,2, _)$ e definir \mathcal{N}_3 como $(3,1, _)$. Finalmente, no nível 0, o sistema tem de estabelecer metas para obter o detalhamento da localização de todos os discos, c , b e a . Portanto, o \mathcal{N}_1 é definido como $(3,2,2)$, o \mathcal{N}_2 é definido como $(1,2,3)$ e \mathcal{N}_3 é definido como $(3,1,3)$. O Plano detalhado ótimo para o nível ground está ilustrado na ultima linha da Figura 16.

Como aplicação desta abordagem, [Mädler 1992] descreve um sistema para o problema de controle de processos, denominado projeto SOLEIL. No desenvolvimento deste sistema foi constatado que estes “needle’s eyes” realmente existem nos processos do mundo real. Um “design” racional assim como as considerações de custo e, também, as leis da natureza organizam o grafo de estados, de uma maneira que estes estados sin-

gulares podem ser identificados. [Mädler 1992] relata uma modelagem de um dispositivo de deposição de plasma efetuado como parte do projeto SOLEIL.

Esta abordagem é chamada de abstração estrutural porque, essencialmente, usa subestruturas do grafo de estados e as propriedades puramente estruturais que podem ser obtidas a partir delas, as quais são utilizadas para aprender acerca das condições e expansões dos operadores.

Comparando com a nossa proposta de estruturação do problema de “planning” usando IA e redes de Petri, podemos dizer que esta abordagem se aproxima da nossa proposta na medida em que utiliza a análise das propriedades estruturais da representação em grafos do problema. A decomposição proposta busca o estabelecimento subproblemas que se localizam em regiões independentes do grafo de estado, de modo que podem ser resolvidos de uma forma independente, sem problemas. Desta forma, uma mesma estratégia de busca da solução pode progredir monotonicamente na direção da solução do problema original. Além disso, o sistema utiliza planos não-lineares, em que os detalhes são acrescentados aos planos à medida que estes se aproximam do nível “ground” de abstração.

O que diferencia nossa proposta é o fato de utilizarmos o mesmo formalismo da rede de Petri tanto para representar o domínio da aplicação assim como o problema de “planning” em si. [Mädler 1992] representa apenas o domínio da aplicação. Além disso, SOLEIL utiliza apenas subestruturas do grafo de estados, chamadas de constelações estruturais de estados, e promove uma análise das propriedades estruturais destas. Ao contrário, em nossa proposta, fazemos a modelagem de todos os estados plausíveis de um sistema em rede de Petri e fazemos a análise das propriedades estruturais destas, usando esta teoria.

2.6) O Uso de Redes de Petri em IA “Planning”

Com o propósito de obter a estruturação do problema de “planning”, as redes de Petri tem sido utilizadas em diversos sistemas. Como uma alternativa às redes procedimentais utilizadas em NOAH para representar planos não lineares, [Drummond 1985] usou a representação baseada numa rede de Petri C/E estendida. [Rillo 1988] desenvolveu um sistema inteligente aplicado para o controle de sistemas que utiliza uma representação baseada em diferentes tipos de redes de Petri, onde a identificação da regra mais aplicável num determinado momento é feita através de um jogador de marcas (“token player”). [Paiva 1993] descreve um sistema de planejamento aplicado para uma célula de montagem onde um plano hierárquico é elaborado na forma de redes de Petri, e suas ações são executadas diretamente pelo módulo de execução do plano. [Hendler 1992] apresenta uma arquitetura para o planejamento e monitoração de sistema dinâmicos que emprega redes de Petri para a representação do conhecimento.

2.6.1) O Plano representado como uma Rede de Petri

A rede procedimental, tal como foi definida em [Sacerdoti 1975] e descrita no item 2.2.1, é “uma rede de ações em diferentes níveis de detalhamento, estruturados numa hierarquia de seqüências (de ações) parcialmente ordenadas no tempo”. Uma rede procedimental pode ser desenhada como sendo um grafo onde nós representam ações e com arcos direcionados conectando estes nós. O arco direcionado denota uma ordem parcial, de tal modo que uma seta indo de um nó α até um nó β indica que a ação α ocorre antes da ação β . [Drummond 1985] faz uma crítica à esta representação usando redes procedimentais, porque, devido à interpretação de ordem parcial estrita dos seus arcos, esta não é capaz de representar processos iterativos. Por este motivo, propôs o uso da rede do plano (“plan net”), baseada em rede de Petri.

A rede do plano é definida pela 6-upla $\langle P, T, R_a, R_b, R_c, R_e \rangle$, onde $P = \{p_1, p_2, \dots, p_N\}$ é um conjunto finito de lugares; $T = \{t_1, t_2, \dots, t_M\}$ é um conjunto finito de

transições; $N \geq 0$, $M \geq 0$; e $P \cap T = \emptyset$, o conjunto vazio. $R_a = \{ (t_i, t_k) \mid \exists p_j \in P [(t_i, p_j) \in R_c \wedge (p_j, t_k) \in R_e] \}$, a relação de atingibilidade (“the allow relation”); $R_b \subseteq (TxT)$ a relação de ordem parcial (“the before relation”), onde R_b deve ser uma relação de ordem parcial estrita sobre T ; $R_c \subseteq (TxP)$, a relação causal (“the cause relation”); e, finalmente, $R_e \subseteq (PxT)$, a relação de habilitação (“the enable relation”).

Um lugar p_i é um lugar de entrada de uma transição t_j se e somente se $(p_i, t_j) \in R_e$, a relação de habilitação. Um lugar p_j é um lugar de saída de uma transição t_j se e somente se $(p_j, t_j) \in R_c$, a relação causal. Uma marcação da rede de planos é o mapeamento $M: P \rightarrow \{0, 1\}$. Uma transição t_j está habilitada se para cada um dos seus lugares de entrada, p_i , a marcação $M(p_i) = 1$. Uma transição t_j pode ser disparada se estiver habilitada. O disparo de uma transição t_j , a partir de uma marcação M , produz uma nova marcação M' , tal que:

- (i) se p_i é um lugar de entrada de t_j , então $M'(p_i) = 0$;
- (ii) se p_j é um lugar de saída de t_j , então $M'(p_j) = 1$;
- (iii) se p_i não é um lugar de entrada nem de saída de t_j , então $M'(p_i) = M(p_i)$.

Esta rede apresenta as relações R_a e R_b , do tipo (TxT) , como extensões da rede de Petri do tipo C/E. A rede de Petri é um grafo bipartido em que são admitidas apenas as relações do tipo (PxT) e do tipo (TxP) . [Drummond 1985] apresenta dois exemplos de representação do plano usando esta rede estendida.

Exemplo 1: O Mundo de Blocos

A seguir, estão descritas as cláusulas que definem o plano para o Mundo de Blocos. Uma marcação inicial da rede é incluída. A Figura 17, seguinte, ilustra a rede estendida que corresponde à estas cláusulas.

$$C = \langle P, T, R_a, R_b, R_c, R_e \rangle$$

$$P = \{ (on\ c\ a), (on\ c\ tb3), (on\ a\ tb1), (on\ b\ tb2), (on\ b\ c), (on\ a\ b), (clear\ tb3), (clear\ a), (clear\ tb1), (clear\ b), (clear\ tb2), (clear\ c) \}$$

$$T = \{ \text{tr1}(\text{move } c \text{ a } tb3), \text{tr2}(\text{move } b \text{ tb2 } c), \text{tr3}(\text{move } a \text{ tb1 } b) \}$$

$$R_c = \{ ((\text{move } c \text{ a } tb3), (\text{clear } a)), ((\text{move } c \text{ a } tb3), (\text{on } c \text{ tb3})), \\ ((\text{move } c \text{ a } tb3), (\text{clear } c)), ((\text{move } b \text{ tb2 } c), (\text{clear } b)), \\ ((\text{move } b \text{ tb2 } c), (\text{on } b \text{ c})), ((\text{move } b \text{ tb2 } c), (\text{clear } tb2)), \\ ((\text{move } a \text{ tb1 } b), (\text{clear } a)), ((\text{move } a \text{ tb1 } b), (\text{clear } tb1)), \\ ((\text{move } a \text{ tb1 } b), (\text{on } a \text{ b})) \}$$

$$R_e = \{ ((\text{on } c \text{ a}), (\text{move } c \text{ a } tb3)), ((\text{clear } tb3), (\text{move } c \text{ a } tb3)), \\ ((\text{clear } c), (\text{move } c \text{ a } tb3)), ((\text{clear } c), (\text{move } b \text{ tb2 } c)), \\ ((\text{clear } b), (\text{move } b \text{ tb2 } c)), ((\text{on } b \text{ tb2}), (\text{move } b \text{ tb2 } c)), \\ ((\text{clear } a), (\text{move } a \text{ tb1 } b)), ((\text{clear } b), (\text{move } a \text{ tb1 } b)), \\ ((\text{on } a \text{ tb1}), (\text{move } a \text{ tb1 } b)) \}$$

$$R_a = \{ (((\text{move } c \text{ a } tb3), (\text{move } b \text{ tb2 } c)), ((\text{move } b \text{ tb2 } c), (\text{move } a \text{ tb1 } b)), \\ ((\text{move } c \text{ a } tb3), (\text{move } a \text{ tb1 } b)), ((\text{move } c \text{ a } tb3), (\text{move } c \text{ a } tb3)), \\ ((\text{move } b \text{ tb2 } c), (\text{move } b \text{ tb2 } c)), ((\text{move } a \text{ tb1 } b), (\text{move } a \text{ tb1 } b)) \}$$

$$R_b = \{ ((\text{move } c \text{ a } tb3), (\text{move } b \text{ tb2 } c)), ((\text{move } b \text{ tb2 } c), (\text{move } a \text{ tb1 } b)) \}$$

$$M(p) = 1 \text{ se } p \in \{ (\text{on } c \text{ a}), (\text{clear } tb3), (\text{clear } a), \\ (\text{clear } b), (\text{on } b \text{ tb2}), (\text{on } a \text{ tb1}) \}$$

$$M(p) = 0 \text{ caso contrário}$$

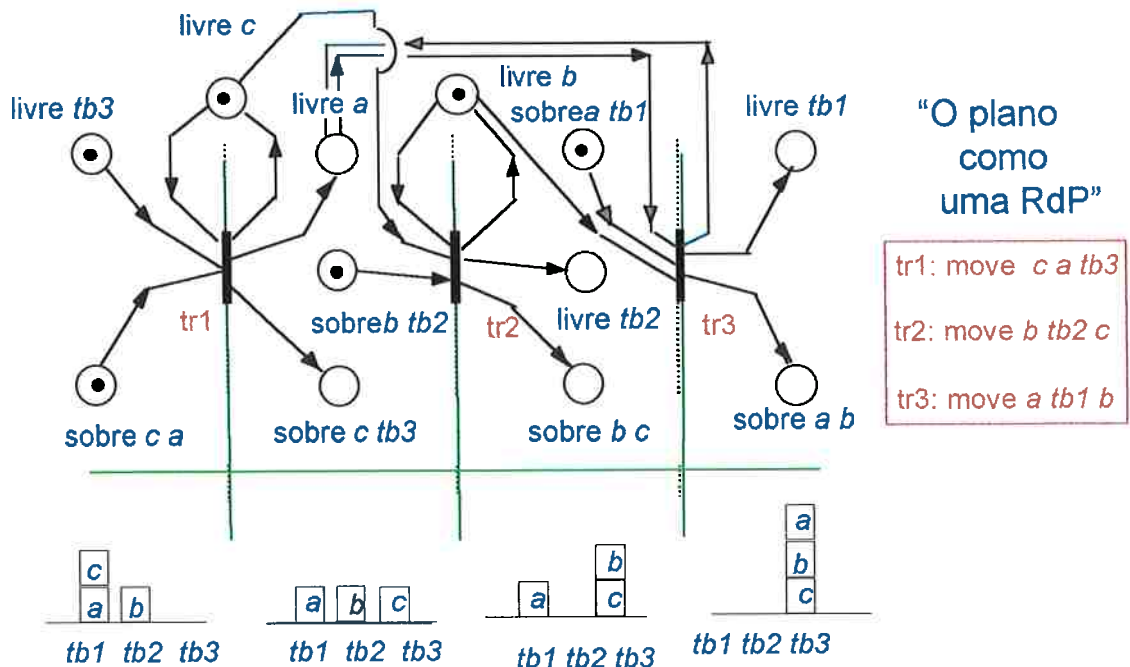


Figura 17 A Rede de Petri estendida para Mundo de Blocos

É possível se pensar na representação os estados mais sintética, por exemplo, usando-se um número menor de lugares. Observe que apesar de os estados de alguns lugares serem deduzidos como consequência do disparo de transições, estes não são mais utilizados posteriormente pelo jogador de marcas para encontrar a solução do problema. A Figura 17 mistura, de uma maneira “flat”, o conhecimento do domínio (ou domain-K) e o plano. Note que, afinal, resta apenas o interesse neste último.

Na parte inferior da Figura 17, ilustramos os estados do sistema que correspondem à marcação da rede do plano. A relação estendida $R_b = \{ tr1 < tr2 < tr3 \}$ equivale à regressão de metas de [Waldinger 1975], e à crítica para eliminar interferências nocivas entre ações “paralelas”, descritas em [Sacerdoti 1975].

[Drummond 1985] apresenta um outro exemplo no qual mostra que esta representação pode modelar sistemas iterativos, i.e., contendo ciclos. O exemplo modela um sistema simples que tem apenas um martelo que muda de estado entre os estados “up” e “down”, indefinidamente. O autor considera esta uma vantagem deste tipo de representação em relação às redes procedimentais.

Comparando com a nossa proposta de estruturação do problema de “planning”, [Drummond 1985] faz apenas a representação do plano em rede de Petri, não usando esta para representar o domínio da aplicação. Por exemplo, o exemplo 1 descreve apenas três tipos de ações, enquanto o conjunto de ações plausíveis no Mundo de Blocos é muito maior. Além disso, ele introduz extensões à teoria da RdP, usando relações do tipo TxT, não suportadas inicialmente nesta teoria.

2.6.2) A Rede de Petri Aplicada ao Controle Inteligente

[Rillo 1988] apresenta uma metodologia baseada em redes de Petri (e regras) como uma linguagem única (de representação) do projeto para o desenvolvimento de sistemas de controle para sistemas dinâmicos a eventos discretos (SDEDs), a partir das especificações funcionais até a implementação final. Desta forma, obtém-se uma homogeneização da linguagem de representação em todas as fases do projeto, ou seja, para

especificar, desenvolver, implementar e validar um sistema de controle. Esta padronização possibilita uma drástica diminuição nos erros que surgem na passagem de uma fase do projeto para a fase seguinte, sendo excelente quando o sistema modelado está sujeito a constantes alterações, como é o caso de sistemas de controle para SDEDs.

A utilização combinada de redes de Petri e regras facilita a tarefa de especificação das tarefas que o sistema deve realizar, incorporando as qualidades dos dois métodos. Isto se deve, por um lado, ao fato de que as RdPs são excelentes para serem utilizadas na fase de especificação de sistemas de controle para SDEDs, para representar os seus aspectos procedimentais. Por outro lado, se deve ao fato de que é extremamente simples expressar conhecimento de uma forma declarativa através de regras. Além disso, as redes de Petri podem, adicionalmente, funcionar exercendo o papel do mecanismo de inferência para determinar quais regras devem ser aplicadas a cada momento.

[Rillo 1988] desenvolveu o sistema PETRIX, que foi implementado no dialeto SCHEME, da Texas Instruments, para a linguagem LISP. Esta aplicação consiste de um *núcleo de sistema de controle* para ser utilizado nos níveis de controle de células de manufatura flexível, de estações de trabalho e de controle local. Este sistema apresenta, basicamente, duas partes: um módulo de interface para E/S e o programa jogador de marcas (“token player”). O programa jogador de marcas é que faz o controle propriamente dito, e opera ciclicamente efetuando os seguintes passos:

- faz a leitura das entradas,
- opera sobre as redes de Petri e regras,
- aciona as saídas.

O sistema PETRIX é bastante flexível quanto ao formato da entrada de dados, apresentando módulos para processar entradas expressas nos seguintes formatos:

- 1) REGRAS: este módulo suporta entrada apenas na forma de regras.
- 2) GRAFCET: este módulo suporta entrada na forma de Grafos de Comando Etapa-Transição (GRAFCET). O GRAFCET é baseado na RdP C/E.

- 3) RdP C/E: este módulo suporta entradas na forma de RdP C/E e regras.
- 4) RdP P/T: este módulo suporta entradas na forma de RdP P/T e regras.
- 5) PETRIX: este módulo suporta apenas redes com marcas individuais e regras. As redes de marcas individuais foram introduzidas com as redes Predicado/Transição. As redes de Petri Coloridas, baseadas na redes Predicado/Transição, representam outro tipo de rede com marcas individuais.

Comparando com a nossa proposta de estruturação, [Rillo 1988] utiliza uma representação que combina as redes de Petri e IA para suportar todas as fases do desenvolvimento de um sistema. Esta forma de representação híbrida tem a vantagem de somar as qualidades apresentadas pelos dois métodos. A diferença está em que o sistema PETRIX está direcionado para aplicações de controle de sistemas de manufatura flexível, enquanto que a nossa proposta direciona os resultados para o problema de “planning” e de “scheduling”.

2.6.3) O Planejamento usando Redes de Petri

[Paiva 1993] descreve o sistema de planejamento de atividades hierárquico, independente do domínio, e que permite a elaboração de planos não lineares, parcialmente ordenados, denominado de Planejador de Atividades de uma Célula de Montagem - PACEM. O PACEM é um sistema de planejamento que gera planos, que são descritos através de redes de Petri, e que são executáveis por um sistema de controle. Esta representação do plano usando RdP, permite uma boa interface entre os sistemas de elaboração e de execução do plano. O PACEM foi desenvolvido utilizando-se a linguagem LISP, em estações de trabalho SUN.

Para cada nível de abstração, o sistema elabora um plano naquele nível de detalhe. Para fazer a representação do plano em redes de Petri, utiliza estruturas de dados, chamados de “redes de processo”. A noção de processo foi inspirada nos “plots” usados pelo SIPE [Wilkins 1988] e nas “Knowledge Areas (KA s)” [Georgeff et Lansky 1986].

O atributo “plot” de um operador SIPE, especifica como uma ação pode ser realizada, passo-a-passo, em termos de outras ações e metas. No SIPE, estas outras ações são representadas através de redes procedimentais [Sacerdoti 1975]. Por outro lado, Knowledge Areas (KAs), são especificações de procedimentos declarativos. KAs descrevem como atingir com sucesso determinadas metas ou como reagir em determinadas situações. KAs são constituídos por duas partes: corpo e condição de invocação. O corpo de uma KA descreve os passos do procedimento, e pode ser representado graficamente por uma rede, do mesmo modo que o “plot” de um operador SIPE.

Uma rede de processos é uma estrutura de dados composta de processos, onde os nós meta podem ser substituídos por outros processos ou, então, por outras metas concorrentes quando forem expandidos para um nível de abstração mais baixo. Um processo descreve metas, algumas redes de Petri e alguns testes. O processo descreve também a ordem temporal das metas, redes de Petri e testes. Um processo é uma estrutura de dados cujos nós podem ser dos seguintes tipos:

- *início-processo*: nó que indica o início do processo.
- *fim-processo*: nó que indica o fim do processo.
- *meta*: nó que contém uma meta a ser satisfeita. As metas devem ser realizadas na ordem em que os nós aos quais elas estão associadas aparecem no processo.
- *separação*: nó que indica o início de uma conjunção de metas, para as quais não se definiu nenhuma ordem parcial (ou metas paralelas).
- *união*: nó que indica o término de metas paralelas.
- *rede de Petri*: nó que indica uma ação descrita através de uma rede de Petri, e que deve ser executada pelo sistema de controle.
- *teste*: nó que indica uma condição que deve ser verificada no mundo real através de sensores.

Um processo que não tem nenhuma meta associada, e é composto apenas por ações (primitivas) descritas através de uma rede de Petri é denominado de “processo pré-satisfeito”. Isto é, para satisfazer este tipo de processo, basta o módulo de execução do plano executar a rede de Petri. Por outro lado, um processo que é composto por metas

(e não por ações) é denominado de “processo pré-condição”. Para satisfazer este tipo de processo, o sistema deve tornar as suas metas verdadeiras. O conjunto destes dois tipos de processos foi denominado de *processos alternativos* (PAs).

O processo alternativo é definido pelos seguintes atributos:

- *variáveis*: lista de nomes de variáveis utilizadas pelo PA e a especificação de restrições sobre os tipos de valores que elas podem assumir.
- *propósito*: cláusula a ser concluída como sendo verdadeira após a execução deste PA.
- *recursos*: recursos utilizados pelo PA. O PACEM representa recursos como no SIPE, descrito no item 2.2.2.
- *pré-condição de invocação*: cláusulas que devem ser verdadeiras no modelo de mundo analisado para que um processo pré-satisfeito possa ser aplicado.
- *processo pré-satisfeito*: processo pré-satisfeito que deve ser aplicado quando a pré-condição do PA for verdadeira.
- *processo pré-condição*: processo pré-condição que deve ser aplicado quando a pré-condição do PA for verdadeira
- *lista de adição* (ou efeito): sentenças que serão verdadeiras no modelo de mundo depois da execução do PA, e que devem ser incluídas no modelo de mundo em que o PA for aplicado.
- *lista de eliminação* (ou efeito-oposto): sentenças que não serão mais verdadeiras no modelo de mundo depois da execução do PA, e que devem ser retiradas do modelo de mundo em que o PA for aplicado.

O processo pré-satisfeito e o processo pré-condição têm em comum os atributos: variáveis, recursos, propósito e lista de adição.

O PACEM não procura tornar verdadeiras as pré-condições dos PAs, ou seja, não as estabelece como submetas como fazem alguns sistemas. Muitas vezes, esta estratégia faz com que o sistema se afaste ainda mais da meta desejada. Ao invés, procura tornar

verdadeiras as metas da rede de processos, através da escolha de PAs que descrevem estas metas no seu atributo propósito.

O PACEM apresenta um mecanismo de dedução causal, dependente do contexto, que pode deduzir novos fatos a partir do modelo de mundo. As relações causais, desta forma, são separadas da descrição dos operadores, tornando-as mais enxutas. O mecanismo de dedução causal, ao acessar diferentes estados do mundo, pode reagir às mudanças ocorridas entre estes dois estados, permitindo, desta forma, que os efeitos do que ocorreu durante a execução de uma ação possam ser deduzidas. As regras dedutivas causais contêm os seguintes atributos:

- *variáveis*: lista de nomes de variáveis utilizadas pela regra e a especificação de restrições sobre os tipos de valores que elas podem assumir.
- *gatilho*: condição necessária para que uma regra dedutiva causal seja analisada pelo mecanismo de dedução causal. Uma regra dedutiva causal pode ser disparada em consequência da execução de uma ação ou de outra regra dedutiva causal.
- *pré-condição de invocação*: conjunto de cláusulas que devem ser verdadeiras no modelo de mundo anterior ao modelo de mundo que se está analisando.
- *condição*: conjunto de cláusulas que devem ser verdadeiras no modelo de mundo que se está analisando.
- *lista de adição* (ou efeito): sentenças que devem ser incluídas no modelo de mundo quando forem satisfeitos o gatilho, a pré-condição e a condição da regra causal.
- *lista de eliminação* (ou efeito-oposto): sentenças que devem ser retiradas do modelo de mundo quando a regra causal for aplicada.

Comparando com a nossa proposta de estruturação do problema, o PACEM faz a representação do plano em redes de Petri, mas, não faz a representação do domínio do problema. A rede de Petri é usada para representar “processos”, inspirado na noção de “plot” usada no SIPE [Wilkins 1988] e “knowledge Areas” [Georgeff et Lansky 1986]. A nossa metodologia faz a representação do problema do “planning” em si e do domínio do problema usando rede de Petri e IA. Além disso, o PACEM ainda usa uma representação baseada no sistema SIPE, descrito no item 2.2.2. E, portanto, ainda assume a hi-

pótese da representação STRIPS, usando o mesmo mecanismo do tipo adição/eliminação.

O módulo de execução do PACEM pode executar diretamente o plano representado em rede de Petri. A nossa proposta de metodologia pode ser estendida para suportar a representação de problemas de “planning” e do domínio da aplicação, usando Redes de Petri de alto nível, para o desenvolvimento de sistema de planejamento que usam planos parciais que são detalhados em tempo de execução.

2.6.4) A Rede de Petri em Sistemas Emergenciais

[Hendler 1992] apresenta uma arquitetura com múltiplas camadas de abstração, para o planejamento e controle de sistema dinâmicos que emprega a rede de Petri para fazer a representação do conhecimento. Nesta arquitetura, está implícita a idéia de urgência ou de emergência (“supervenieny”), ou seja, durante a execução do plano corrente, tarefas novas podem surgir em decorrência das informações percebidas pelos sensores. Dependendo da urgência destas, a tarefa corrente é interrompida para a realização da tarefa nova. Ao término desta, a tarefa que foi interrompida pode ser retomada.

[Hendler 1992] chama a atenção para uma diferença sutil que existe entre esta idéia de “emergência” e os sistemas reativos, porque nestes o comportamento reativo é dirigido pela ocorrência de uma falha, enquanto que nos sistemas “emergenciais”, geralmente, esta falha não ocorre. Uma situação de emergência pode ocorrer em decorrência do aparecimento de uma tarefa nova que requer uma atuação urgente ou até mesmo para livrar o sistema de uma situação de risco potencial.

A arquitetura “supervenient” (ou seja, dirigida por interrupções) está baseada no modelo dos sistemas “blackboard” com múltiplos níveis. Ela consiste de diversos níveis de dados/processamento arranjados numa ordem parcial. Cada nível comunica-se apenas

com aqueles níveis imediatamente acima ou abaixo dela na hierarquia. Só o nível mais baixo (i.e., os elementos minimais da ordem parcial) têm acesso direto aos sensores e aos atuadores. A comunicação entre os níveis é facilitada através de sistemas de tradução localizados abaixo de cada nível (exceto dos níveis no fundo da hierarquia). O sistema de tradução passa metas para baixo e conhecimento dos estados do mundo para cima. Os níveis mais altos podem prover metas, mas a determinação da ação apropriada de baixo nível a ser efetuada para atingir tais metas é uma prerrogativa do nível mais baixo. Em suma, tem-se metas para baixo e conhecimento do mundo para cima (“goals down, world knowledge up”). Esta arquitetura está aberta quanto ao tipo de mecanismo de inferência utilizado em cada camada de abstração, porque está baseado num mecanismo do tipo “blackboard”, com protocolo uniforme. Por exemplo, pode-se adotar redes neurais na camada mais baixa, técnica de programação lógica na camada intermediária e sistemas de planejamento como o NONLIN [Tate 1977] no seu nível mais alto de abstração.

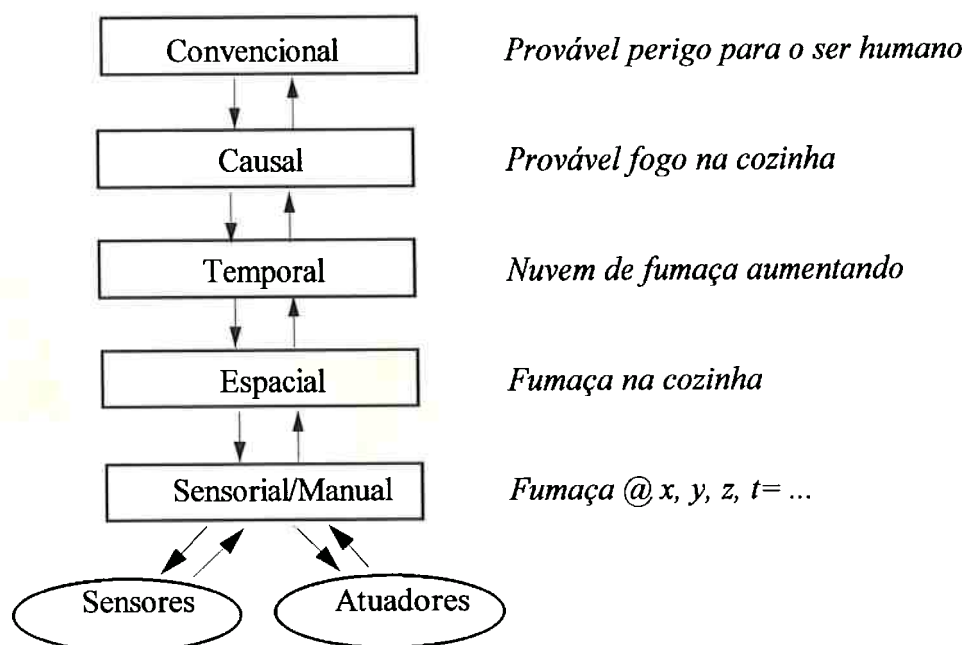


Figura 18 - Os níveis de abstração do APE, frente ao evento fumaça na cozinha

[Hendler 1992] desenvolveu um sistema baseado nesta proposta de arquitetura do tipo “emergencial”, denominado de Avaliador do Sistema Particionado-por-Abstração (“Abstraction-Partitioned Evaluator” - APE). Os níveis de abstração no APE, a partir do nível mais baixo para o mais alto são: sensorial/manual, espacial, temporal, causal e convencional. A Figura 18, acima, ilustra estes níveis de abstração, para um exemplo de detectar fumaça na cozinha.

Cada nível de abstração do APE, apresenta um conhecimento especializado acerca do estado do mundo, segundo uma determinada dimensão. Para cada nível de abstração existe uma estrutura independente composta por: um conjunto de operadores, um sistema blackboard e um sistema de tradução. Cada nível de abstração possui suas próprias metas e elabora um plano específico para a dimensão do conhecimento especializado que representa. É o somatório dos efeitos produzidos pela execução da conjunção destes planos especializados que produz o comportamento global desejado do sistema controlado, por exemplo, de um robô.

Para um determinado nível de abstração, visto isoladamente, a arquitetura do APE se assemelha à arquitetura utilizada no PRS [Georgeff et Lansky 1987], inclusive os operadores apresentam atributos comuns, tais como, *gatilho* e *pré-condição de invocação*. Além de operadores, a Base de Conhecimentos em cada nível de abstração apresenta ainda: *demons* e dois tipos de metas. Os *demons* são pequenos pedaços de código que podem ser colocados no blackboard para observar a ocorrência de mudanças no estado do sistema, e para disparar sempre que isto ocorre. O uso de *demons* permite a especificação intuitiva de monitores. Os tipos de metas são: metas *teleológicas*, i.e., metas de atingimento de estados desejados, e metas *teleoepistêmicas*, i.e., metas para o atingimento de conhecimento. Estas visam atualizar o conjunto de crenças correntes do sistema, adquirindo mais conhecimento.

Os operadores APE se assemelham aos "knowledge sources" utilizados nos sistemas "blackboard" tradicionais. Eles ainda mantêm as mesmas características dos operadores tradicionais, utilizando os mecanismos do tipo listas de adição e de eliminação, característicos da representação STRIPS. Os aspectos procedimentais dos operadores estão especificados de forma semelhante às redes de Petri do tipo C/E, como está descrito em [Drummond 1985] e [Paiva 1993].

A seguir, ilustramos um operador APE *reflexo_de_dor*, usado numa aplicação do APE para controlar um robô doméstico, denominado HOMEBOT.

```
(defoperador: reflexo_de_dor
  : nível : sensorial_manual
  : propósito (eliminar (dolorido : ?))
  : filtros ((igual ?1 :?))
  : rede de Petri
    : passos (( REFLEXO_DOR_NA_MÃO
      ( dúvida ` (dolorido mão ) )
      ( eliminar ` (dolorido mão ) : prioridade : urgente
      : call_back
      # ' (lambda ( ) ( término REFLEXO_DOR_NA_MÃO )))
      ( REFLEXO_DOR_DE_CABEÇA
      ( dúvida ` (dolorido cabeça ) )
      ( eliminar ` (dolorido cabeça ) : prioridade : urgente
      : call_back
      # ' (lambda ( ) ( término REFLEXO_DOR_DE_CABEÇA))))
    : inicializa (REFLEXO_DOR_NA_MÃO
      REFLEXO_DOR_DE_CABEÇA)
    : transições ((REFLEXO_DOR_NA_MÃO:>
      REFLEXO_DOR_NA_MÃO)
      (REFLEXO_DOR_DE_CABEÇA:>
      REFLEXO_DOR_DE_CABEÇA)))
  : no-sucesso ( eliminado (dolorido :?))
```

Um operador é definido unicamente por: um *nível* de abstração, um *nome* simbólico, uma lista de *variáveis*, um *propósito*, uma *condição de invocação*, *filtros* (lista de

variáveis a serem verificados quando da instanciação do operador), uma especificação em *rede de Petri* dos aspectos procedimentais do operador e uma lista de comandos a serem executados em determinadas situações (*na-falha*, *no-sucesso*, *na-suspensão*, *na-retomada*, *no-término*). Na situação *no-sucesso*, estes comandos correspondem às tradicionais listas de adição/eliminação dos operadores de “planning” do tipo STRIPS.

O núcleo do operador -- a descrição do que ele faz -- é descrito pela rede de Petri do tipo C/E. A rede de Petri é especificada em *passos*, argumentos *inicializa* e *transições*. [Hendler 1992] usa o exemplo de uso do operador *reflexo_de_dor*, acima, para mostrar a capacidade reativa da camada mais baixa de abstração do APE, implementado no sistema HOMEBOT. Inicialmente, o robô pode ter uma conjunção de metas, tais como (obter ‘ (*prazer*)). Entretanto, o robô pode receber uma meta emergencial do tipo (eliminar ‘ (*dolorido* :?), que é estabelecido no nível sensorial/manual. Esta meta corresponde ao *propósito* do operador *reflexo_de_dor*, que o APE instancia para satisfazer esta meta emergencial.

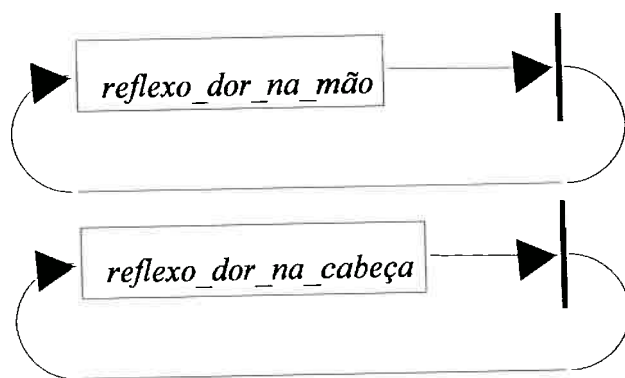


Figura 19 - A rede de Petri para o operador *reflexo_de_dor*

A Figura 19 *⁴, acima, ilustra o operador *reflexo_de_dor*. Este apresenta dois *passos*, cada um dos quais é especializado para tratar um tipo de reflexo de dor. O passo

*⁴ Observe que, por uma questão de preferência, [Hendler 1992] usa os lugares para representar ações, ao invés de utilizar as transições, como fazemos em nosso trabalho.

REFLEXO_DOR_NA_MÃO primeiro emite um comando para descobrir o estado de dor na mão. Então, estabelece a meta “eliminar (dolorido mão)” com a prioridade: *urgente*, e acrescenta um demon a esta meta que irá concluir o passo REFLEXO_DOR_NA_MÃO quando (dolorido mão) for eliminado. A primeira cláusula do argumento *transições* cria um laço na rede de Petri que o passo REFLEXO_DOR_NA_MÃO é re-executado cada vez que é terminado.

A meta “eliminar (dolorido mão)”, estabelecido pelo operador *reflexo_de_dor*, dispara uma instancia do operador *reflexo_de_dor_na_mão*. A Figura 20*⁵, a seguinte, ilustra a rede de Petri para este operador. Esta apresenta oito *passos*, que podem ser executados em paralelo.

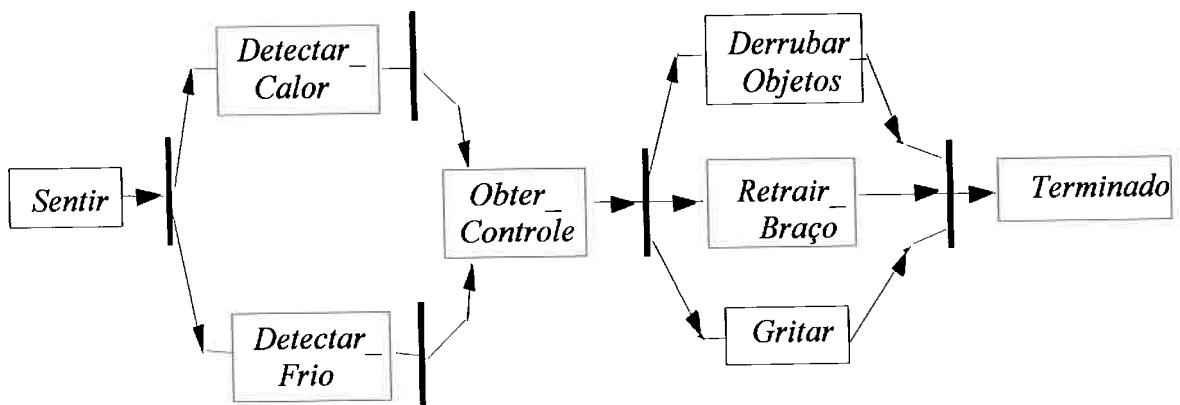


Figura 20 - A rede de Petri para o operador *reflexo_de_dor_na_mão*

Comparando com a nossa proposta de estruturação do problema, [Hendler 1992], não usa redes de Petri para fazer a representação do domínio do problema. A rede de Petri é usada para representar aspectos procedimentais dos operadores, de uma forma semelhante a idéia de “processos” utilizada no PACEM [Paiva 1993], inspirado na noção de “plot” usada no SIPE [Wilkins 1988] e “knowledge areas” [Georgeff et Lansky 1987]. Ao contrário, a nossa metodologia faz a representação do problema do

*⁵ Observe que, ao invés da notação tradicional, HomeBot representa as ações através dos boxes. Entretanto, existe um erro nesta rede porque as ações *detectar-frio* e *detectar-calor* deveriam ser alternativas de escolha e que nem sempre são ações paralelas.

“planning” em si e do domínio do problema usando rede de Petri e IA. Ao contrário, em nossa proposta, a implementação preserva a estrutura da rede de Petri e todos os seus elementos: lugares, transições, arcos, ao invés de utilizarmos um “operador” baseado na representação do tipo STRIPS.

Concluindo, podemos afirmar que a nossa proposta procura fazer uma estruturação do problema de “planning”, pelos mesmos motivos que foram descritos porque os sistemas de planejamento baseados em IA têm buscado esta estruturação. A nossa proposta está baseada em IA e no formalismo das redes de Petri, como descrito em [Rillo 1988].

Os planos elaborados são planos hierárquicos, não lineares. A metodologia da nossa proposta efetua uma abstração estrutural do problema de “planning”, como descrito em [Mädler 1992]. Entretanto, a rede de Petri é usada para representar tanto o domínio do problema assim como o problema de “planning em si”. O sistema é capaz de aprender estratégias de planejamento, independente do domínio da aplicação, como descrito em [Day 1992]. Entretanto, o aprendizado de possíveis estratégias de busca da solução é feita através da análise das propriedades estruturais das redes de Petri. Além disso, esta análise pode sugerir partições do problema em subredes, para decompor o problema em subproblemas que podem ser resolvidos de uma forma independentes, como foi usado em [Day 1992].

Quanto à monitoração da execução do plano, podemos afirmar que este pode ser efetuado, com vantagens, usando-se a representação do plano em redes de Petri, como foi usado em [Paiva 1993], ao invés de se fazer uma descrição dos estados de execução/habilitação de uma tarefa como em [McDermott 1978].

Quanto ao replanejamento na falha, a representação do plano na forma de rede de Petri pode ser vista como uma alternativa aos mecanismos de tabela triângulo usada em STRIPS+PLAN-EX [Fikes, Hart et Nilsson 1972] e ao mecanismo da árvore de submetas e do grafo de decisão, utilizados por [[Hayes 1975]. A vantagem da nossa proposta

está em se homogeneizar o tipo de representação utilizado em todo o sistema, em todas as etapas do “planning”, como foi enfatizado por [Rillo 1988].

Finalmente, a representação do modelo de ação em nossa proposta não utiliza mais o mecanismo do tipo adição/eliminação da hipótese STRIPS. Desta forma, evitamos os problemas inerentes à esta representação: (i) o “frame problem”, (ii) o problema da interdependência entre ações. Por exemplo, no problema do Mundo de Blocos, podemos evitar a ocorrência da Anomalia de Sussman, sem necessidade de se elaborar antes um plano com falhas para depois corrigi-lo como é feito em [Sussman 1977], [Waldinger 1975], [Lifschitz 1987], [Sacerdoti 1975]. Ao contrário, um plano sem falhas é elaborado desde o princípio.

CAPÍTULO 3

3

O FORMALISMO DAS REDES DE PETRI

Em nossa proposta de estruturação do problema de “planning”, utilizamos o formalismo das redes de Petri associado com regras heurísticas em IA. A maioria dos sistemas de “planning” e de “scheduling” trata de domínios que, na verdade, são sistemas discretos ou que admitem uma aproximação como sistemas a eventos discretos (SEDs). A rede de Petri [Murata 1989], [Reisig 1985], [Valette 1992], [Miyagi 1993a], [Peterson 1981], é uma forma tradicional de modelagem de controle de sistemas dinâmicos a eventos discretos (SDEDs).

Já descrevemos anteriormente no capítulo 2, item 2.6, como as redes de Petri são utilizadas nos sistemas de planejamento baseados em IA. [Drummond 1985] usa a RdP para fazer a representação do plano. [Rillo 1988] desenvolveu um sistema de controle inteligente que utiliza uma representação do conhecimento híbrida, baseada em IA e redes de Petri. [Paiva 1993] e [Hendler 1992] utilizam a RdP para descrever os aspectos procedimentais (ações primitivas) dos operadores.

As redes de Petri são especialmente adequadas para se fazer a representação das relações de causa-efeito que ocorrem nos “design” de sistemas de engenharia [Silva 1992]. Esta abordagem é interessante porque o problema de “planning” pode ser visto como uma instância do problema mais geral de “design”. Ou seja, em engenharia, podemos comparar o processo de síntese de um plano com o processo de design de produto ou de um dispositivo. As redes de Petri podem ser aplicadas no “job shop scheduling” para uma célula de manufatura flexível [Valette et Silva 1989].

Finalmente, mas não menos importante, podemos fazer um paralelo entre os SDEDs e os sistemas de planejamento reativos. Os SDEDs do mundo real podem ser caracterizados pelo fato de o estado do sistema mudar como consequência de eventos que não foram causados pelo sistema controlador, i.e., que estão fora do seu controle. A rede de Petri é capaz de modelar todos os potenciais estados de um SDED, através da repre-

sentação de seus elementos e características essenciais. Entretanto, é a marcação da rede que indica que o sistema está num determinado estado dentre todos os estados potenciais. A evolução destas marcas determina um comportamento do sistema discreto.

Por sua vez, os sistemas de planejamento baseados em IA que apresentam um comportamento de um agente reativo, operam num ambiente em que convivem múltiplos agentes, eventualmente com objetivos antagônicos, atuando de uma forma independente. Por este motivo, muitas vezes ocorrem eventos que modificam o estado do mundo sem o controle do agente responsável pelo planejamento. Por este motivo, em geral, estes sistemas operam com planos incompletos que são mantidos factíveis a todo instante, modificando-os de uma maneira reativa para responder aos eventos que ocorrem no ambiente em que atuam. Um exemplo de sistema de planejamento baseado em IA para sistemas emergenciais, baseado na arquitetura do tipo “blackboard”, está descrito em [Hendler 1992]. Um agente que suporta planos emergenciais diferencia-se do agente reativo porque, neste, é a chegada de uma nova tarefa mais urgente que dispara o gatilho para se fazer uma mudança do plano corrente, ao invés de ser disparado a partir da ocorrência de uma falha na execução do plano corrente, como é no caso do agente reativo.

O que é importante assinalar é que a nossa proposta pode ser estendida para representar tanto o domínio da aplicação, ou seja, o ambiente no qual o atua o agente reativo, assim como o estado interno do agente reativo. O estado interno do agente reativo corresponde à suas atitudes psicológicas em termos de crença, desejo e intenção correntes [Georgeff et Lansky 1986].

Existem diversos tipos de redes de Petri. Inicialmente, vamos descrever as redes elementares.

3.1 As Redes de Petri Elementares

Os princípios básicos na teoria das redes de Petri Elementares (REs) [Rozemberg 1995], para formular as noções de estados e mudança-de-estados (ou transições de estados), são:

- (i) condições (C) e transições (E) são duas noções inter-relacionadas, porém distintas ($C \cap E = \emptyset$), que podem ser tratadas como eventos discretos.
- (ii) Ambas, estados e transições, são entidades distribuídas. As condições representam os estados do sistema, (enquanto que)
- (iii) as transições representam as mudanças de estado. A mudança de estados se faz através da habilitação e disparo de transições.
- (iv) Uma transição (e_j) está habilitada e pode ser disparada num determinado estado se e somente se (sse) todas suas pré-condições são verdadeiras e todas suas pós-condições não são verdadeiras naquele estado.

Definição: Uma estrutura de Rede Elementar $N = (C, E; A)$ consiste de: dois conjuntos disjuntos (C e E) que representam elementos estáticos (p -elementos) e dinâmicos (t -elementos) e relações binárias de fluxo $A \subseteq (C \times E) \cup (E \times C)$, em N . Para que N seja uma estrutura de grafo, é necessário ainda que:

- (i) N seja *pura*, i.e., não contenha laços internos ou “self-loops”. Um par (c, e) é um “self-loop” sse $c A e$ e $e A c$.
- (ii) N não contém *nós isolados*. Um nó $x \in N$ é dito um nó isolado sse ${}^\bullet x \cup x^\bullet = \emptyset$.
- (iii) N é *simples*. Uma estrutura de grafo é chamada simples sse seus distintos elementos (p -elementos e t -elementos) $x_1, x_2 \in N$ não têm as mesmas pós-condições e mesmas pós-condições, i.e.,

$$\forall x_1, x_2 \in N : ({}^\bullet x_1 = {}^\bullet x_2) \wedge (x_1^\bullet = x_2^\bullet) \Rightarrow x_1 = x_2.$$

À seguir, vamos descrever a definição de um Sistema de Rede Elementar (RE), que é muito útil para se fazer a descrição de sistemas do mundo real. Um Sistema de Eventos Discretos (SED) não está completamente especificado até que se faça a inclusão dos estados (M) que queremos considerar, somando-se à estrutura de grafo $N = (C, E; A)$. Um Sistema de Rede Elementar (RE) é também chamado de Rede de Petri Condição / Evento (RdP C/E).

Definição: Uma Sistema de Rede Elementar (RE) [Rozemberg 1995], [Reisig 1985] é um grafo ordenado, bipartido, definido pela 4-upla $\Sigma = (C, E; A, M)$, onde $C = \{c1, c2, \dots\}$ é o conjunto de condições, $E = \{e1, e2, \dots\}$ é o conjunto de transições, e $A \subseteq (C \times E) \cup (E \times C)$ é o conjunto de arcos, e $M \subseteq C$ é a marcação da rede. $(C, E; A)$ representa a (estrutura da) rede subjacente em Σ , enquanto $M \subseteq C$ representa a marcação da rede Σ . $M_0 \subseteq C$ representa a marcação inicial da rede Σ .

Um Sistema de Rede Elementar (RE) requer ainda que:

- (i) a estrutura de grafo $(C, E; A)$ seja simples, sem nós isolados,
- (ii) $\forall e \in E \exists M \in ge(\Sigma) \mid M[e]$, ou seja, para cada transição $e \in E$, existe um marcação da rede que pertence ao espaço estado da rede*¹ $ge(\Sigma)$ em que a transição e está habilitada. Este requisito equívale a dizer que o grafo deve ser puro porque uma transição que contém um “self-loop” nunca está habilitada.

Graficamente, uma condição (c_i) é representada por uma circunferência; uma transição (e_j) é representada por um retângulo; um arco é representado por um arco direccionado conectando $(C \times E)$ ou $(E \times C)$; as marcas são representadas através de círculos de cor preta no interior das circunferências que denotam condições.

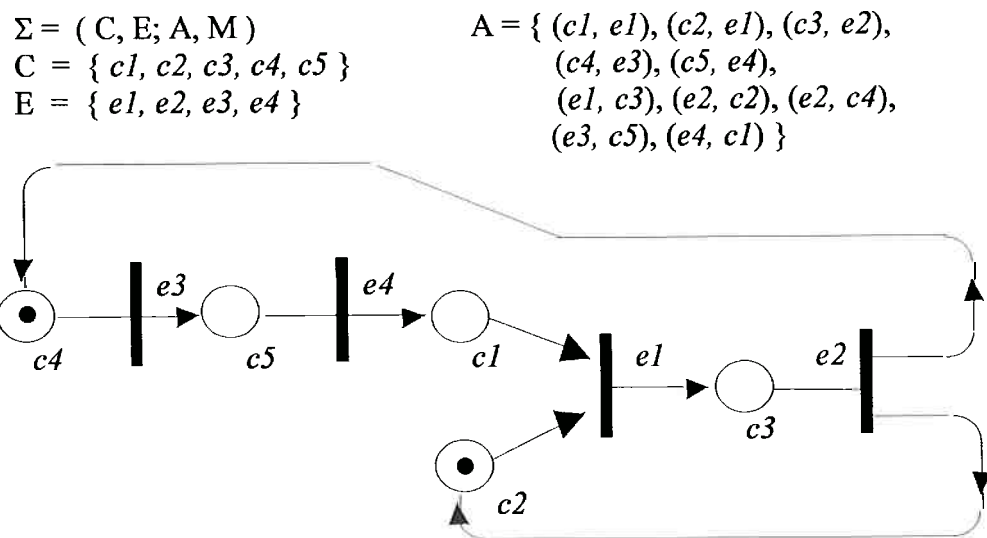


Figura 21 - Exemplo de uma rede de Petri Elementar (RE)

*¹ Veja o item 3.1.2 - O espaço de estado dos sistemas.

A Figura 21, acima, ilustra um exemplo de Rede de Petri Elementar. Considere uma transição qualquer (e_j). As pré-condições de (e_j) são definidas por $\bullet e_j = \{ c_j \in C \mid (c_j, e_j) \in A \}$. As pós-condições de (e_j) são definidas por $e_j \bullet = \{ c_j \in C \mid (e_j, c_j) \in A \}$. Por exemplo, seja a transição e_1 , na Figura 21.

$$\bullet e_1 = \{ c_1, c_2 \}$$

$$e_1 \bullet = \{ c_3 \}$$

Habilitação da transição: Numa determinada marcação M da rede Σ , uma determinada transição e_j pode ser disparada, i.e., está habilitada ($M [e_j >$), se e somente se todas as suas pré-condições são verdadeiras em M ($\bullet e_j \in M$) e nenhuma pós-condição é verdadeira em M ($e_j \bullet \cap M = \emptyset$).

Este requisito de observação das pós-condições ($e_j \bullet \cap M = \emptyset$) para a habilitação de uma transição e_j é muito importante na modelagem de problemas do mundo real. Vamos supor que esta condição não seja observada. Isto significa que, em termos práticos, podemos ter a especificação de uma seqüência indesejável de ações, como por exemplo, permitir que um arquivo já gravado anteriormente possa ser regravado, que um copo já cheio possa ser enchido de novo ou que um carro possa ser movido para o mesmo lugar onde um outro carro já estiver estacionado. Para o caso da habilitação da transição e_1 , para a rede de Petri na Figura 21, temos :

$$M [e_1 > \Leftrightarrow (c_1, c_2 \wedge \neg c_3)$$

Disparo da transição: Numa determinada marcação M da rede Σ , quando ocorre o efetivo disparo de uma transição habilitada e_j , ($M [e_j > M'$), todas as suas pré-condições ($\bullet e_j$) cessam de serem verdadeiras e todas as pós-condições ($e_j \bullet$) começam a ser verdadeiras; as demais condições da rede permanecem inalteradas. Como conseqüência, o estado resultante $M' = ((M - \bullet E) \cup E \bullet)$. A Figura 22, seguinte,

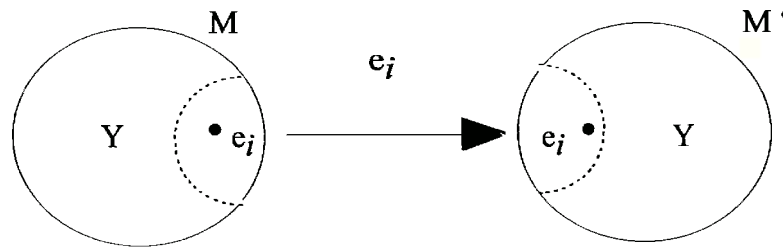
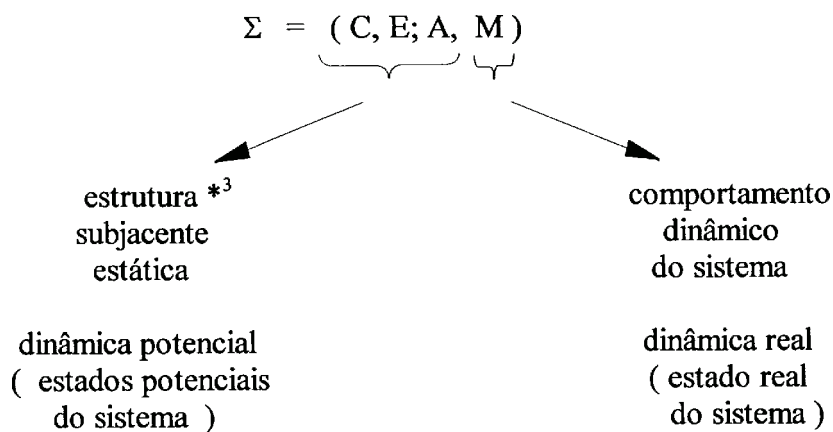


Figura 22 - A mudança-de-estados no disparo de uma transição

ilustra a mudança-de-estados no disparo de uma transição. Observe na Figura 22 que a mudança-de-estado produzida pelo disparo de uma transição está confinada estritamente à sua vizinhança*². Por exemplo, seja a transição e_1 , na Figura 21.

$$M = (c_1, c_2 \wedge \neg c_3) [e_1] M' = (\neg c_1, \neg c_2 \wedge c_3)$$

Uma RE Σ pode ser vista como um modelo abstrato de um sistema distribuído por Eventos Discretos, onde:



*² Note que esta representação da mudança de estados do sistema através do mecanismo de habilitação e de disparo de transições numa Rede de Petri é capaz de especificar corretamente os efeitos da aplicação de uma ação. Por este motivo, torna-se uma alternativa interessante para a solução do "frame problem" em IA, descrito no cap. 4, item 4.1.1. O "frame problem" [Hayes 1973] é um fenômeno que decorre da necessidade de se escrever uma grande quantidade de axiomas "frame" semelhantes uns aos outros para garantir que cada propriedade do domínio que não é afetado pela aplicação de uma determinada ação causando a mudança de estados.

*³ Neste trabalho, utilizamos apenas a estrutura da rede de Petri para fazer a estruturação do problema de "planning".

3.1.1 - As Relações entre as transições:

Numa RE, em termos estruturais, duas transições, e_i e e_j , podem ocorrer segundo três tipos de arranjos: em seqüência, escolha (ou conflito) e em paralelo.

Seqüência: Duas transições, e_i e e_j , estão arranjados em seqüência para uma determinada marcação M , se e somente se $M [e_i > \wedge \neg M [e_j > \wedge M' [e_j >$, onde $M [e_i > M'$. Por exemplo, na Figura 21, e_3 e e_4 estão dispostos em seqüência.

Escolha (ou Conflito): Duas transições, e_i e e_j , estão arranjados numa situação de escolha, ou de conflito, para uma determinada marcação M , se e somente se $M [e_i > \wedge M [e_j > \wedge \neg M [\{ e_i, e_j \} >$. Ou seja, as transições habilitadas e_i e e_j podem ser disparadas individualmente, mas, não podem ser disparadas ao mesmo tempo.

Paralelo: Duas transições, e_i e e_j , estão arranjadas em paralelo, para uma determinada marcação M , se e somente se $M [\{ e_i, e_j \} >$. Ou seja, as transições habilitadas e_i e e_j podem ser disparadas ao mesmo tempo, sem que exista nenhuma interferência entre elas. Além disso, não existe nenhuma limitação quanto à ordem de disparo destas transições. Desde que os efeitos do disparo de uma determinada transição (e_i) está confinada a $(\bullet e_i \cup e_i \bullet)$, a noção de *independência* entre duas transições pode ser formalizada como:

$$(\forall e_i, e_j) [(e_i \neq e_j) \Rightarrow (\bullet e_i \cup e_i \bullet) \cap (\bullet e_j \cup e_j \bullet) = \emptyset]$$

Confusão: Duas transições, e_i e e_j , estão numa situação de confusão quando apresenta um arranjo que combina as situações de conflito e de paralelismo. Porque estas transições, e_i e e_j , apresentam pré-condições em comum, dependendo do estado M que habilita estas transições, estas podem ser habilitadas numa situação de conflito ou em paralelo.

3.1.2 - O Espaço de Estados dos Sistemas:

O espaço de estados do sistema corresponde ao conjunto de marcações acessíveis de uma rede de Petri, ou seja ao conjunto de estados do sistema que se pode atingir, a partir do estado inicial, através de uma seqüência de disparos de transições. Desde que este espaço é finito, podemos representá-lo graficamente na forma de um grafo, onde vértices representam marcações da rede e um arco liga dois vértices do grafo se existe um conjunto de transições paralelas que permite a passagem de um vértice para o outro. Este grafo representa o conjunto de marcações acessíveis, e é chamada de *grafo de estados*, ou de grafo de atingibilidade da RdP.

Definição: Um grafo de estados (ge) de uma rede Σ , é um grafo direcionado com arcos rotulados, inicializado, $((M, T), M_0)$, tal que os nós do grafo (M) representam todos os estados do sistema e os arcos rotulados (T) representam o disparo de um conjunto de transições paralelas habilitadas (U) entre dois estados do sistema (M_i, M_j) . Um arco rotulado é definido por:

$$T = \{ (M_i, U, M_j) \mid M_i, M_j \in M \wedge U \in T \wedge M_i [U > M_j] \}.$$

O ge em que o arco rotulado (T) representa o disparo de uma única transição (“singleton”), i.e., $T = \{ e_i \}$, o grafo é chamado de grafo seqüencial de estados (gse), ou uma *máquina de estados*. O gse pode ser obtido a partir de um grafo de estados eliminando-se todas as arestas que não sejam de transições isoladas (“singletons”). O ge para a rede Elementar (RE) ilustrada na Figura 21, está mostrado na Figura 23, seguinte.

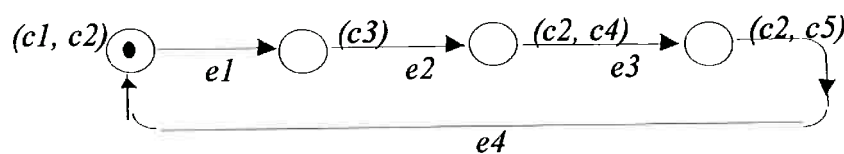


Figura 23 - O Grafo de estados para a RE da Figura 21

Duas redes Σ_1 e Σ_2 apresentam o espaço de estados semelhantes, $\Sigma_1 \cong \Sigma_2$, se e somente se o grafo de estados de Σ_1 é isomórfico ao grafo de estados de Σ_2 .

Agora, podemos descrever as redes de Petri ordinárias, ou as redes do tipo “Lugar / Transição”, ou RdP P/T.

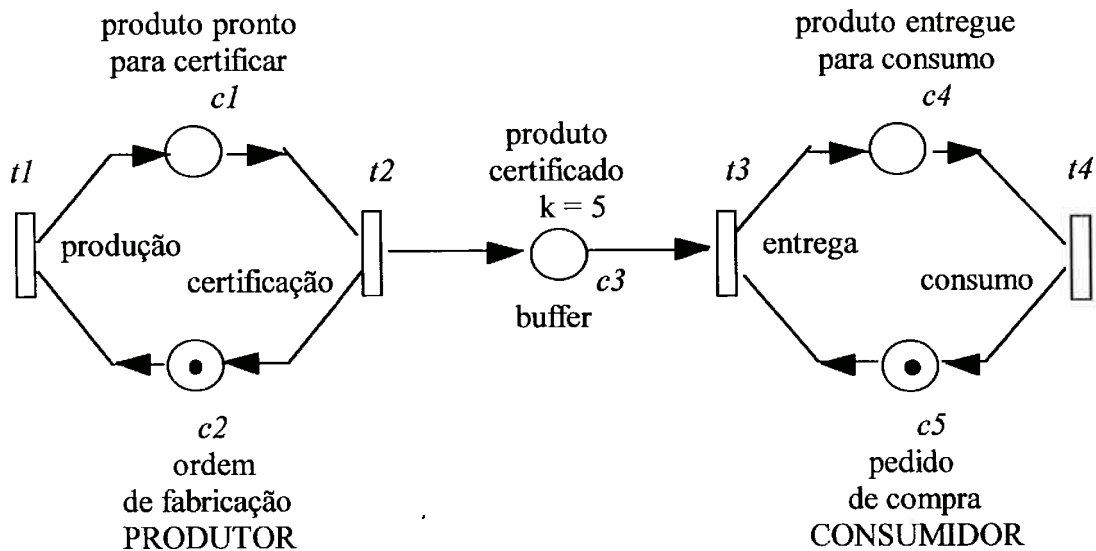


Figura 24 - A RdP P / T para representar uma relação produtor - consumidor

3.2 A Rede de Petri "Lugar/Transição"

A RdP "Lugar/Transição" (RdP P/T) [Murata 1989], [Reisig 1985], [Valette 1992], [Miyagi 1993a], [Peterson 1981], ilustrado na Figura 24, é um grafo ordenado, bipartido, definido pela 5-upla $\Sigma = (P, T; A, W, M)$, onde P representa um conjunto finito de lugares. $P = [p_i]$, T representa um conjunto finito de transições $T = [t_j]$. A é um conjunto de "arcos" que indica uma relação entre P e T ($A \subseteq (P \times T) \cup (T \times P)$). $W: A \rightarrow \mathbf{N}^+$ é o peso dos arcos e M a marcação da rede. A estrutura da rede de Petri, sem uma associação de qualquer marcação inicial M_0 , é denotada por $N = (P, T; A, W)$.

Como nas Redes Elementares (REs), descrito no item 3.1, acima, o estado do sistema é indicado pela distribuição de marcas ("tokens"), representado graficamente (quando possível) como círculos de cor preta colocados no interior dos lugares; e a

evolução entre os estados é representada pela movimentação destas marcas. A Figura 24, acima, ilustra uma RdP P / T para representar um sistema clássico produtor-consumidor.

3.2.1 - A Rede de Petri “Condição/Evento” (C/E)

A RdP Condição/Evento (C/E), já descrita anteriormente no item 3.1, pode ser considerada como sendo um tipo especial de RdP P/T em que as capacidades de todos os lugares e os pesos em todos dos arcos são iguais a 1. Do mesmo modo, uma rede de Petri P/T com as capacidades dos lugares e os pesos dos arcos iguais a 1 comporta-se como uma rede de Petri que é constituída apenas por condições e eventos [Reisig 1985].

Por suas características, as redes de Petri C/E se aplicam à abstração de sistemas complexos (configuração, modelagem conceitual, etc.) ou mais diretamente na representação de algoritmos de controle de sistemas seqüenciais. Portanto, neste processo de modelagem, é preciso evitar certos estados a partir das quais a evolução do sistema não é muito clara, como o *contato*. Uma situação de contato ocorre para uma determinada transição e_j quando, para uma marcação M de uma rede Σ , mesmo existindo marcas em todos os seus lugares de entrada para habilitar o disparo da transição e_j , não existe capacidade suficiente nos seus lugares de saída para tanto. O contato é definido por: $\bullet e_j \subseteq M \wedge e_j \bullet \cap M \neq \emptyset$.

Numa RdP C/E, tais situações de contato sempre podem ser evitadas através da transformação da rede numa RdP C/E completa equivalente, que é livre de contatos. Uma RdP C/E completa resulta da transformação de uma RdP C/E, acrescentando lugares duais (ou complementares) correspondentes a todos os seus lugares. De maneira análoga, uma RdP P/T pode ser completada se a capacidade dos lugares for finita, incluindo-se lugares de uma forma que o seu comportamento não é modificado mas as situações de contato são excluídas.

3.3 As Propriedades da Rede de Petri

A seguir, descrevemos brevemente as propriedades estruturais e comportamentais de uma Rede de Petri $\Sigma = (P, T; A, W, M)$ [Valette 1992]. As propriedades estruturais derivam apenas da estrutura estática subjacente à rede Σ , ou seja, não dependem da marcação (M), mas, apenas da estrutura $N = (P, T; A, W)$. As propriedades comportamentais dependem da estrutura subjacente $N = (P, T; A, W)$ e da marcação (M) da rede Σ .

As propriedades estruturais da Rede de Petri são independentes da marcação inicial. Desta forma, os métodos de cálculo para se fazer a determinação se estas se verificam ou não podem ser obtidos diretamente a partir de suas definições. Estes métodos fazem a resolução de um sistema de equações lineares. Estas propriedades estão relacionadas com os componentes conservativos e repetitivos estacionários descritos em seguida.

3.3.1 - Os Invariantes de Lugar

Os componentes conservativos (ou invariantes de lugar) constituem uma propriedade que provém diretamente da análise da estrutura da rede de Petri e não depende da sua marcação inicial. Por exemplo, considere a RdP da Figura 24, na página 81. Vamos considerar a soma de marcações para os lugares $c1$ e $c2$, i.e., $M(c1) + M(c2)$. Esta soma vale sempre 1 porque a marcação inicial M_0 é igual a 1 (o mesmo vale para os lugares $c4$ e $c5$, cuja soma $M(c4) + M(c5)$ é sempre igual a 1). O disparo da transição $t1$ não altera o valor desta soma, nem tampouco o disparo da transição $t2$. De fato, qualquer que seja a marcação inicial M_0 , temos sempre que:

$$M(c1) + M(c2) = M_0(c1) + M_0(c2) \quad \forall M \in \text{ge}(\Sigma)$$

Por este motivo o conjunto de lugares $c1$ e $c2$ são chamados de componentes conservativos e a soma $M(c1) + M(c2)$ é chamada de invariante de lugar. Os componentes conservativos definem uma subrede na rede de Petri.

3.3.2 - Os Invariantes de Transição:

Considere novamente a Figura 24, na página 81. Se executarmos a seqüência de disparo de transições $t_1 t_2 t_3 t_4$ a partir da marcação inicial M_0 da rede Σ , obtemos novamente esta mesma marcação da rede*⁴, i.e., $M = M_0$. Neste caso, os componentes repetitivos a considerar são todos os lugares e transições da RdP. Um componente repetitivo ou invariante de transição é caracterizado pela existência de uma seqüência de disparo de transições que não modifica a marcação da RdP. Tal invariante corresponde a uma seqüência de eventos (ou seja, um passo) que pode ser repetida indefinidamente.

As definições das propriedades comportamentais de uma RdP Σ dependem do conjunto de marcações acessíveis a partir da marcação inicial da rede M_0 . Portanto, não se pode derivar algoritmos diretamente a partir daquelas definições para se determinar se uma propriedade comportamental se verifica ou não, porque este conjunto de marcações acessíveis nem sempre é finito e, portanto, algumas aproximações se fazem necessárias. No item 3.3, mais adiante, apresentamos alguns métodos clássicos de análise de propriedades. À seguir, mostramos algumas propriedades comportamentais das Redes de Petri.

3.3.3 - A Rede Limitada:

Um lugar é dito *k-limitado* se o número máximo de marcas no lugar está limitada em k . Uma rede de Petri é *k-limitada* se todos os seus lugares são k -limitados. A idéia da rede limitada corresponde ao fato de que um sistema físico está sempre limitado. Entretanto, alguém pode ser levado a utilizar a RdP não limitada quando quiser avaliar o desempenho de um sistema sem levar em conta as limitações dos seus elementos de estocagem intermediária.

*⁴ Note que a rede pode ser reversível, apesar de não ser, necessariamente, cíclica.

3.3.4 - A Rede Cíclica:

Uma rede Σ é chamada de rede cíclica sse $\forall M, M' \in ge(\Sigma), M [\sigma > M'$, onde σ é uma seqüência de disparo de transições. Em outras palavras, numa rede cíclica uma determinada transição sempre pode ser disparada de novo. É claro que uma rede cíclica é uma rede viva.

3.3.5 - A Vivacidade:

O conceito de vivacidade está intimamente ligada à completa ausência de gargalos (“deadlocks”) no sistema alvo. Uma transição dita uma *transição viva* sse $\forall M \in ge(\Sigma), \exists t' M [t' > M' \wedge M' [t >$. Uma transição *morta* é uma transição que não é viva. Uma rede dita uma *rede viva* sse $\forall t \subseteq T : t$ é uma transição viva. A RdP da Figura 24, na página 81, é uma rede de Petri viva.

3.3.6 - A Rede Reversível:

Uma rede de Petri é dita uma *rede reversível* sse $\forall M' \in ge(\Sigma), \exists t M' [t > M_0$, onde $ge(\Sigma)$ é o grafo de estados da rede Σ -- o grafo de estados foi descrito no item 3.1.2. Em outras palavras, uma RdP é reversível se a sua marcação inicial M_0 da RdP pode ser alcançada novamente. Limitação, vivacidade e reversibilidade são propriedades independentes entre si [Murata 1989]. Por exemplo, uma rede pode ser reversível, mesmo não sendo uma rede cíclica. A RdP da Figura 24, na página 81, é uma RdP reversível.

3.4 - A Análise das Propriedades da Rede de Petri

A análise da estrutura da rede de Petri pode indicar, por exemplo, a existência de determinados tipos de arranjos entre transições: seqüência, conflito, paralelismo ou confusão. A estratégia de busca da solução a ser utilizada deve se adequar a cada caso, por exemplo, no caso de transições em seqüência ou em paralelo, basta usar um mecanismo

do tipo “forward chaining”; no caso de conflito ou de confusão, é preciso estabelecer regras para a escolha da transição a ser disparada.

Do mesmo modo, a análise das propriedades comportamentais também pode auxiliar a definição da estratégia de busca da solução (embora a sua determinação ser mais complexa). Por exemplo, o fato de uma rede ser viva assegura que não existe o risco de o processo de busca da solução atingir um gargalo (ou seja, um “dead-lock”).

Vimos no item 3.2.2, acima, que as propriedades podem ser estruturais e comportamentais. As propriedades estruturais derivam da estrutura $N = (P, T; A, W)$ subjacente à rede Σ , e independem da marcação da rede (M). Por este motivo, os métodos de cálculo para se fazer a determinação se estas se verificam ou não podem ser obtidos diretamente a partir da resolução de um sistema de equações lineares. As propriedades comportamentais dependem de ambos, da estrutura subjacente $N = (P, T; A, W)$ e da marcação da rede (M). Uma maneira de levar em conta a marcação da rede é através da enumeração completa do espaço de estados (ge) da rede Σ . O problema decorre do fato de que, muitas vezes, este espaço de estados não é finito. Para sistemas muito grandes, este problema pode se tornar intratável, devido à explosão combinatória. Por este motivo, limitamos a nossa proposta à análise da estrutura da RdP.

[Valette 1992] apresenta três métodos para se fazer a análise das propriedades da rede de Petri: Método da enumeração completa das marcações, Método de redução das redes (método gráfico) e Método da resolução de equações lineares (método algébrico).

Vamos ilustrar, a seguir, o método da enumeração completa das marcações [Valette 1992]. Este método baseia-se na construção do grafo de estados (ou árvore de recobrimento das marcações da rede). Este método termina até mesmo no caso em que a rede é ilimitada. O que se deseja obter com esta análise é determinar as boas propriedades comportamentais da rede, i.e. que a rede é *k-limitada*, *reversível* e *viva*.

Para se construir o grafo de estados (ge), inicia-se a partir da marcação inicial (M_0). Cada transição habilitada nesta marcação M_0 dá origem a uma ramificação. Cal-

culam-se as marcações atingidas pelo disparo destas transições e recomeça-se o processo para cada marcação atingida. A construção do grafo de estados (ge) termina quando:

- é encontrada uma marcação igual a uma marcação já encontrada e para a qual todas as marcações sucessoras já foram calculadas ou vão ser calculadas porque correspondem a ramificações pendentes que serão exploradas em seguida.
- é encontrada uma marcação estritamente maior do que uma marcação da ramificação em curso de exploração. Neste caso, a construção do (ge) é interrompida porque a RdP não é limitada. Concluimos que uma determinada seqüência de disparo de transições está sendo repetida e, por isso, o número de marcas aumenta a cada vez em pelo menos um lugar.

A partir o algoritmo para construção do grafo de estados (ou árvore de recobrimento), descrito acima, concluimos que a propriedade da rede ser limitada é decidível, até mesmo no caso de redes ilimitadas.

O que se deseja obter com a análise de propriedades é determinar as boas propriedades comportamentais da rede, i.e. que a rede é *k-limitada*, *reversível* e *viva*. [Valette 1992] recomenda os seguintes passos para fazer a determinação das propriedades comportamentais:

PASSO1: provar que a rede é *k-limitada*.

PASSO2: se a rede for *k-limitada*, então construir o grafo de estados do sistema (ge).

PASSO3: se o grafo de estados (ge) for fortemente conexo, então a rede é *reversível*.

PASSO4: se a rede é reversível, então a rede é *viva*.

Seja a RdP ilustrada na Figura 21. Esta rede é isenta de transições em paralelo, em conflito ou em confusão -- todas as transições estão segundo um arranjo do tipo seqüência. Ela é *k-limitada*, porque o número máximo de marcas em cada lugar é $k = 1$. Traçando-se o grafo de estados, tem-se o (ge) mostrado na Figura 23. Este é um grafo fortemente conexo. Daí, concluimos que a rede é reversível e, portanto, viva. Desta forma, temos que esta RdP é *k-limitada*, reversível e viva.

O método gráfico de redução da redes pode ser visto em [Valette 1992]. O método algébrico de resolução de equações lineares pode ser visto em [Silva et Miyagi 1996], [Silva et all 1996].

Existem diversos programas que implementam ambientes para editar uma rede de Petri graficamente, analisar suas propriedades e fazer a sua simulação. A nossa proposta de metodologia, está baseada na utilização do ambiente computacional HIPER (“A High Level Integrated Petri Net Environment for the Design of FMS”) [Silva et all 1996], para fazer o modelamento do modelo conceitual do problema de “planning”. O ambiente HIPER promove o modelamento e simulação de uma rede de Petri estendida Orientada a Objetos Ghenesys (“General Hierarchical Extended Net System”). O Ghenesys é baseado na metodologia PFS / MFG [Miyagi 1988]. Para fazermos a análise das propriedades da rede de Petri C/E do nosso exemplo, descrito no capítulo 4, item 4.5, utilizamos o programa “Petri Netze für Windows” [Frank et Schmidt 1993].

3.5 A Rede de Petri Estendida Orientada a Objetos Ghenesys

A modelagem de sistemas complexos de grande porte usando redes de Petri apresenta algumas dificuldades. Por exemplo, à medida que o tamanho da rede aumenta, torna-se difícil manter uma interpretação consistente dos elementos básicos do sistema. O problema piora se tentarmos fazer uma análise das propriedades estruturais, como invariantes de lugar ou invariantes de transição. Uma solução possível para este problema está em se utilizar redes de Petri mais sintéticas, baseadas nas redes de alto nível e/ou nas redes estendidas.

O HIPER (“A High Level Integrated Petri Net Environment for the Design of FMS”) [Silva et Miyagi 1996], [Silva et all 1996] é um ambiente computacional desenvolvido no Laboratório de Automação de Sistemas (LAS) da Escola Politécnica da Universidade de São Paulo (EPUSP), para fazer o modelamento, análise e simulação de sistemas usando a rede de Petri estendida Orientada a Objetos Ghenesys (“General Hierarchical Extended Net System”). A rede estendida Ghenesys foi inspirada na metodo-

logia PFS / MFG [Miyagi 1988]. Esta consiste de uma abordagem “top-down” para se fazer a modelagem e o mapeamento dos elementos físicos do sistema produtivo (representado através do “Production Flow Schema”- PFS) nos elementos da rede de Petri estendida MFG (“Mark Flow Graph”). O produto final da aplicação deste método PFS / MFG é um conjunto hierárquico de redes MFG, com diversos níveis de abstração. O MFG é uma rede aplicativa segura, baseada na rede de Petri C / E.

A rede de Petri é composta por duas classes de elementos, normalmente associados com os elementos *estáticos* e *dinâmicos* da aplicação alvo. A rede estendida Ghenesys é constituída por duas classes de objetos: a classe *box* (o elemento passivo -- *estático*) e a classe *atividade* (o elemento ativo - *dinâmico*).

Na rede de Petri estendida Ghenesys, cada objeto estático da classe *box*, pode representar uma única *condição* ou pode representar um “*elemento estático composto*”, i.e., uma subrede estática indicando elementos passivos de armazenamento. Desta forma, pode se representar modos específicos de armazenamento de itens, como FIFO, LIFO, etc. Processos de montagem e de desmontagem (que afetem apenas o fluxo de itens) também podem ser representados por um *box*. Na RdP estendida Ghenesys, cada objeto dinâmico denominado *atividade*, também pode representar um único *evento* instantâneo (ou *transição*) ou um “*elemento dinâmico composto*”, i.e., uma subrede dinâmica.

Uma rede de Petri Ghenesys, apresenta diversos níveis interligados de abstração. Esta conexão entre níveis de abstração é tal que, cada objeto atividade pode ser substituído por uma subrede que começa e termina com uma transição, enquanto que um cada objeto *box* pode ser substituído por uma subrede que começa e termina com um *box*. Desta maneira, a dualidade do formalismo da rede de Petri é resgatada.

Outro elemento existente no Ghenesys é a *porta* (“gate”), que consiste de um tipo especial de fluxo ou de uma relação entre os elementos estáticos e dinâmicos originados a partir de sinais externo ao sistema alvo, como por exemplo, a partir de botões de de-

sarme, de chaves de potência, e de outros tipo de atuadores de controle (que são representados como pseudo-condições).

O ambiente de modelagem e de “design” HIPER suporta refinamentos sucessivos e composicionalidade, usando uma abordagem mista, tanto do tipo “top-down” assim como do tipo “bottom-up”. Além disso, diferentemente da proposta original do PFS / MFG, o Ghenesys é uma rede completa e pode ser simulada em qualquer nível de abstração.

A rede estendida Ghenesys [Silva et Miyagi 1996], [Silva et al 1996] é um grafo bipartido, orientado a objetos, definido pela tripla $\Sigma = (B, T, A)$, onde B é um conjunto de objetos denominados *boxes*, T é um conjunto de objetos denominados de *transições* (chamadas aqui como *atividades*), e $A \subseteq (B \times T) \cup (T \times B)$ é um conjunto de relações entre os objetos B e T.

Cada elemento nos conjuntos B ou T é representado como um objeto caracterizado por um conjunto de atributos. O Ghenesys possui duas super-classes principais: *box* e *atividade*.

3.5.1 A Super-Classe Box

Por simplicidade, a super-classe *box* é caracterizada por um conjunto minimal de atributos $box :: \langle \#nome, \#marca \rangle$, onde o atributo *#nome* é uma variável alfanumérica que identifica de uma forma única o objeto box, enquanto que o atributo *#marca* é uma variável inteira que indica o número de marcas no interior do box no caso de um box-capacidade.

A classe *box* apresenta duas subclasses: *box-capacidade* e *box-de-tempo*. A classe *box-capacidade* é caracterizada por $box-capacidade :: \langle box \mid \#capacidade \rangle$, ou seja, ela recebe por herança todos os atributos da classe *box* e, além disso, possui um parâmetro extra *#capacidade*, que é uma variável inteira que indica o número máximo de marcas que este objeto pode armazenar.

Existe uma subclasse da classe *box-capacidade*: o *elemento-estático-composto* ou a *sub-rede-estática*. A classe *sub-rede-estática* é caracterizada por *sub-rede-estática* :: < *box-capacidade* | *#relação-es*, *#ponteiro-subrede* >, ou seja ela recebe por herança todos os atributos da classe *box-capacidade* e, além disso, possui os atributos *#relação-es*, que indica uma relação de entrada-saída (por exemplo, $1:N$, no caso de um objeto do tipo *box* de desmontagem, e $N:1$, no caso de um objeto do tipo *box* de montagem), e *#ponteiro-subrede* que é um ponteiro para a subrede estática correspondente.

A classe *box-de-tempo* é caracterizada por *box-de-tempo* :: < *box* | *#tempo-estimado* >, ou seja, ela recebe por herança todos os atributos da classe *box* e, além disso, possui um parâmetro extra *#tempo-estimado*, que é uma variável inteira que indica o intervalo de tempo (estimado) que leva para se habilitar o disparo.

3.5.2 A Super-Classe Atividade

A super-classe *atividade* é caracterizada por um conjunto minimal de atributos *atividade* :: < *#nome* >, onde o atributo *#nome* é uma variável alfanumérica que identifica de uma forma única o objeto atividade.

A classe atividade apresenta duas subclasses: *atividade-simples* e *atividade-composta*. A classe *atividade-simples* é caracterizada por *atividade-simples* :: < *atividade* | *#lista-paralelas* >, ou seja, ela recebe por herança todos os atributos da classe atividade e, além disso, possui um parâmetro extra *#lista-paralelas*, que é uma lista de todas outras atividades que podem ser disparadas em paralelo com ela. Desta forma, uma *atividade-simples* é uma definição de um evento com a possibilidade de se especificar outras atividades que podem ser disparadas ao mesmo tempo.

A classe *atividade-composta*, ou *sub-rede dinâmica*, é caracterizada por *atividade-composta* :: < *atividade* | & [2, *atividade-simples*], [1, *box-de-tempo*], *#ponteiro-subrede* >. Uma *atividade-composta*, ou uma *sub-rede-dinâmica*, recebe por herança todos os atributos da classe atividade e, além disso, é composta por duas *atividades-*

simples, chamadas respectivamente de atividades de entrada e de saída, por um *box-de tempo*, que indica o tempo de processamento estimado e um ponteiro *#ponteiro-subrede* para outra subrede (dinâmica) que define os processos especificados.

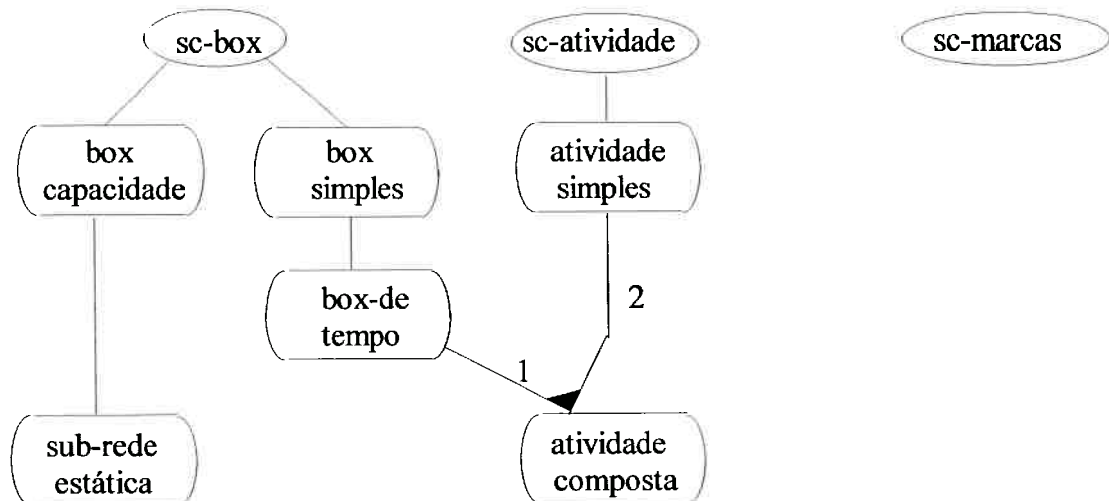


Figura 25 - Hierarquia de classes na rede Ghenesys

O Ghenesys suporta ainda a super-classe *marcas*. A idéia é a de introduzir elementos de alto nível na caracterização das marcas, como nas redes de Petri coloridas [Jensen 1996]. A Figura 25, acima, ilustra a hierarquia de objetos no Ghenesys.

3.5.3 As relações de Fluxo na rede Ghenesys

O conjunto de relações de fluxo A é dado por $[arco, porta-interna, porta-externa]$. As relações de fluxo não definem uma nova classe de objetos, mas sim, uma relação entre os objetos das classes B e T , ou seja, $A \subseteq (B \times T) \cup (T \times B)$. Um fluxo genérico, denotado por $g(a, b)$, é uma relação simples do tipo *arco*, se não existirem marcas persistentes nos objetos a ou b . Se a ou b apresentar uma marca persistente, então a relação de fluxo é do tipo *porta* (ou “gate”). A porta será uma *porta-interna* ou uma *porta-externa*, se o elemento com a marca persistente pertencer ou não à rede. Um elemento associado com uma porta-externa é representado por uma pseudo-condição, por exemplo, $pc1$ e $pc2$, na Figura 26. As portas podem ainda ser classificadas como portas inibidoras ou portas habilitadores, dependendo do papel exercido pela marcação persis-

tente de inibir ou de habilitar uma transição da rede. Na Figura 26, seguinte, a relação $g(pc1, t2)$ é uma porta-habilitadora externa, a relação $g(pc2, t5)$ é uma porta-inibidora externa e as demais relações são arcos simples.

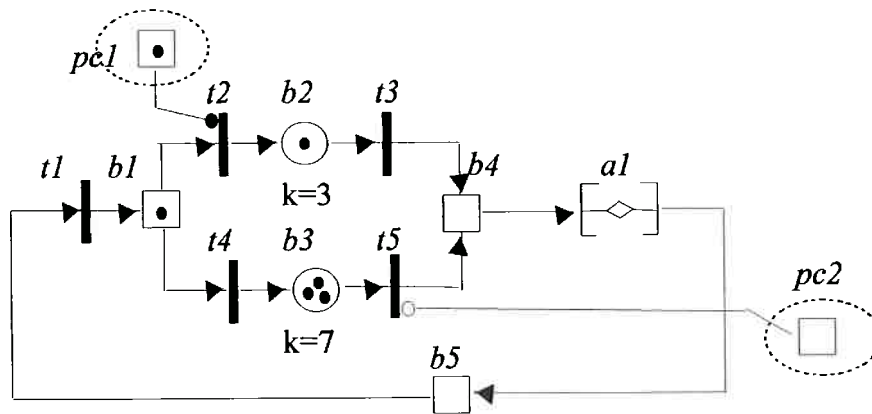


Figura 26 - Exemplo de rede Ghenesys

3.5.4 A Equação de Estado para a rede Ghenesys

A equação de estado para a rede estendida Ghenesys é muito semelhante à equação de estado das Redes Elementares. De fato, existe a intenção de melhorar a sua expressividade, preservando a maior parte do formalismo e das propriedades verificadas nas redes de Petri.

Inicialmente, vamos definir o processo de construção da matriz de incidência $A = [a_{ij}]$, que é uma matriz retangular $n \times m$ de inteiros $\{-1, 0, 1\}$, dada por $a_{ij} = a_{ij}^+ - a_{ij}^-$, onde a_{ij}^+ é o arco a partir da transição i para o seu box de saída j (pós-condição) e a_{ij}^- é o arco a partir da transição i para portas inibidoras j , e a_{ij}^- é o arco a partir do box de entrada i até a transição j (pré-condição) e a partir de portas habilitadoras i até a transição j . Todos os elementos da rede são incluídos na matriz de incidência A . Deste modo, n é o número de elementos dinâmicos da rede, e m é o número de elementos estáticos da rede.

Como as pseudo-condições possuem marcas persistentes, é necessário uma forma de reintroduzi-las após o disparo de uma transição. Para isto, define-se a matriz adicional Δ . A matriz adicional Δ tem a mesma ordem da matriz de incidência A , e é dada por

$[\mathbf{0} \mid \Pi]$, onde $\mathbf{0}$ é a matriz nula e Π é uma sub-matriz de A contendo apenas as colunas correspondentes às pseudo-condições pc_j .

A matriz de incidência A e a matriz adicional Δ , para a rede de Petri na Fig. 26 são:

$$\mathbf{A} = \begin{array}{c|cccccc} & b1 & b2 & b3 & b4 & b5 & pc1 & pc2 \\ \hline t1 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ t2 & -1 & 1 & 0 & 0 & 0 & -1 & 0 \\ t3 & 0 & -1 & 0 & 1 & 0 & 0 & 0 \\ t4 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ t5 & 0 & 0 & -1 & 1 & 0 & 0 & 1 \\ a & 0 & 0 & 0 & -1 & 1 & 0 & 0 \end{array} \quad \mathbf{\Delta} = \begin{array}{c|cccccc} & b1 & b2 & b3 & b4 & b5 & pc1 & pc2 \\ \hline t1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ t2 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ t3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ t4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ t5 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ a & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

3.5.4.1 O Disparo de uma transição:

O vetor t_k de disparo da transição é o vetor coluna, de dimensão $l \times n$, que tem o valor 1 na linha k e o valor 0 nas demais linhas.

Existe uma seqüência finita de transições habilitadas, representada pelo vetor $\sigma_k = [t1] + [t2] + \dots + [tn]$, para uma determinada marcação M_k da rede Σ , se existir uma seqüência $M_k \xrightarrow{t1} M'_k \xrightarrow{t2} M''_k \dots \xrightarrow{tn} M_{k+1}$ tal que $\forall j, 1 \leq j \leq n, M_{j+1} [t_j > M_j$. Ou seja, diz-se que a marcação M_{k+1} é atingível a partir da marcação M_k se e somente se existe (sse) uma seqüência de disparo $\sigma_k : M_k [\sigma_k > M_{k+1}$.

Sejam \mathbf{A}^T e $\mathbf{\Delta}^T$ as matriz transpostas da matriz de incidência A e da matriz adicional de portas Δ , M_k o vetor de marcação da rede Σ , e σ_k o vetor das transições habilitadas. A equação de estado, então, é dada por:

$$M_{k+1} = M_k + (\mathbf{A}^T - \mathbf{\Delta}^T) \sigma_k$$

Com o uso do artifício do uso da matriz adicional Δ , faz-se a retirada das “marcas permanentes” no disparo das transições e, logo em seguida, faz-se a reintrodução destas marcas.

É possível prover um algoritmo fechado para um simulador de rede Ghenesys a partir de algumas definições similares àquelas das redes de Petri convencionais, acrescentada a uma representação algébrica dos termos adicionais como a retenção de marcas no box de tempo e a capacidade $k \in \mathbf{N}$ dos elementos passivos.

3.5.4.2 A Habilitação de uma transição:

Seja \mathbf{a}_k o vetor linha situado na linha k da matriz de incidência \mathbf{A} . Note na matriz \mathbf{A} que o vetor linha \mathbf{a}_k , de dimensão $1 \times m$, descreve as pré-condições e as pós-condições necessárias para a habilitação e disparo da transição t_k . De fato, $\mathbf{a}_k = \bullet t_k + t_k \bullet$. O vetor $\bullet t_k$ corresponde a um vetor coluna $1 \times m$, que tem o valor -1 na linha que corresponde as pré-condições da transição t_k e o valor 0 nas demais linhas. O vetor $t_k \bullet$ corresponde a um vetor coluna $1 \times m$, que tem o valor 1 na linha que corresponde as pós-condições da transição t_k e o valor 0 nas demais linhas. Para um determinado vetor coluna de marcação \mathbf{M} , de dimensão $1 \times m$, da rede Σ , tem-se que a transição t_k está *habilitada*, ou seja $\mathbf{M} [t_k >$, se e somente se (sse):

$$\mathbf{M} \otimes \mathbf{a}_k = [\bullet t_k]$$

onde, \otimes denota o produto direto de duas matrizes, definido como: $\forall \mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbf{a}_k \cup \mathbf{t}_k, k = 1, \dots, n$: $\mathbf{a} \otimes \mathbf{b} = \mathbf{c}$, se para todo i, j , $[\mathbf{c}_{ij}] = [\mathbf{a}_{ij} \cdot \mathbf{b}_{ij}]$.

Levando-se em conta as propriedades associativa, comutativa e distributiva do produto direto, podemos re-escrever o antecedente da equação anterior como [Silva et all 1996]:

$$[(2.M - \mathbf{1} + \mathbf{a}_k) \otimes \mathbf{a}_k]^2 = 0$$

onde, $\mathbf{1}$ é o vetor unitário, de dimensão apropriada $(1, m)$, e o expoente quadrado significa o produto escalar de vetores. Esta equação pode ser avaliada diretamente a

partir dos termos da equação de estado, dispensando a necessidade de se distinguir as relações de entrada das relações de saída, como exigido na formulação original, anteriormente descrita.

Entretanto, a condição acima é apenas necessária. Isto porque as transições “habilitadas” segundo a formulação acima podem estar estruturalmente dispostas segundo arranjos que as coloca numa situação de conflito ou de contato, o que requer uma análise mais aprofundada. Dois vetores correspondentes a duas transições tk e tr podem ser (ou estar):

- (i) independentes: sse $\mathbf{a}_k \otimes \mathbf{a}_r \mid_j = 0, \quad \forall j \ (j = 1, \dots, m)$
- (ii) em conflito: sse $\mathbf{a}_k \otimes \mathbf{a}_r \mid_j = 1, \quad (j = 1, \dots, m)$
- (iii) em contato: sse $\mathbf{a}_k \otimes \mathbf{a}_r \mid_j = -1, \quad (j = 1, \dots, m)$

Uma vez detectadas, estas situações de conflito podem ser resolvidas facilmente, até mesmo com o disparo concorrente, desde que cada transição concorrente esteja na lista concorrente das transições pertinentes. Outra possibilidade está no uso de uma política de escolha da transição, que seja a mais adequada para o domínio da aplicação.

3.5.4.3 A Modularização de M para o box-capacidade:

Os resultados dos cálculos usando-se as equações algébricas, descritas acima valem, em princípio, para o caso de *redes de Petri seguras*, i.e., com o vetor de marcação $M \in \{0, 1\}$. Entretanto, novos elementos foram incorporados à rede de Petri estendida Ghenesys, que a distanciam da rede de Petri C/E , em especial o box-capacidade que pode armazenar k marcas simultaneamente.

À seguir, mostramos o processo de modularização para o box-capacidade (com capacidade finita) que se faz para conseguir que a rede estendida Orientada a Objetos Ghenesys mantenha a características de uma rede segura. Para que as equações descritas

nos itens 3.5.4.1 e 3.5.4.2, acima, sejam aplicáveis, é necessário normalizar o vetor de marcação M para que este contenha apenas valores $\{0, 1\}$.

Seja o vetor coluna C , de dimensão $(1, m)$, o vetor capacidade dos boxes da rede Σ . A modularização do box-capacidade da marcação M , em relação à uma transição t_k , denotada pela função de mapeamento $\mu(M, t_k) = \bar{M}^{tk}$ é definida como sendo:

$$\bar{M}^{tk} |_j = \begin{cases} 1 & \text{se } M \otimes a_k |_j < 0 \\ 0 & \text{se } M \otimes a_k |_j > 0 \text{ e } C |_j - M \otimes a_k |_j > 0 \\ M |_j & \text{caso contrário.} \end{cases} \quad \forall j, (j = 1, \dots, m)$$

3.5.4.4 A Modularização de M para o box-de-tempo:

Na rede de Petri estendida Ghenesys, os objetos da classe *atividade* são essenciais no encapsulamento de processos e de subsistemas num modelo de grande porte. Os objetos da subclasse *atividade-composta*, descrita no item 3.5.2, e ilustrada na Figura 25, são compostas por pelo menos três objetos: duas atividades-simples (sendo uma transição de entrada e uma transição de saída) e um box-de-tempo. Um ponteiro *#ponteiro-subrede* que aponta para uma sub-rede dinâmica é incluída na sua lista de atributos.

A rede de Petri estendida Orientada a Objetos Ghenesys é completa, e pode ser simulada em qualquer nível de abstração, desde que uma estimativa da duração de tempo seja fornecida para cada atividade-composta. No processo de detalhamento, o ponteiro para outra subrede deve ser informado e portas internas apropriadas devem ser estabelecidas para sincronizar o comportamento da subrede.

Entretanto, este novo elemento, o box-de-tempo requer uma modificação nas regras de habilitação e de disparo de transições, afastando do formalismo da rede de Petri

segura C/E. Definem-se dois vetores de tempo: o vetor de tempo (global) mínimo requerido \mathbf{T} (em relação à rede Σ) e o vetor de tempo local de simulação \mathbf{t} (em relação a cada box).

O vetor de tempo (mínimo) requerido \mathbf{T} , tem uma dimensão l, m igual ao vetor de marcação \mathbf{M} , e os seus elementos são variáveis inteiras que guardam o tempo mínimo requerido para que os boxes preservem suas marcas.

A segunda modularização de box-de-tempo, em relação à uma transição t_k , é feita após a primeira modularização $\bar{\mathbf{M}}^{tk}$. A segunda modularização é denotada pela função de mapeamento $\eta(\bar{\mathbf{M}}^{tk}, \mathbf{t}) = \bar{\mathbf{M}}^{tk}$ é definida como sendo:

$$\bar{\mathbf{M}}^{tk} |_j = \begin{cases} 0 & \text{se } 0 < \mathbf{t} |_j < \mathbf{T} |_j \\ \bar{\mathbf{M}} |_j & \text{caso contrário.} \end{cases}$$

$\forall j, (j = 1, \dots, m)$

A segunda modularização, portanto, nada mais é do que um filtro que só permite que a marcação da rede $\bar{\mathbf{M}}^{tk}$, em relação à uma transição t_k , seja levada em consideração apenas quando o tempo local de simulação \mathbf{t} for maior do que o tempo mínimo requerido \mathbf{T} .

3.5.5 O Ambiente Computacional HIPER

Neste trabalho, utilizamos o ambiente computacional HIPER (“A High Level Integrated Petri Net Environment for the Design of FMS”) para criar as redes de Petri estendidas Ghenesys que representam o “planning self-model”. O ambiente HIPER é composto por um editor gráfico de redes estendidas Ghenesys, um analisador de propriedades da rede de Petri (ainda em desenvolvimento) e um simulador analítico da rede de Petri. O editor gráfico foi escrito na linguagem C, usando a biblioteca Suit da Universi-

dade de Virginia. Através do editor, pode-se configurar uma rede Ghenesys, com vários níveis interligados de abstração. Exemplos do uso destas redes Ghenesys estão mostradas mais adiante, no capítulo 5. A Figura 27, seguinte, ilustra o editor gráfico do ambiente HIPER.

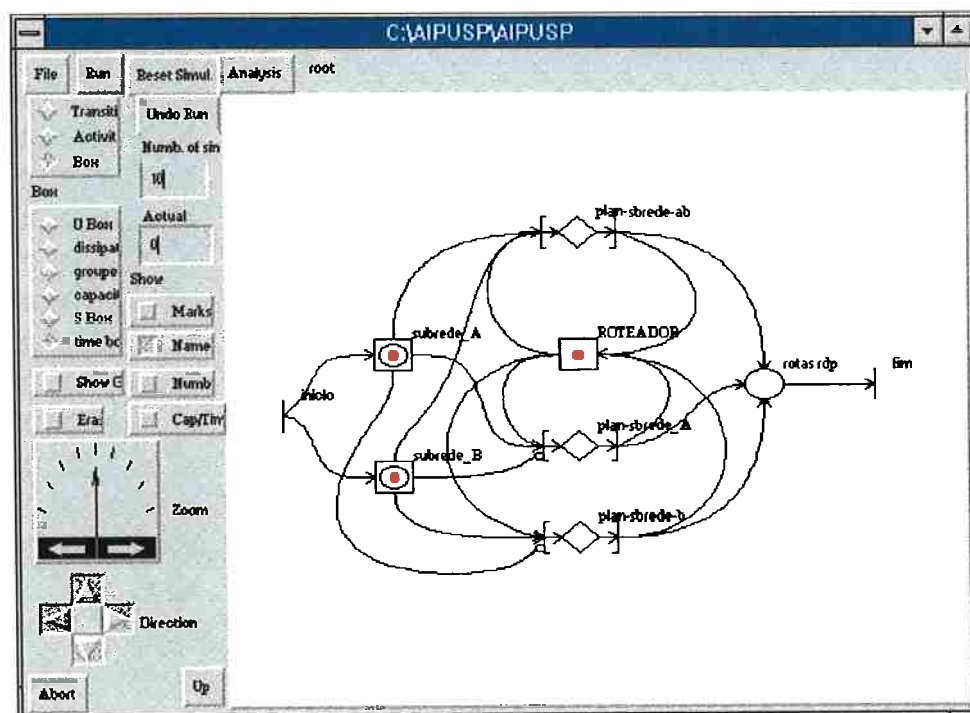


Figura 27 - Editor gráfico do ambiente HIPER

Uma vez que a rede de Petri estendida Ghenesys foi criada, pode-se fazer uma análise de suas propriedades e fazer uma simulação da rede. A implementação do analisador de propriedades está baseada na representação algébrica, descrita no item 3.5.4, acima, que corresponde à rede que foi criada através do editor gráfico. O resultado da análise de propriedades é gravado em arquivo no formato texto.

Além disso, o HIPER apresenta outra funcionalidade, o simulador gráfico. A simulação pode ser usada como uma etapa de validação de um modelo feito em rede de Petri. Este processo consiste em verificar se a evolução de marcas na rede corresponde ao comportamento desejado para o sistema alvo.

A implementação do simulador gráfico foi feita em associação com o editor gráfico. De fato, o simulador utiliza a mesma interface do editor gráfico para mostrar a evolução de marcas. A programação do simulador está inteiramente baseada na equação de estado e em algoritmos algébricos para se determinar a transição e o disparo de transições, como mostramos no item 3.5.4, acima. Atualmente, estão sendo desenvolvidos novos algoritmos para se fazer a análise de invariantes (em andamento) e para verificar o sincronismo estrutural entre transições [Silva et Zanata 1997]⁵.

3.6 A Representação da RdP em Lógica

Apresentamos, acima, os motivos porque a rede de Petri pode constituir um bom formalismo para se estruturar o problema de “planning”. Posto isto, surge a questão de como representar a rede de Petri em lógica uma vez que o objetivo final é tratar o problema de “planning” com IA. A abordagem mais imediata seria a de se fazer o mapeamento direto da RdP para cláusulas de Horn.

[Silva 1992], [Tuomin 1989] apresentam uma abordagem que consiste numa axiomatização completa da rede de Petri em Lógica Dinâmica. A Lógica Dinâmica é uma forma de lógica modal (ou lógica complementar). Para ver mais sobre lógica modal, veja as referências [Hughes et Cresswell 1968] e [Costa 1992]. A Lógica Dinâmica está descrita em [Fischer 1979] e [Harel 1979].

Entretanto, podem surgir algumas “armadilhas” neste processo, principalmente com respeito à marcação (M) da rede de Petri. [Valette et Courvoisier 92] observam que este mapeamento direto da RdP para cláusulas de Horn não pode ser feito, especialmente com respeito à marcação da rede M. Por exemplo, considere transição e_3 na rede de Petri da Figura 21, reproduzida novamente na página seguinte.

⁵ [Silva, J.R., Zanata, S.A. 1997] Análise de Distância Síncrona em Redes de Petri Estendidas. www.mcca.ep.usp.br

Na marcação inicial M_0 , a transição e_3 está habilitada para disparar apenas uma única vez. Mas, não é isto o que ocorre com um provador de teoremas típico usado como mecanismo de inferência para operar no conjunto “equivalente” em cláusulas de Horn. Estas constituem uma forma de implementação da implicação lógica.

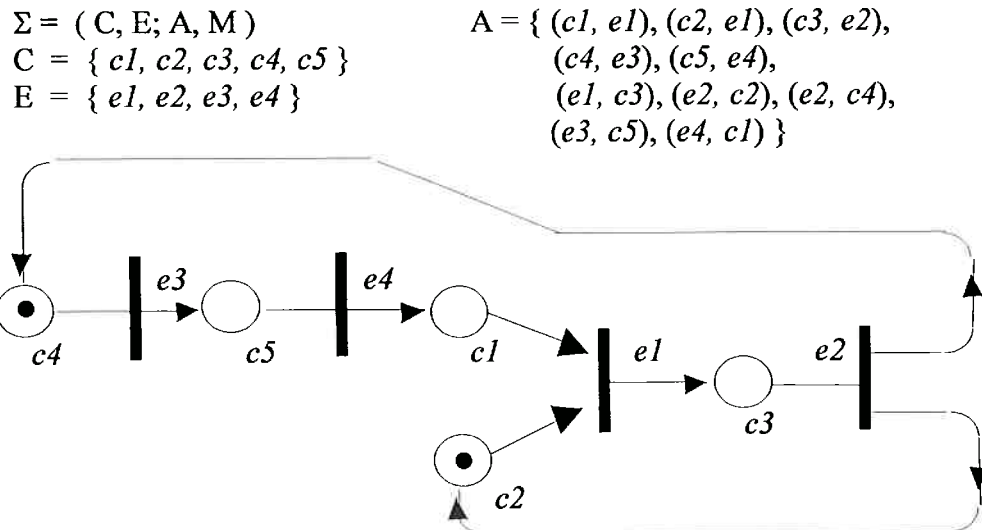


Figura 21 - Exemplo de uma rede de Petri Elementar (RE)

Num mapeamento direto da RdP C/E para cláusulas de Horn, uma transição poderia ser representada como:

- (i) habilita e_i : pós-condições $(e_i^*) \wedge$ pré-condições $(^*e_i)$
 exemplo: habilita e_3 se: $\neg(c_5) \wedge (c_4)$
- (ii) dispara e_i : \neg pós-condições $(\neg e_i^*) \leftarrow$ pré-condições $(^*e_i)$
 exemplo: dispara e_1 se: $(c_5) \leftarrow (c_4)$

A interpretação da sentença (ii) acima é a seguinte: o disparo da transição t ocorre se pré-condições implicam logicamente \neg (pós-condições). Na rede de Petri, o disparo de uma transição retira marcas dos lugares de entrada e, ao mesmo tempo, insere marcas nos lugares de saída. Mas, na representação em cláusula de Horn, acima, o (valor verdade do) modelo de mundo após o disparo da transição t_i apresenta problemas, porque os termos que representam as pré-condições não são modificadas pela implicação lógica. A determinação do valor verdade, que correspondente ao disparo de

uma transição, não tem nenhum efeito de modificar o valor verdade destas pré-condições. Por exemplo, se estas pré-condições representarem recursos, a alocação destes recursos não estaria representada corretamente -- um recurso poderia sumir sem ser utilizado ou um mesmo recurso poderia ser alocado diversas vezes.

Para superar estas deficiências que aparecem no mapeamento da rede de Petri para a lógica clássica, [Valette et Courvoisier 92] propuseram o uso de uma lógica especial, denominada lógica linear. Na lógica linear [Girard 1987] existem operadores especiais como: implicação linear (\multimap), marcação inicial (\multimap), conectivo \wedge (\oplus) e \vee exclusivo ($\&$). Ao contrário da implicação tradicional, a implicação linear (\multimap) consome as cláusulas usadas na dedução. Um fato representado através da marcação inicial (\multimap) só pode ser utilizado uma única vez. Além disso, não existem quantificadores de variáveis (\forall, \exists) e não valem os dois axiomas seguintes da lógica clássica:

- i) o axioma da relaxação (“weakening”) $\frac{a \rightarrow b}{a \wedge c \rightarrow b}$ e
- ii) o axioma da contração (“contraction”) $\frac{a \wedge a \rightarrow b}{a \rightarrow b}$.

Para contornar o problema do mapeamento da RdP que foi descrito acima, não tentamos elaborar um conjunto de cláusulas de Horn “equivalentes” à Rede de Petri. Ao invés, neste trabalho, os elementos da RdP são preservados exatamente como são, i.e., os lugares e as transições da RdP são representadas através de uma estrutura de dados na forma de termos que são gravados na Base de Dados Global do sistema que foi escrito em Visual PROLOG. O mecanismo de inferência (MI) é um ambiente de rede de Petri que atua sobre esta estrutura de dados.

Observe que, apesar de haver semelhanças entre o processo de elaboração de planos em nossa proposta com o mecanismo utilizado nos sistemas que usam a representação STRIPS, a nossa abordagem é bem distinta daquela. Primeiro, porque as transições não consistem de ações primitivas, que não levam em conta o contexto no qual a ação é aplicada, como no sistema STRIPS. Segundo, porque o mecanismo de inferência utilizado no Planejador é um ambiente de Rede de Petri que opera sobre a rede que representa o conhecimento do domínio (“application model”).

CAPÍTULO 4

4 PROPOSTA DA UMA ABORDAGEM HÍBRIDA IA / REDES DE PETRI

Os sistemas de planejamento baseados em Inteligência Artificial (IA), descritos no capítulo 2, utilizam bastante conhecimento implícito e incompleto, o que torna o desenvolvimento de sistemas para resolução deste tipo de problema muito complexo. Para se obter um melhor desempenho computacional destes sistemas, é necessário se adotar mecanismos auxiliares para estruturar o problema -- não só o domínio do problema, mas, principalmente, o problema de “planning” em si. Com este objetivo, usamos a teoria das Redes de Petri (RdP) para a construção de um modelo conceitual do problema, visando a identificação de estratégias de busca que possam auxiliar na solução do problema de planejamento.

Reportando às metodologias de desenvolvimento de software, descritas no capítulo 1, item 1.2, podemos dizer que a nossa proposta de metodologia consiste na construção de um modelo conceitual (*mc*), i.e., de especificações formais usando a teoria da Rede de Petri, para representar:

- (1) o problema de “planning” em si, e
- (2) o domínio do problema.

Podemos descrever nossa proposta como sendo a:

- (1) separação do conhecimento do domínio do meta-conhecimento de “planning”,
- (2) elaboração da representação do domínio do problema usando RdP,
- (3) análise estrutural da rede de Petri que representa o domínio da aplicação,
- (4) elaboração da representação do problema de “planning” usando RdP,
- (5) utilização de toda experiência acumulada, seja para esta classe de problemas ou para este domínio de aplicação, na forma de regras heurísticas.

A metodologia se fundamenta na elaboração de um modelo conceitual (*mc*) usando RdP para representar, separadamente, o problema de *planning* em si e o domínio do problema. O Planejador sempre se reporta a este modelo conceitual em todas as etapas do processo. Por exemplo, o planejador pode usar o meta-conhecimento explicitado pela RdP que representa o problema de “*planning*”, raciocinando acerca dos vários níveis de abstração do problema, por exemplo, para estabelecer submetas para um nível de abstração mais baixo. Por sua vez, a análise da RdP que representa o domínio do problema pode sugerir estratégias de busca para dirigir o processo de solução do problema, por exemplo, para definir quais transições estão habilitadas em determinado instante. Desta forma, com um modelo conceitual, o Planejador tem ao alcance da mão um “mapa rodoviário” que descreve todo o processo de “*planning*”.

Neste trabalho, o modelo conceitual que representa o problema de “*planning*” foi implementado usando o ambiente computacional HIPER [Silva et al 1996].

O ambiente HIPER foi usado para elaborar os diversos níveis hierárquicos de abstração da rede de Petri que representa o problema de “*planning*”. Este *mc* para o problema de “*planning*” pode ser visto como uma representação *do meta-planejamento*, ou seja, plano para se construir o plano. Meta-planejamento pode ser visto como descrição ou axiomatização do processo de planejamento. Na prática, a axiomatização de meta-planos serve para descrever as diversas estratégias de planejamento existentes para se construir o plano para o domínio do problema.

Por sua vez, o modelo conceitual que representa o domínio do problema também foi elaborado usando rede de Petri estendida Orientada a Objetos Ghensys (veja o problema do Roteador de Helicópteros no cap. 5). Note que a rede Ghensys se reduz a uma rede de Petri Elementar (veja exemplo do Mundo de Blocos), no nível de abstração “*ground*”, eliminando-se todos os seus elementos compostos -- *box-composto*, *box-capacidade* e *atividade-composta*.

4.1 - O problema da Representação no “Planning”

A habilidade de atuar em ambientes dinâmicos é crucial para a sobrevivência de todos os seres vivos. Nas formas de vida superior, existe a capacidade de *prever* (antecipar) *o futuro* e de *elaborar planos de ação* para o atingimento de suas metas. Deste modo, o estudo de *ações* e de *planos* é fundamental no desenvolvimento de sistemas inteligentes que sejam capazes de lidar eficazmente com os problemas do mundo real.

No processo de planejamento, a decisão do planejador acerca de qual curso de ação tomar é escolhida a partir de um vasto repertório de possibilidades. Esta decisão, por sua vez pode influenciar os estados do mundo de maneiras diversas e complicadas. Tudo isto é realizado num mundo complexo e dinâmico em que o meio-ambiente está em mudança contínua. Para se elaborar um sistema de planning baseado em IA, é necessário um modelo de mundo e de um modelo de ação para atuar neste modelo de mundo.

[Georgeff 87] apresenta os conceitos necessários para representar o modelo de ação. Nesta abordagem, o mundo estaria num dos seus *estados* ou *situações* (ou contexto) potenciais. O *estado* pode ser visto como uma foto instantânea do mundo num determinado instante de tempo. Uma seqüência de estados é dito *comportamento*. O comportamento pode representar uma história passada ou, então um potencial comportamento futuro desejado para o sistema (ou seja, um plano).

A transição entre os estados do mundo ocorre devido a um evento ou uma ação.

Evento: conjunto de comportamentos ou uma seqüência de estados.

Ação: tipo especial de evento efetuado por um agente de maneira intencional.

Uma questão importante nos sistemas de “planning” é a manutenção da verdade. A manutenção da verdade no mundo real, complexo e dinâmico, requer um modelo de ação mais estruturado. Para isto são definidos as relações contextuais e o conceito de “fluent”. As relações contextuais são representadas pelo par *verdade(A,s)* - A é verda-

deiro no estado s . O conceito de "fluent" pode representar objetos e suas instâncias, propriedades associadas a objetos, relações entre objetos e funções de objetos.

4.1.1 - O Cálculo Contextual

No Cálculo Contextual [McCarthy et Hayes 1969] são usados predicados para se fazer afirmação acerca dos valores de "fluents" em estados particulares. A sua notação é feita como $(f,s) \equiv$ "fluent" f é verdadeiro no estado s . Exemplo:

$\text{verdade}(\text{sobre}(A,B),s)$ - a relação bloco A está sobre bloco B é verdadeira no estado s .

As fórmulas bem formadas no cálculo contextual podem incluir conectores lógicos e quantificadores. Exemplo:

$$\forall s \text{ verdade}(\text{livre}(A, s) \wedge \text{livre}(B, s)) \rightarrow \text{verdade}(\text{resulta}(\text{mover}(A,B), s) \rightarrow \text{verdade}(\text{sobre}(A,B), s'))$$

que representa que: para todo estado s em que for verdadeiro que bloco A está livre no estado s e o bloco B está livre no estado s , é verdadeira a relação bloco A está sobre o bloco B no estado atingível (digamos, s') a partir do estado inicial s que resulta da aplicação da ação de colocar o bloco A sobre o bloco B .

O problema do baixo desempenho computacional apresentado pelo cálculo contextual está no "frame problem"[Hayes 1973]. A especificação dos efeitos diretos da aplicação de uma ação não é problemática nos sistemas de "planning". O problema está na especificação do que pode ou não ser afetado indiretamente como efeito da aplicação de uma determinada ação. O "frame problem" decorre da necessidade de se escrever uma grande quantidade de axiomas "frame", semelhantes uns aos outros, para garantir que cada propriedade dos objetos no modelo de mundo que não é afetado pela aplicação de uma determinada ação que modifica o modelo de mundo. O "frame problem" pode ser sintetizado como sendo o problema de se encontrar coleções adequadas de leis de movimento entre dois estados do modelo de mundo.

Portanto, para que o planejador mantenha sempre um passo firme em direção ao atingimento das metas e “não perca a floresta pelas árvores” é preciso prover uma forma de estruturação para o problema. Mais adiante, no item 4.3, mostramos como o uso da estruturação do problema usando grafos é capaz de contextualizar corretamente o efeito da aplicação de uma ação.

4.2 Crítica da representação STRIPS

A maioria dos sistemas de planejamento baseados em IA utiliza uma representação do tipo STRIPS [Fikes et Nilsson 1971], como foi descrito no capítulo 2, item 2.1.2.1. Nesta representação, a descrição de uma ação apresenta: uma pré-condição, uma lista de adição e uma lista de eliminação. Os sistemas aplicam ações para modificar o estado do mundo se suas pré-condições estiverem verdadeiras. O efeito da aplicação de uma ação está expresso nas listas de adição e nas listas de eliminação.

Entretanto, esta forma de representação pioneira apresenta problemas, como explicitamos na crítica que fizemos no item 2.1.2.2, sendo a sua principal deficiência o fato de não considerar a dependência entre metas concorrentes. Por este motivo, um sistema que emprega este tipo de representação não é capaz de contextualizar corretamente o efeito da aplicação de uma ação. Uma consequência importante desta deficiência é a Anomalia de Sussmann. Na anomalia de Sussman [Sussman 1974] o atingimento de algumas metas pode requerer o estabelecimento de submetas para satisfazer pré-condições que irão desfazer cláusulas do estado meta anteriormente já verdadeiras. Deste modo, o programa entra num processo de "looping" sem progresso em direção ao estado meta desejado.

No histórico dos sistemas de planejamento baseados em IA que apresentamos no capítulo 2, mostramos a estruturação do problema de “planning” sempre foi buscada, de diferentes maneiras, entretanto, sem promover uma ruptura com os sistemas pioneiros no que diz respeito ao uso da representação STRIPS para o modelo de ação. É claro que, surgiram diversas contribuições nos diferentes sistemas pioneiros ou clássicos -- prote-

ção de metas já atingidas, críticas para detectar conflito, regressão de metas, hipótese do mundo fechado, ... -- que tentaram sanar esta e outras deficiências. Até mesmo, nos sistema que integram planejamento e execução, são necessários os diferentes mecanismos propostos -- tabela triângulo, árvore de submetas e grafo de decisão, ... -- tratam da questão de se contextualizar os efeitos da aplicação de uma determinada ação definida no plano. Mostramos, a seguir, como a análise de uma estrutura simples baseada em grafos pode ajudar a superar esta dificuldade.

4.3 - Uma estrutura simples baseada em grafos

A teoria das Redes de Petri (RdP) é aplicada tradicionalmente no problema de controle de sistemas discretos. A Rede de Petri (RdP) permite a representação formal dos estados e da evolução de um sistema de eventos discretos. Permite explicitar relações entre estes estados, por ex. dependência de ordem parcial entre estados, a existência de eventos concorrentes ou excludentes, compartilhamento de recursos, etc. Uma descrição de diferentes tipos de Rede de Petri foi feita no capítulo 3.

A análise estrutural de uma RdP pode prover informação importante para uma abordagem não-procedimental (baseada em IA) em sistemas de eventos discretos. O conhecimento sobre a estrutura do problema pode subsidiar formas de busca mais eficazes, e a minimização do "backtracking".

[Shimad 94] mostra uma abordagem para se fazer a análise das estrutura da rede de Petri para obter meta-conhecimento necessário para dirigir o processo de "planning". Esta abordagem é mostrada através do problema exemplo clássico: o planejamento de ações no conhecido exemplo do Mundo dos Blocos. [Shimad 94] justifica a escolha deste exemplo pela simplicidade do problema em si em contraposição com o fato de apresentar todas as características que dificultam a automação do planejamento.

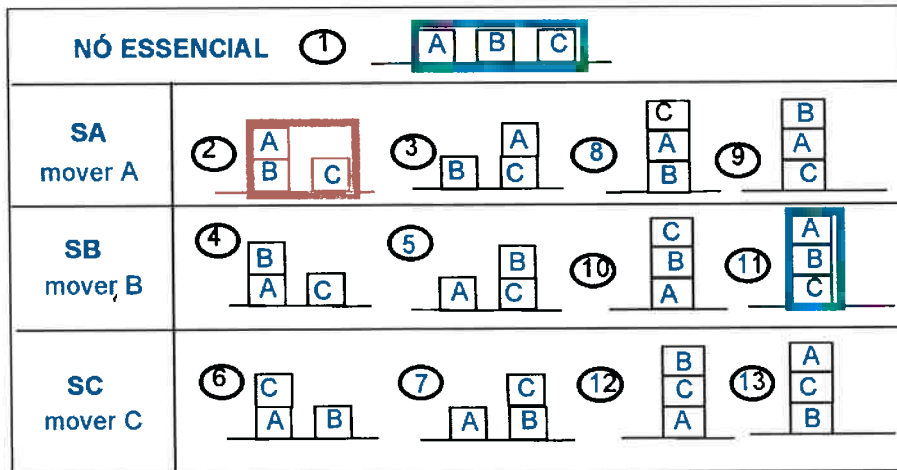


Figura 28 - Enumeração Completa dos Estados do Mundo de Blocos

4.3.1 - O Exemplo do Mundo de Blocos

Seja uma instância particular do Mundo de Blocos, ilustrado na Figura 4 e descrito no item 2.1.2, na página 20. Seja o estado inicial onde os três blocos estão distribuídos aleatoriamente sobre uma "mesa". A Figura 28, acima, ilustra a enumeração completa dos estados do sistema para este problema (para 3 blocos).

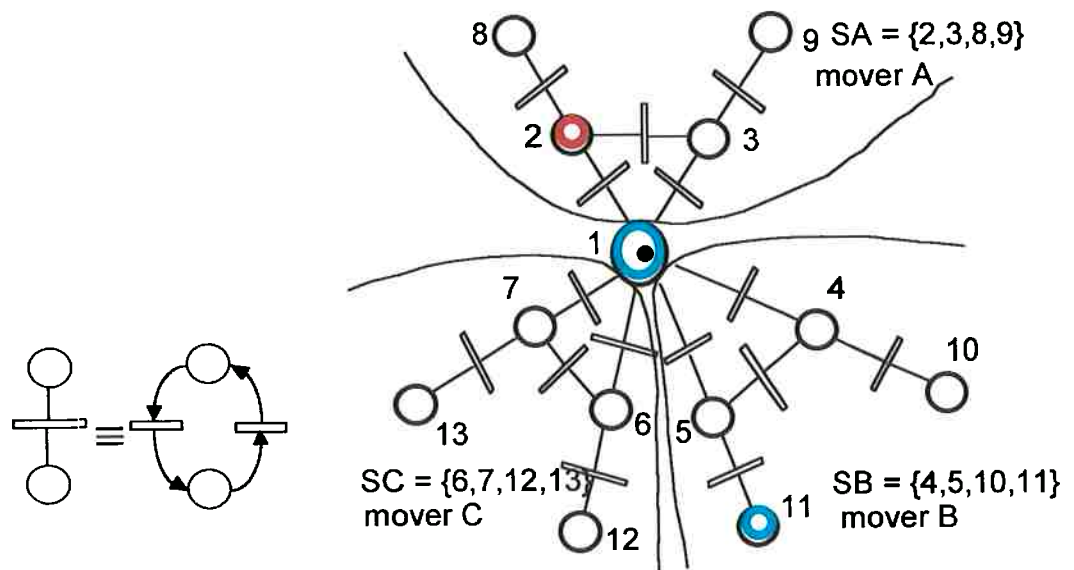


Figura 29 - "Application model" usando RdP C/E para o Mundo de Blocos

A Figura 29, na página anterior, mostra a representação em RdP do tipo C/E que constitui o modelo do domínio da aplicação (“application model”) para este problema. A Figura 29 mostra que todos os estados podem ser atingidos a partir do estado inicial designado aqui de “nó essencial”, em que todos os blocos estão sobre a mesa.

Além disso, podemos identificar, na Fig. 29, três componentes conectadas ou subredes (SA, SB e SC). Estas subredes definem agrupamentos de ações inter-relacionadas (interdependentes) pela causalidade. Estas subredes estão interligadas através do estado 1, ou seja, o “nó essencial”^{*1}. Desta forma, analisando a estrutura da RdP, podemos efetuar uma decomposição do problema em subproblemas que são independentes, de uma maneira tal que:

- 1) o estado onde todos os blocos estão sobre a mesa é um *nó essencial* (NE) do grafo.
- 2) o primeiro bloco movido a partir do nó essencial determina uma sucessão de estados dependentes desta ação que estão na mesma componente conectada.
- 3) existe sempre um plano que corresponde à distância mínima entre dois estados que pertencem à uma mesma componente conectada.
- 4) o plano envolvendo dois estados em duas componentes conectadas diferentes deve passar necessariamente pelo restabelecimento do nó essencial.

O conhecimento obtido a partir de uma análise da Rede de Petri que representa o domínio para este problema é muito importante. Como resultado da análise da RdP que representa o “application model”, podemos sugerir o seguinte algoritmo para resolver este problema de planejamento :

Sejam dados EI = estado inicial, M = conjunção de metas e NE = nó essencial.

^{*1} “Nó essencial” (NE) de um grafo: é o estado singular da RdP (não fortemente conectada) a partir da qual são derivadas as subredes formadas por transições que são interdependentes (Veja definição de transição independente no item 3.1.1). Por este motivo, é uma boa estratégia restabelecer o NE para se atingir um estado meta que não pertence à sub-rede do estado corrente. O NE constitui um candidato natural para se fazer uma decomposição estrutural do problema de “planning”. Então, nesse caso, a solução de problema de “planning” envolvendo estados localizados em subredes distantes requer o restabelecimento do NE. Sobre decomposição estrutural, veja [Mädler 1992], descrito no capítulo 2, item 2.5.

1) SE M já é verdadeiro ($EI = M$) ENTÃO fim.

Ex. $EI = (\text{sobre}(a\ b) \wedge \text{sobre}(b\ c) \wedge \text{sobre}(c\ \text{mesa}) \wedge \text{livre}(a))$

$M = (\text{sobre}(a\ b) \wedge \text{sobre}(b\ c) \wedge \text{sobre}(c\ \text{mesa}) \wedge \text{livre}(a))$

2) SE EI e M estão na mesma componente conectada ENTÃO \exists caminho entre EI e M sem passar pelo nó essencial. Resolva M usando mecanismo do tipo “meios-fins”, obtendo o plano final P .

3) Caso Contrário, gere uma submeta $NE =$ nó essencial obtendo um sub-plano Parcial-1. À seguir, resolva M a partir do nó essencial (NE) usando mecanismo do tipo “meios-fins” obtendo um sub-plano Parcial-2. O plano final resulta da concatenação dos planos parciais $P = \text{Parcial-1} + \text{Parcial-2}$.

O algoritmo usa o "conhecimento" obtido a partir de uma análise da Rede de Petri que representa o conhecimento do domínio (o domain-K) -- na verdade, um meta-conhecimento -- e é capaz de evitar o problema da anomalia de Sussman*². Este algoritmo faz a decomposição do problema baseado na análise da estrutura da rede. Este meta-conhecimento está baseado na noção de transições independentes para assegurar a independência de cada uma das subredes e na existência do “nó essencial” que faz a conexão entre todas estas subredes. A interdependência pela causalidade ocorre apenas entre ações que pertencem à mesma componente conectada do grafo.

A Figura 30, na página seguinte, ilustra uma representação do algoritmo acima em rede de Petri estendida Ghensys e representa o “planning self-model” para este problema do Mundo de Blocos.

Na Figura 30, podemos ver que o passo 1 está representado pela transição “ $EI=M$ ”. O disparo desta transição retira marcas dos boxes EI e M e introduz uma marca no box plano-final.

*² Seja a meta: $ATINJA(\text{sobre}(a\ b) \wedge \text{sobre}(b\ c) \wedge \text{sobre}(c\ \text{mesa}) \wedge \text{livre}(a))$ que corresponde ao lugar 11 na Fig. 29, a partir do estado $((\text{sobre}(a\ \text{mesa}) \wedge \text{sobre}(b\ \text{mesa}) \wedge \text{sobre}(c\ \text{mesa}) \wedge \text{livre}(a) \wedge \text{livre}(b) \wedge \text{livre}(c))$ que corresponde ao lugar 1 na Fig. 29. A Anomalia de Sussman ocorre como decorrência da escolha (usando-se a estratégia meios-fins) da ação que corresponde ao disparo da transição que leva ao estado $(\text{sobre}(a\ b) \wedge \text{sobre}(b\ \text{mesa}) \wedge \text{sobre}(c\ \text{mesa}) \wedge \text{livre}(a) \wedge \text{livre}(c))$ que corresponde ao lugar 2 na Fig. 29. Note que o disparo desta transição, na realidade, AUMENTA a distância em termos de número de disparo de transições para se atingir o estado meta.

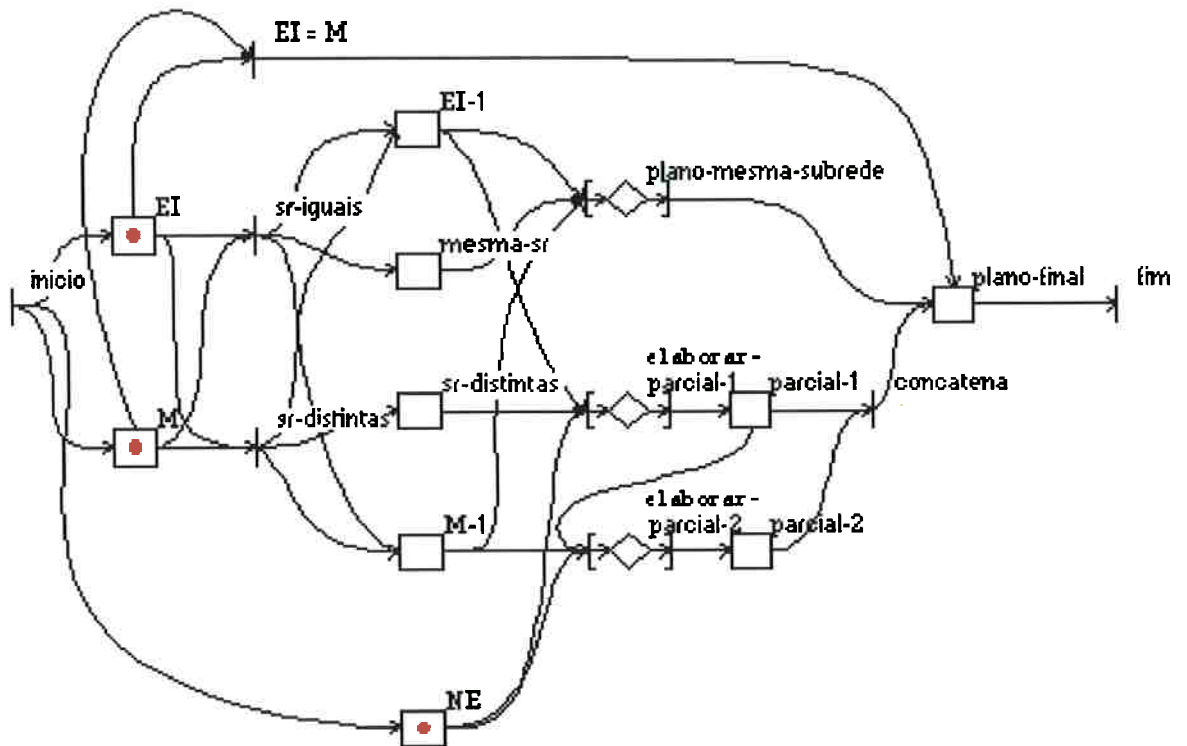


Figura 30 - “Planning self-model” usando rede Ghenesys para Mundo de Blocos

O passo 2 está representado pela transição “sr-iguais”. O disparo desta transição retira marcas dos boxes EI e M e introduz marcas nos boxes “mesma-sr”, EI-1 e M-1. Uma marca no box “mesma-sr” indica que o plano deve ser obtido pela atividade [plano-mesma-subrede]. O estado meta M-1 pode ser alcançado a partir de EI-1 usando um mecanismo do tipo “meios-fins”, obtendo o plano final P.

O passo 3 está representado pela transição “sr-distintas”. O disparo desta transição retira marcas dos boxes EI e M e introduz marcas nos boxes “sr-distintas”, EI-1 e M-1. Uma marca no box “sr-distintas” indica que o plano deve ser obtido através de dois sub-problemas: [elaborar-parcial-1] e [elaborar-parcial-2]. O Plano parcial-1 é obtido para EI-1 como estado inicial e NE como estado meta, e o plano parcial-2 é obtido para NE como estado inicial e M-1 como estado meta. O plano final resulta da concatenação dos planos parcial-1 e parcial-2. Observe que os subproblemas ficam confinados dentro dos limites de cada uma das três subredes.

4.3.2 O Construtor de Planos para o Mundo de Blocos

A rede Ghensys que representa o “planning model” é hierárquica, No nível mais baixo de abstração, localizamos o Construtor de Planos, que é um ambiente de Rede de Petri que opera sobre a rede que representa o conhecimento do domínio.

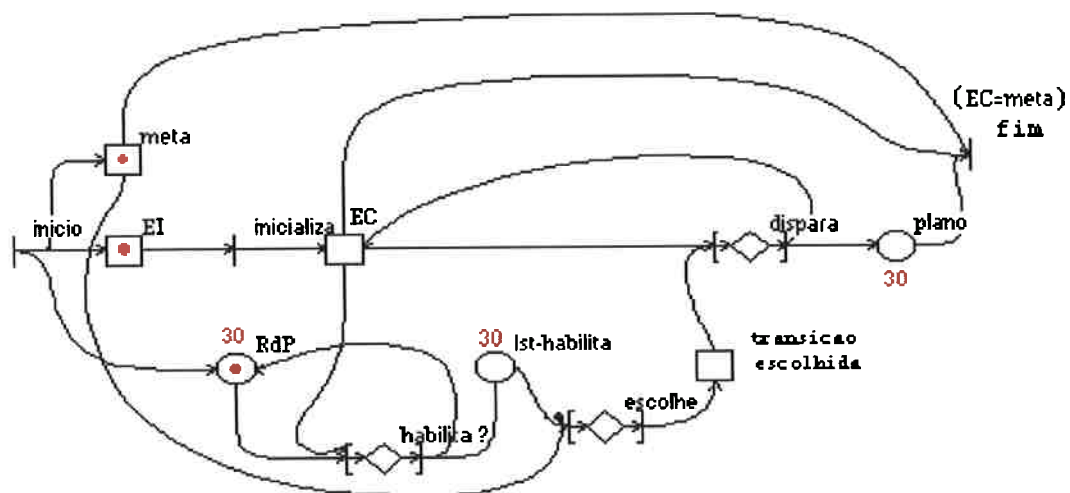


Figura 31 - O Construtor de Planos para o Mundo de Blocos

A Figura 31, acima, ilustra o Construtor de Planos para o Mundo de Blocos. As atividades de elaborar o [plano-mesma-subrede], de [elaborar-parcial-1] e de [elaborar-parcial-2], ilustradas na Figura 30, usam o mesmo mecanismo de inferência, ou seja, o mesmo Construtor de Planos mostrado na Figura 31.

A transição *inicia* faz a inicialização das marcas para os boxes: *meta*, *EI* (estado inicial) e a *RdP* (o “application model”). As marcas no box *RdP* têm atributos que identificam cada transição do “application model”. A transição *inicializa* torna o *EC* (estado corrente) igual ao *EI*.

O Construtor de planos é um ambiente de Rede de Petri que opera sobre a rede que representa o conhecimento do domínio. O plano é construído incrementalmente e, em cada passo, é escolhida uma transição habilitada que modifica o estado corrente (*EC*), até que o estado *meta* seja atingido. A transição *fim* é habilitada se houver marcas

no box *plano* e o atributo, que indica o estado do mundo, da marca no box *EC* é igual ao atributo da marca no box *meta* (ou seja, $EC=meta$).

Num passo de planejamento típico, existindo marcas nos boxes *RdP* e *EC*, ocorre a habilitação e o disparo da atividade *habilita_?* que insere n marcas no box *lst_habilita* e devolve as marcas no box *RdP*. Então, pode ocorrer a habilitação e o disparo da atividade *escolhe**³, que leva em conta (a existência e) os atributos da marca no box *meta* -- através de uma porta habilitadora -- e insere uma marca no box *transição_escolhida*. O atributo da marca no box *meta* pode ser usado na heurística para escolha da próxima transição a ser disparada. Existindo marcas nos boxes *EC* e *transição_escolhida*, ocorre a habilitação e o disparo da atividade *dispara* que insere uma marca nos boxes *EC* e *plano*. Note que o atributo da nova marca inserida no box *EC* é diverso daquele que foi usado como pré-condição da atividade *dispara*. Então, inicia-se um novo passo.

Neste trabalho, os elementos da RdP (do “application model”) são preservados exatamente como são, i.e., os lugares e as transições da RdP são representadas através de uma estrutura de dados na forma de termos que são gravados na Base de Dados Global do sistema que foi escrito em Visual PROLOG. O mecanismo de inferência (MI) é um ambiente de rede de Petri que atua sobre esta estrutura de dados.

Observe que, apesar de haver semelhanças entre o processo de elaboração de planos em nossa proposta com o mecanismo utilizado nos sistemas que usam a representação STRIPS, a nossa abordagem é bem distinta daquela. Primeiro, porque as transições não consistem de ações primitivas, que não levam em conta o contexto no qual a ação é aplicada, como no sistema STRIPS. Segundo, porque o mecanismo de inferência utilizado no Planejador é um ambiente de Rede de Petri que opera sobre a rede que representa o conhecimento do domínio (“application model”).

*³ A regra heurística associada com a transição *escolhe* pode ser do tipo “meios-fins”, sem que exista a necessidade de se fazer “backtracking” e nem ocorra o problema da Anomalia de Sussman, porque o subproblema definido pelas subredes é linear (para o problema de 3 blocos). Para ver mais estratégias, veja o item 4.7.1, mais adiante.

4.4 - O Modelo Conceitual (mc):

A nossa proposta de estruturação está fundamentada no uso de especificações formais usando a teoria da Rede de Petri. A estruturação proposta faz uma separação clara dos tipos de conhecimento existentes num problema de “planning”: o conhecimento do domínio da aplicação e o meta-conhecimento para dirigir o processo de busca de soluções.

Um modelo conceitual (mc) é construído para separar estes tipos de conhecimentos e para representar: (1) o problema de “planning” em si (o “planning self-model”), e (2) o domínio do problema (o “application model”). O “planning model” pode ser visto como uma representação *do meta-planejamento*, ou seja, plano para se construir o plano. Este meta-conhecimento do problema de “planning” serve para descrever as diversas estratégias de planejamento existentes para se construir o plano para o domínio do problema. O “application model” representa o domínio do problema e não corresponde a nenhum plano em particular mas, sim, corresponde à combinação de todos os planos factíveis ou não que constituem soluções para o problema representado

No exemplo descrito no item 4.3, utilizamos a rede de Petri estendida Ghenesys para o elaborar o “planning self-model” que representa o meta-conhecimento, i.e., as estratégias de busca de soluções, como mostra a Figura 30, e para representar o “application model”^{*4} para representar o domínio da aplicação do problema, como mostra a Figura 29. Estas redes de Petri constituem um “mapa rodoviário” ao alcance da mão do Planejador e descrevem todo o processo de “planning”.

O modelo conceitual que consiste das redes de Petri, expressa vários tipos de conhecimento necessários para a suportar decisões no processo de elaboração do planeja-

^{*4} Note que a rede Ghenesys se reduz a uma rede de Petri Elementar, no nível de abstração “ground”, eliminando-se todos os seus elementos compostos -- box-composto, box-capacidade e atividade-composta.

mento. Podemos classificar este conhecimento em três classes: conhecimento da estratégia, conhecimento do domínio do problema e conhecimento de controle.

O *conhecimento da estratégia* consiste do meta-conhecimento para dirigir o processo de busca das soluções. Este é representado na RdP estendida Ghensys que representa o problema de "planning" em si, caracterizando, portanto, um modelo do sistema de si mesmo ("self model") do sistema de "planning".

O *conhecimento do domínio* consiste de aspectos "salientes" do domínio do problema. Este conhecimento é expresso pelos elementos e pela estrutura da rede de Petri. Por exemplo, no caso da seqüenciação de um SFF, os elementos fundamentais do problema seriam a linha de produtos produzidos, a demanda, os recursos produtivos e os estoques de produtos acabados. Estes constituem um estado ou um "caso" representando marcações dos elementos fundamentais do sistema e que corresponde a um estado do sistema. As limitações tecnológicas para a produção de cada linha de produtos, como necessidade de máquinas, tempo de processamento e ordem parcial entre operações, seriam explicitadas na modelagem do problema em RdP.

O conhecimento da estratégia pode ser obtido a partir da análise da estrutura da RdP que representa o domínio do problema. Nesta análise, é importante observar se a RdP tem a forma de árvore ou de rede (i.e., se contém ou não processos iterativos), a existência de conflitos, a existência de condições necessárias para a ocorrência de "deadlocks", etc. Existem construções típicas na RdP, como p. exemplo, para representar operações excludentes que compartilham recursos -- conflitos -- às quais devem ser associadas regras de escolha. Além disso, deve-se fazer uma análise das propriedades da RdP, descritas no item 3.3.

Este conhecimento pode, e deve, ser ainda associado às técnicas de busca da usadas na IA e na teoria da seqüenciação aplicadas na engenharia de produção [Baker 1974], [Kusiak 1990] para a elaboração de sistemas mais inteligentes. O conhecimento da teoria da seqüenciação deve ser empregado, sempre que possível, através da reutili-

zação de algoritmos e regras de despacho extensivamente provadas na gestão do chão-de-fábrica.

O *conhecimento de controle* constitui a essência de todo Sistema por Eventos Discretos (SED). Ele não deriva dos conhecimentos anteriores. Na rede de Petri, toda a dinâmica dos processos representados é expressa naturalmente. Além disso, esta dinâmica pode ser controlado pelo planejador. Por este motivo, temos adicionalmente um modelo conceitual dinâmico.*⁵ A dinâmica é expressa através da evolução de marcações que descrevem estados na rede de Petri: o estado inicial, o estado corrente e o estado meta. O resultado desta evolução das marcas, no caso de o atingimento do estado meta ser um sucesso, constitui um plano. Chamamos este processo de planejamento por simulação ou simulação do plano.

Observe que estas três classes de conhecimento que derivam do modelo conceitual é característico da RdP e, não poderiam ser obtidas nem através de uma enumeração completa dos estados do problema, ilustrada na Figura 28, e nem através de uma análise cuidadosa do operador STRIPS. No item 4.7.1, mostramos resultados adicionais, derivados do modelo conceitual, que foram obtidos para o problema do Mundo de Blocos. No Apêndice II, mostramos como é possível se resolver o problema generalizado para qualquer número de blocos.

O modelo conceitual deve ser a mais simples possível, representando estritamente a essência do problema e desprezando-se os detalhes sem importância. Idealmente a RdP usada deve ser uma máquina de estados. Caso isto não seja possível, pode-se tentar usar uma rede segura do tipo C/E. À medida que aumenta a complexidade do sistema, o tamanho das redes "flat" torna-se muito grande, dificultando sua análise. Neste caso, recomenda-se o uso das redes de Petri "dobradas", por exemplo, como são as redes estendidas Ghensys [Silva et al 1996], o E-MFG [Santos 1993], [Santos 1993a], ou redes de alto nível [Jensen 1996].

*⁵ Neste trabalho, não utilizamos a dinâmica da Rede de Petri, utilizamos apenas a estrutura da RdP para fazer a estruturação do problema de "planning".

Evidentemente, para problemas reais em SFF, uma representação em RdP do tipo C/E apresenta o inconveniente tornar-se muito grande quando o sistema a ser modelado é grande. Devemos esperar que problemas reais resultem em grafos muito extensos onde a capacidade de análise torna-se difícil devido a sua complexidade. Mesmo em sistemas flexíveis para pequena e média empresa, e portanto menores, uma análise estrutural é proibitiva. [Silva et Shimad 1995] ilustram um exemplo de aplicação para SFF, em que foi usada a rede de Petri estendida Ghensys -- que é uma rede mais sintética. Descrevemos esta aplicação para “Scheduling Job Shop” no apêndice I.

4.5 - A Descrição da Proposta

Propomos uma metodologia para fazer a estruturação e o desenvolvimento de problemas de planejamento, baseado em IA e nas Redes de Petri [Shimad 1994], [Silva et Shimad 1995]. Esta proposta está fundamentada na elaboração de um *modelo conceitual* (mc) baseado em Rede de Petri, como descreve o item 4.3 acima. O nosso enfoque é o de se usar abstração e introduzir técnicas de modelagem mais apuradas na representação do domínio. A mesma hierarquia serve para representar o domínio e também pode, se for o caso, representar o plano.

Lembrando que a Hipótese do Mundo Fechado, descrita no cap. 1, item 1.4, só requer a existência de um modelo lógico do domínio, i.e., um conjunto de axiomas para representar o domínio. Afirmamos, sem demonstrar, que se existe um modelo lógico do domínio, então, também existe um modelo (estrutural) em Rede de Petri, pelo menos a nível abstrato.

O Planejador sempre se reporta a este modelo conceitual em todas as etapas do processo. Por exemplo, o planejador pode usar o meta-conhecimento explicitado pela RdP que representa o problema de “planning”, raciocinando acerca dos vários níveis de abstração do problema para estabelecer submetas para um nível de abstração mais baixo. Por sua vez, a análise da RdP que representa o domínio do problema pode sugerir estratégias de busca para dirigir o processo de solução do problema ou para definir quais

transições estão habilitadas em determinado instante. Espera-se que a análise da rede de Petri, que neste trabalho é feita pelo modelador humano, possa ser automatizada através de análise de propriedades (estruturais) da rede.

4.5.1 - A Metodologia de Estruturação do Problema de “Planning”:

- PASSO1: abordar do problema como um sistema por eventos discretos, identificando os seus elementos determinantes dos possíveis estados do sistema. Separar claramente os tipos de conhecimento existentes no problema; identificando o que é conhecimento do domínio e o que é meta-conhecimento do processo de “planning”.
- PASSO2: elaborar o “application model” do *modelo conceitual* (mc), que representa o domínio do problema usando Rede de Petri. Tentar usar inicialmente tipos de RdP mais simples, recorrendo às RdP mais complexas quando o domínio do problema apresentar maior complexidade, exigindo uso de RdP de alto nível.
- PASSO3: analisar a estrutura da RdP elaborada no PASSO2 acima para obter estratégias para atingir as soluções desejadas, i.e., para obter o meta-conhecimento para estruturar o problema de “planning” em questão e, levando em conta os resultados desta análise, fazer a decomposição estrutural do problema em subproblemas independentes*⁶ (se possível).
- PASSO4: elaborar o “planning self-model” do *modelo conceitual* (mc), que representa o problema de “planning” acima, levando em conta os resultados obtidos no PASSO3. O nível mais baixo de abstração do “planning self-model” deve incluir uma representação do Construtor de Planos, que é um ambiente de Rede de Petri que opera sobre o “application model”.
- PASSO5: implementar o sistema modelado nos passos PASSO2 e PASSO4 usando uma linguagem de programação simbólica, por exemplo o Visual PROLOG.
- PASSO6: validação da solução implementada fazendo testes usando exemplos cujos resultados sejam conhecidos ou possam ser verificados.

*⁶ Veja definição da noção de *independência*, no item 3.1.1.

A metodologia sempre inicia com a separação dos tipos de conhecimento (PASSO1) e com a elaboração do “application model”, que representa o domínio do problema (PASSO2). Por exemplo, no caso do Mundo de Blocos, o “application model” está ilustrado na Figura 29, na página 108. Note que é possível representar o domínio do problema sem, necessariamente, enumerar todos os estados, como foi feito no exemplo do Mundo de Blocos. De fato, a Hipótese de Mundo Fechado, descrita no item 1.4, só requer a existência de um modelo lógico do domínio, ou seja um conjunto de axiomas.

O algoritmo na página 110 ilustra um possível resultado da análise da estrutura da RdP (PASSO3). A Figura 30, na página 111, mostra a RdP estendida Ghensys que representa o “planning self-model” para o Mundo de Blocos (PASSO4). A implementação do sistema (PASSO5) foi feita em linguagem Visual PROLOG. Alguns resultados obtidos no teste do sistema são apresentados nos itens 4.6 e 4.7, mais adiante.

A Figura 32, seguinte, ilustra a estrutura do problema de “planning” segundo nossa proposta.

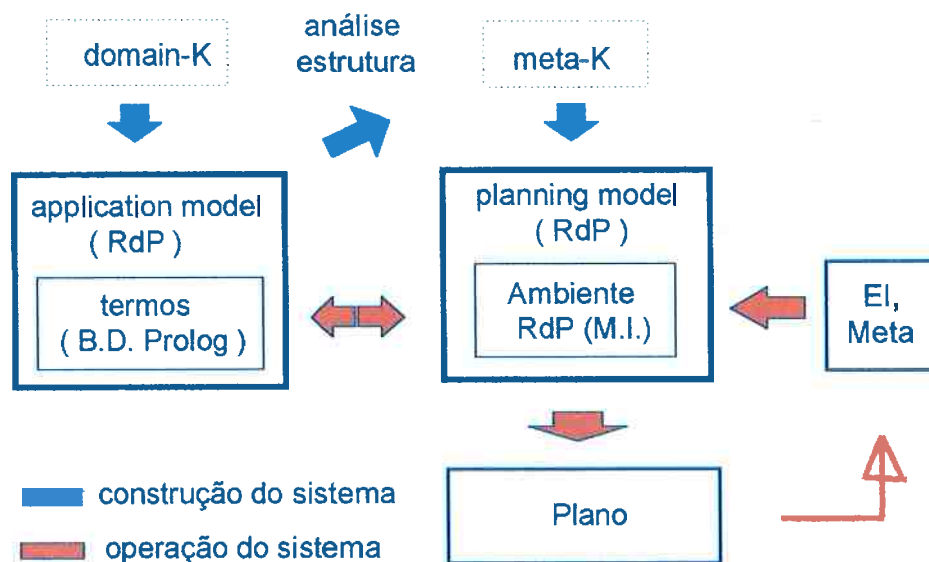


Figura 32 - Estrutura do Problema de “Planning”

A abordagem proposta é hierárquica, de forma que o esquema mostrado na Figura 32 é válido para qualquer nível de abstração. Um plano é elaborado para cada nível de abstração. A solução para o nível de abstração mais alto torna-se meta para o nível de abstração mais baixo.

Entretanto, ao invés de representar apenas o plano através da abstração hierárquica, em nossa proposta, tem-se uma hierarquia (de modelos) do domínio da aplicação “application model”. Para cada nível de abstração existe uma rede de Petri para representar o conhecimento do domínio (ou domain-K, o “application model”). Neste trabalho, os elementos da RdP que representa o domain-K são preservados exatamente como são, i.e., os lugares e as transições da RdP são representadas através de uma estrutura de dados na forma de termos que são gravados na Base de Dados Global do sistema que foi escrito em Visual PROLOG. Este modelo conceitual tem o papel de promover um “fechamento” sobre o domínio do problema, de forma que, todos os seus estados potenciais são capturados.

O “planning model” é um Ambiente de Rede de Petri que atua sobre a rede que representa o domain-K. O nível mais baixo do “planning self-model” deve incluir uma representação do Construtor de Planos, que é um ambiente de Redes de Petri que opera sobre a rede que representa o conhecimento do domínio (ou domain-K).

Note que apesar da semelhança deste processo de disparo da transição usado no nosso trabalho com o mecanismo utilizado na representação STRIPS, a nossa representação é bem distinta. O Construtor de Planos do “planning model” é o mesmo qualquer que seja o nível de abstração do “application model” tratado.

A abordagem proposta pode ser reaplicada para outro domínio de aplicação qualquer -- ela não depende do domínio da aplicação em nenhuma de suas etapas. O Construtor de Planos, que é um ambiente de Rede de Petri, opera a rede que representa o domain-K, e não sobre a sua interpretação para um determinado domínio da aplicação.

4.6 - O Uso da Análise de Propriedades da RdP

Vamos considerar novamente o exemplo do Mundo de Blocos, descrito no item 4.3. As propriedades da rede de Petri estão descritas no capítulo 3, item 3.1.2. Utilizamos o software “Petri Netze für Windows” [Frank et Schmidt 1993], para analisar a rede de Petri da Figura 29. A modelagem daquela rede usando este programa está ilustrada na Figura 33, seguinte.

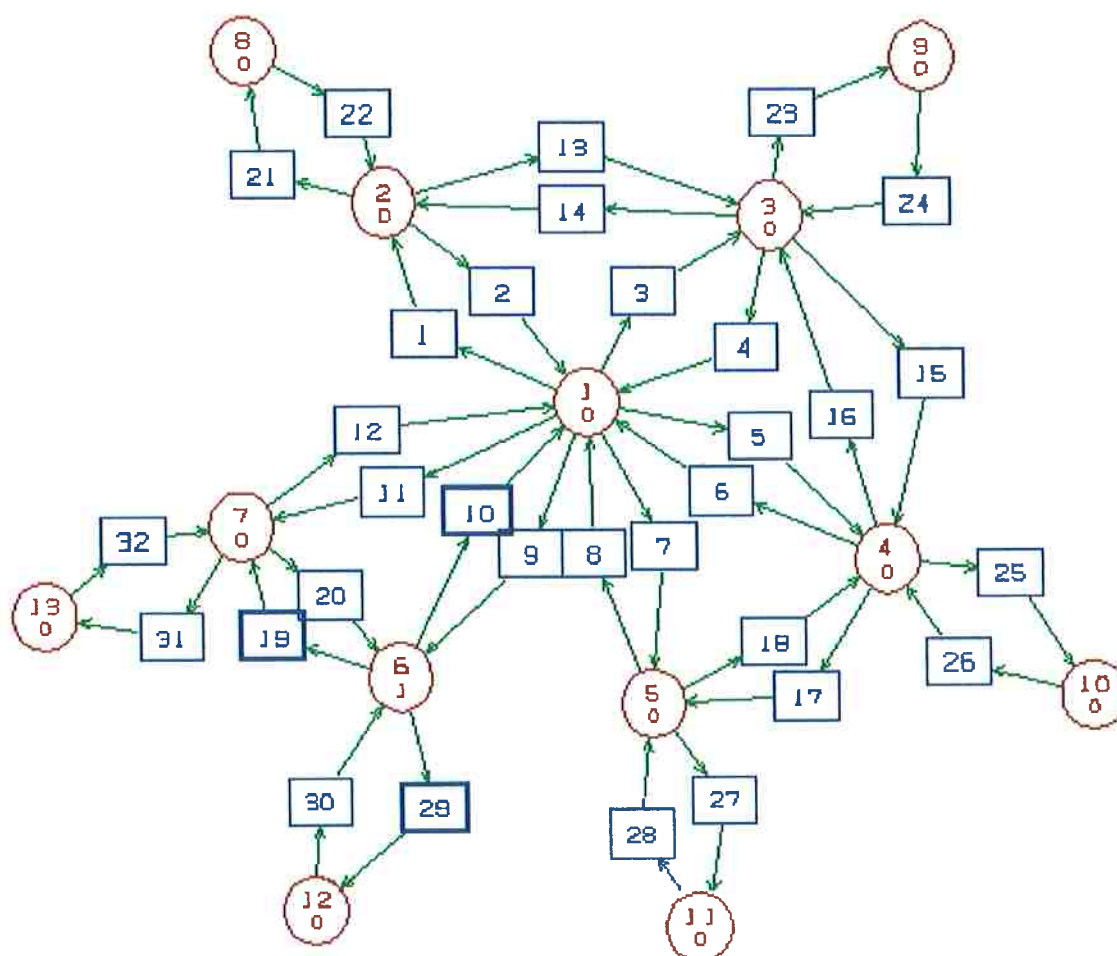


Figura 33 - Implementação da RdP da Figura 30 em Petri Netze für Windows

Na Figura 33, os lugares 1-13 tem a capacidade igual a 1. Apenas um dos lugares é marcado, portanto, o número máximo de marcas na rede é 1. As transições t1-t32 têm apenas um lugar de entrada e um lugar de saída. Uma vez modelado o sistema, o programa pode fazer a simulação da rede e também fazer uma análise das propriedades da rede. A simulação serve para validar a rede, i.e., para verificar se a mesma apresenta o comportamento desejado, como prevê o PASSO6 da nossa metodologia.

Em seguida, listamos a análise das propriedades obtida usando o programa Petri Netze für Windows.

Ergebnisse der graphentheoretischen Analyse von c:\winpetri\blocos.net:
Resultado da análise da teoria de grafos para c:\winpetri\blocos.net:

allgemeine Netzdaten:
dados gerais da rede:

-
- Das Netz besteht aus 13 Stellen.
 - A rede é composta de 13 lugares.
 - Das Netz besteht aus 32 Transitionen.
 - A rede é composta de 32 transições.
 - Der Analyse wurde die starke Schaltregel zugrundegelegt.
 - A análise foi interrompida devido a forte conexão da rede.

Zusammenfassung der Ergebnisse:
Conclusão dos resultados:

-
- Berechnet wurde der Erreichbarkeitsgraph und dessen Kondensation.
 - Foi calculado o grafo de atingibilidade e sua condensação.
 - Das Netz ist beschränkt: Die grösste Stellenmarkierung ist 1, das Netz ist daher sicher.
 - A rede é limitada. A maior número de marcas de lugares é 1, a rede é portanto segura.
 - Das Netz enthält keine toten (nicht schaltfähigen) Transitionen.
 - A rede não contém nenhuma transição morta (que não pode disparar).
 - Es gibt keine partiellen Verklemmungen.
 - Não tem nenhum gargalo parcial.
 - Es gibt keine totalen Verklemmungen (tote Markierungen).
 - Não tem nenhum gargalo total (marcação morta).
 - Bei 7 Netz-Markierungen treten Konflikte auf.
 - 7 marcações da rede apresentam conflito.
 - Das Netz ist kontaktfrei.
 - A rede é livre de contato.
 - Der Erreichbarkeitsgraph ist stark zusammenhängend. Das Netz ist daher reversibel.
 - O grafo de atingibilidade é fortemente conexo. A rede portanto é reversível.
 - Das Netz ist lebendig.
 - A rede é viva.

Konfliktsituationen:
 Situações de conflito:

1: [t10,t19,t29]	2: [t1,t3,t5,t7,t9,t11]	3: [t12,t20,t31]
4: [t2,t13,t21]	5: [t4,t14,t15,t23]	6: [t6,t16,t17,t25]
7: [t8,t18,t27]		

Em suma, como resultado da análise do programa temos que a rede de Petri que representa o “application model” para o Mundo de Blocos é: limitada, segura, livre de contato, viva, reversível e apresenta sete situações de conflito potencial. Sabemos que a Rede de Petri da Figura 33 é uma rede de Petri Elementar (do tipo C/E). De maneira que ela é uma rede limitada onde a marcação dos lugares pertence a $\{0,1\}$. E, portanto, ela é uma rede segura. Uma rede de Petri segura é isenta de contatos.*⁷

Algumas das propriedades da rede são extremamente importantes para se obter um sistema de planejamento baseado em IA de qualidade. Por exemplo, o fato de a rede ser viva é muito importante por assegurar que não existe possibilidade de gargalos (“dead-lock”) na rede, não tendo nenhuma transição morta. Portanto, a implementação do sistema de planejamento baseado nesta Rede de Petri em particular será livre de “dead ends”. O fato da rede ser reversível -- sendo o seu grafo de atingibilidade fortemente conexo -- assegura que a marcação inicial pode ser sempre alcançada.

Para tratar das situações de conflito - que caracterizam situações de escolha e, portanto, podem levar à Anomalia de Sussman - já descrevemos anteriormente (no item 4.3.1, pag. 108), como podemos fazer uma decomposição estrutural do domínio em subredes que são independentes. E que, ao associarmos uma estratégia inteligente de busca da solução à esta (decomposição estrutural) podemos até mesmo eliminar completamente a necessidade de se fazer “backtracking”.

A análise acima foi elaborada, em grande parte, de uma forma manual. Espera-se que a análise da rede de Petri, até então feita pelo modelador humano, possa ser auto-

*⁷ Veja a definição de contato no item 3.2.1, pag. 82.

matizada através de análise de propriedades (estruturais) da rede. O ambiente HIPER, que utilizamos para modelar o meta-K, está em desenvolvimento.

4.7 - O Sistema Implementado para o Mundo de Blocos

Vamos descrever o uso do programa para o Problema do Mundo de Blocos, escrita em Visual PROLOG, com base nas Redes de Petri ilustradas nas Figuras 29, 30, 31 e 33. Inicialmente, deve-se definir o estado inicial e o estado meta usando a função *Inicializa* do Menu. Para mostrar que o Planejador não elabora a solução usando qualquer interpretação existente no domínio do problema mas, sim, usando o modelo conceitual em Rede de Petri que representa o domínio do problema, criamos modos para se acionar o Planejador: modo *Rede de Petri* e modo *Mundo de Blocos*, com mostra a Figura 34, a seguir.



Figura 34 - O Menu de Barras para Mundo de Blocos

As funções do Menu nestes dois modos são equivalentes. Nos dois modos, as funções disponíveis são iguais, como mostra a Figura 35, a seguir. Pode-se inicializar o problema definindo os estados inicial e meta em qualquer dos modos.



Figura 35 - O Menu de Funções do Planejador para Mundo de Blocos

Do mesmo modo, pode-se ativar o Planejador para resolver o problema corrente através da função *Resolve* do Menu em quaisquer dos dois modos. Os passos do plano elaborado são mostrados através da função *Mostra-Solução* do Menu. Pode-se ativar esta função também em qualquer dos dois modos.

Exemplo 1: vamos ver o caso em que EI e M estão na mesma componente conectada. Seja EI = 8 e M = 9. Observe, nas Figuras 28 e 29, na página 108, que ambos pertencem à subrede SA. Suponha que esta inicialização foi feita no modo *Rede-de-Petri* do Menu. Acionando a função *Resolve* do Menu Rede-de-Petri, temos a solução:

condição inicial: 8	condição meta: 9
atividade: 8-2	resultado: 2
atividade: 2-3	resultado: 3
atividade: 3-9	resultado: 9

Observe no algoritmo, na página 111, que o Planejador efetua o PASSO2 em que EI e M estão na mesma componente conectada. Nesse caso, não existe a necessidade de criar nenhuma meta intermediária porque a simples aplicação da abordagem “meios-fins” é suficiente para se encontrar a solução acima. Para obter a interpretação no domínio do problema para esta solução que está na memória do computador, pode-se acionar, agora, o modo *Mundo-de-Blocos* na barra de Menu da Fig. 34 e acionar a função *Mostra-Solução*, como mostra a Fig. 35. Fazendo isto, tem-se uma listagem na tela da solução corrente, agora, interpretada para o domínio do problema - compare com a enumeração dos estados para o Mundo de Blocos, mostrada na Figura 28:

condição inicial: [“blocob”,”mesa”,”blocoa”]	
condição meta: [“blococ”,”blocoa”,”mesa”]	
ação: Move o bloco C sobre a mesa	resultado: [“blocob”,”mesa”,”mesa”]
ação: Move o bloco A sobre o bloco C	resultado: [“blococ”,”mesa”,”mesa”]
ação: Move o bloco B sobre o bloco A	resultado: [“blococ”,”blocoa”,”mesa”]

Exemplo2: Vamos ver agora um caso em que EI e M estão em subredes distintas da RdP. Seja EI = 10 e M = 9. Observe, na Figura 29, que enquanto EI pertence à subrede SB, M pertence à subrede SA. Suponha que esta inicialização foi feita no modo *Rede-de-Petri* do Menu de barras, como mostra a Fig. 34. Acionando a função *Resolve* do Menu Rede-de-Petri, mostrada na Fig. 35, temos a solução:

condição inicial: A (= 10 , em hexadecimal)	condição meta: 9
atividade: A-4	resultado: 4
atividade: 4-1	resultado: 1
atividade: 1-3	resultado: 3
atividade: 3-9	resultado: 9

Observe no algoritmo, agora, que o Planejador efetua o PASSO2 em que EI e M estão localizados em subredes distintas. Nesse caso, deve-se criar a meta intermediária = nó essencial da rede, porque aplicar simplesmente a abordagem “meios-fins” causaria a anomalia de Sussmann, como vimos no capítulo 2. Entretanto, o uso deste meta-conhecimento pelo Planejador é totalmente imperceptível para o Usuário, de modo que ele apenas obtém a solução final já concatenando os planos parciais. Para obter a interpretação no domínio do problema para esta solução final que está na memória do computador, pode-se acionar, agora, o modo *Mundo-de-Blocos* do Menu e acionar a função *Mostra-Solução*. Fazendo isto, tem-se a solução no domínio do problema:

condição inicial:	["mesa", "blocoA", "blocoB"]	
condição meta:	["blocoC", "blocoA", "mesa"]	
ação:	Move o bloco C sobre a mesa	resultado: ["mesa", "blocoA", "mesa"]
ação:	Move o bloco B sobre a mesa	resultado: ["mesa", "mesa", "mesa"]
ação:	Move o bloco A sobre o bloco C	resultado: ["blocoC", "mesa", "mesa"]
ação:	Move o bloco B sobre o bloco A	resultado: ["blocoC", "blocoA", "mesa"]

Vimos assim, através dos exemplos acima, que o Planejador não elabora a solução usando qualquer interpretação existente no domínio do problema mas, sim, usando o modelo conceitual em Rede de Petri que representa o domínio do problema. Em outras palavras, o sistema não usa regras heurísticas. Ao contrário, faz uma implementação do mc, obtendo o resultado em RdP. Daí, a interface com o Usuário é que faz a interpretação desta solução para o domínio do Mundo de Blocos. Portanto, tem-se um sistema flexível onde o Planejador sempre encontra uma solução, quaisquer que sejam EI e M.

4.7.1 - Resultados Adicionais para o Mundo de Blocos

Resultados adicionais foram obtidos para o sistema acima descrito, usando a noção de que o *caminho mínimo* corresponde ao “menor número de disparos” entre transições. Este conhecimento adicional permite até que se descarte o uso do meta-conhecimento para criar a meta intermediária no caso de o EI e M pertencerem a subredes distintas.

caminho mínimo:*⁸ seqüência com o menor número de disparo de transições possível para ocorrer o disparo de duas determinadas transições.

Incluindo-se este conhecimento na forma de termos PROLOG e usando a regra heurística da menor distancia síncrona para escolha da transição dentre todas as transições habilitadas, obtivemos uma solução equivalente à solução descrita acima. Nesse caso, não foram criadas submetas intermediárias no caso de *EI* e *M* pertencerem a subredes distintas, ou seja, este meta-conhecimento tornou-se redundante com o conhecimento adicional que foi incluído. O programa que implementa esta melhoria foi elaborado na linguagem Visual PROLOG.

Em nossa proposta, a rede de Petri que representa o “application model” não precisa ser necessariamente construída fazendo-se uma enumeração completa dos estados do domínio do problema. A Hipótese do Mundo Fechado só requer a existência de um modelo lógico do domínio. No Apêndice II, apresentamos uma fórmula analítica dos estados do sistema para o problema do Mundo de Blocos generalizado para n blocos. Descrevemos a implementação feita e mostramos a solução provida pelo sistema para cinco e para dez blocos.

4.8 - Uma Proposta de Arquitetura para o Sistema

Os sistemas clássicos de planejamento baseados em IA, que descrevemos no capítulo 2, não levam em conta o processo de execução do plano. Eles elaboram o plano completo sem levar em conta a dinâmica do ambiente em que o plano é executado. A simplificação geralmente feita é a de que o domínio é fechado, de que todas as metas são consistentes e de que todos os eventos presentes nos estados subseqüentes ao estado de

*⁸ Uma simplificação do conceito de distância síncrona. Distância síncrona [Reisig 1985] é uma propriedade definida para sistemas de RdP do tipo C/E e que serve para medir o grau de dependência entre as ocorrências de suas transições, i.e., de que maneira a ocorrência de uma determinada transição depende da ocorrência de outras transições. A distância síncrona é uma medida de sincronização de duas transições. Observa-se a freqüência com que duas transições $t1$ e $t2$, respectivamente, ocorrem em cada processo p do sistema. A diferença absoluta destas freqüências de ocorrências é chamada de *variância* de $t1$ e $t2$. O supremo desta variância é definida como sendo a distância síncrona entre $t1$ e $t2$.

referência (“baseline”), que foram antecipados pelo planejador, são verdadeiros. Além disso, a execução de todas as ações programadas no plano é efetuada sem falhas.

Entretanto, a execução de um plano pode: 1) lograr sucesso: o executor do plano consegue executar todas as etapas do plano sem falhas, 2) não lograr sucesso: o executor do plano consegue executar algumas etapas do plano sem falhas, mas falha em etapas subsequentes. Portanto, para obtermos um sistema que contempla planejamento e a execução devem ter um sistema de monitoração da execução do plano.

A Figura 36, a seguir, mostra os módulos necessários para compor o sistema de Planejamento Semi-Reativo da nossa proposta.

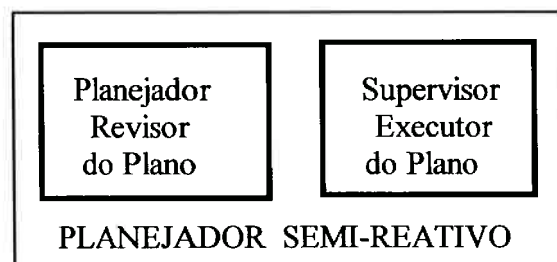


Figura 36 - Os Módulos do Sistema Planejador Semi-Reativo (PSR)

Os módulos de um Sistema Planejador Semi-Reativo (PSR) são dois: o módulo Planejador e Revisor do Plano e o módulo Supervisor e Executor do Plano. Normalmente, o módulo Planejador constrói o plano completo antes que a sua execução seja iniciada. Durante a execução do plano, o módulo Supervisor e Executor do Plano efetua as ações programadas no plano invocando diversas rotinas de baixo nível. O módulo Supervisor monitora se a execução destas rotinas atinge os objetivos desejados. O módulo Supervisor é também responsável pela atualização contínua do estado de referência (ou “baseline”), atualizando eventos como cancelamento de tarefas, chegada de novas tarefas, disponibilidade dos recursos, etc. Quando uma ação programada no plano corrente não logrou sucesso ou, então, não pode nem ser iniciada, o módulo Supervisor e Executor invoca o módulo Planejador para que este modifique o plano corrente.

O módulo Supervisor consiste de um sistema de comunicação (sensoreamento , atuação) com o meio-ambiente onde atua o sistema. Isto se faz através de:

- 1) um gerador de comandos para os atuadores que geram comandos na forma de rotinas primitivas para realizar as ações programadas pelo Planejador
- 2) atuadores que efetuam estas rotinas primitivas do sistema no meio-ambiente,
- 3) sensores que adquirem sinais a partir do meio-ambiente,
- 4) um sistema monitor que transmite os sinais dos sensores para a base de dados, atualizando constantemente as crenças atuais do sistema.

O módulo Supervisor e Executor do Plano está localizado na camada de Controle do SIM. Por este motivo, também é perfeitamente compatível com a modelagem por Rede de Petri. O Módulo Supervisor, entretanto, está fora do escopo deste trabalho.

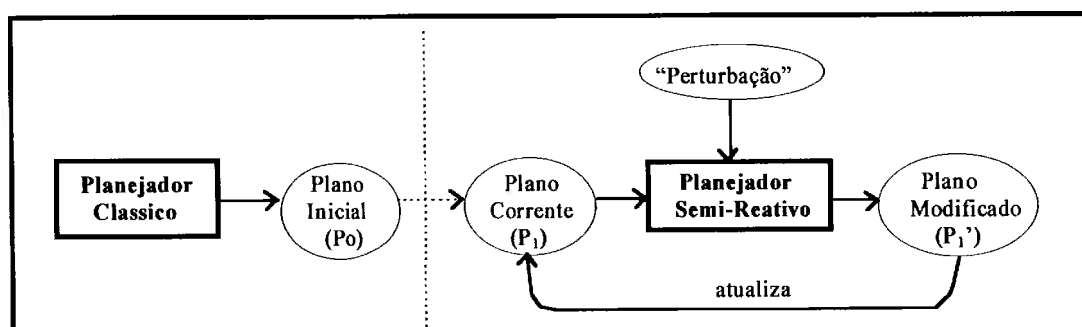


Figura 37 - Os Componentes do Módulo Planejador / Revisor

4.8.1 - O Módulo Planejador e Revisor do Plano

O principal objetivo em se usar sistemas de “planning” baseados em IA é o de se obter sistemas mais flexíveis, capazes de adaptar rapidamente às mudanças ocorridas no sistema alvo. Espera-se que, nestes sistemas dinâmicos, a ocorrência de mudanças seja a regra e não a exceção. Portanto, o sistema de “planning” deve ser concebido, desde o princípio, para que seja eficiente e eficaz no tratamento destas mudanças que, certamente, irão ocorrer.

A Figura 37, acima, ilustra os componentes do módulo Planejador e Revisor do Plano, de [Silva et Shimad 1995]. O sistema de “planning” possui duas componentes:

- (1) o Planejador Clássico (PC) que gera o Plano Inicial (Po) e,
- (2) o Planejador Semi-Reativo (PSR) que tenta manter continuamente factível um Plano Modificado (P1’) a partir do Plano Corrente (P1), levando-se em conta “perturbações” ocorridas no sistema dinâmico.

Este sistema PSR não elabora planos de contingência. Ao contrário, elabora inicialmente um plano factível que é tornado o plano corrente. Então, o sistema tenta manter este plano corrente factível, atuando continuamente de uma forma semi-reativa.

O que é comum nestas duas componentes do sistema de “planning” está no fato de ambas utilizarem o mesmo modelo conceitual (mc). PC aplica um método clássico de “planning” para elaborar o Plano Inicial (Po). PSR deve validar / modificar continuamente o Plano Corrente (P1), levando em conta “perturbações”^{*9} do sistema obtidas através do módulo de Supervisão do sistema alvo, elaborando (ou não) um Plano Modificado (P1’).

A nossa proposta trata do Planejador Semi-Reativo porque dirigimos os resultados para os Sistemas Flexíveis de Fabricação (SFF), em que consideramos importante ter-se sempre à mão um plano corrente completo. Em contraposição, os sistemas puramente reativos operam com planos parciais. Estes planos parciais são continuamente modificados em decorrência da monitoração do meio ambiente dinâmico em que estão implementados e são especializados apenas no instante de sua execução.

^{*9} Por “perturbação” ao plano queremos significar: modificações, não muito drásticas, nos atributos dos objetos que estão presentes no modelo de mundo. Note que o modelo conceitual feito em rede de Petri efetua um fechamento sobre o domínio do problema. Uma “perturbação” que envolve apenas a eliminação de objetos do modelo conceitual pode ser tratado de uma forma trivial. Este sistema também é capaz de tratar “perturbações” que envolvem a inclusão de objetos de classes já conhecidas no modelo conceitual. Note que estas podem causar uma modificação da estrutura da rede de Petri que representa o domain-K. Por este motivo, este último tipo de perturbação deve ter uma extensão limitada. Em suma, nenhuma perturbação não pode ser tão extensa a ponto de tornar este modelo conceitual não aderente ao problema.

Vamos descrever como o PSR trata das “perturbações” ao plano corrente. No caso de cancelamento de uma tarefa anteriormente prevista e já programada no plano corrente, o PSR não refaz todo o processo de elaboração do plano. Apenas elimina a tarefa do plano corrente, mantendo as demais tarefas anteriormente programadas no plano. A reconstrução do todo ou de parte do plano corrente deve ser feita apenas no caso de inclusão de novas tarefas ao plano corrente ou, então, no caso de algum recurso previsto não estar mais disponível, por exemplo, por motivo de quebra.

Estas “perturbações” ao plano corrente devem ser refletidas no modelo conceitual (mc) do problema. No caso de cancelamento de tarefas ou de indisponibilidade de um recurso, apenas elimina-se a marcação correspondentes da rede de Petri apropriada. Entretanto, no caso de acréscimo de tarefas não previstas anteriormente, deve-se modificar a topologia da RdP para levar em conta esta “perturbação”.

4.8.2 - Porque a Reatividade é Importante

Este conceito de reatividade surgiu, principalmente, em decorrência da necessidade de desenvolver aplicações que tratam da navegação de robôs móveis. Estes sistemas operam essencialmente em tempo-real, monitorando continuamente o ambiente em que atuam. Nesse contexto, é fundamental obter uma solução num tempo de resposta baixo para o problema de reagir às “perturbações” do ambiente. Entretanto, os resultados obtidos no desenvolvimento de tais sistemas podem e devem ser utilizados no planejamento de sistemas dinâmicos em outras áreas de aplicação, como são os sistemas flexíveis de manufatura (SFF).

Um sistema reativo, geralmente, não constrói um plano completo. A invés disso, opera sempre com planos parciais cujas tarefas são especializados ou detalhados no momento de sua execução.

[Georgeff 1987] descreve outras questões impostas pela necessidade de obter um tempo de resposta baixo, e que não pode ser obtido pelos sistemas tradicionais porque:

- 1) tipicamente, técnicas de planeamento requerem um longo tempo de processamento. Isto é inaceitável em aplicações em sistemas dinâmicos, porque nestes a ocorrência de eventos imprevistos é a regra e não a exceção.
- 2) sistemas tradicionais não possuem nenhum mecanismo para modificar suas metas e intenções durante a execução do plano de modo a permitir uma rápida reação em face a novas situações. Estes sistemas deveriam ser capazes de raciocinar acerca das suas intenções correntes, modificando-as e trocando-as a luz de suas crenças e metas em mudança.
- 3) sistemas tradicionais confiam excessivamente na elaboração do plano a partir dos "primeiros princípios", i.e., a partir de ações primitivas. É necessária uma grande quantidade de "conhecimento procedimental" pré-compilado acerca de como operar (reativamente) no mundo real.
- 4) sistemas tradicionais estão muito comprometidos com estratégias de planeamento. Usam sempre uma única técnica de planeamento e não possuem métodos opcionais que possam ser escolhidos em diferentes situações. Em qualquer situação, mesmo nas emergenciais, gastam igual tempo de processamento para planejar o atingimento de determinada meta antes de efetuar qualquer tarefa externa. Não tem capacidade para decidir quando o planeamento deve ser parado, nem fazem balanço entre a decisão de continuar planeamento e o tempo de execução restante. Até mesmo sistemas que intercalam planeamento e execução ainda estão muito comprometidos com o atingimento das metas que lhe foram atribuídas inicialmente. Não tem nenhum mecanismo para trocar o foco, adotar metas diferentes ou reagir a mudanças rápidas e inesperadas que ocorrem no seu meio-ambiente.

4.9 - Diferença com outras abordagens

A maioria dos sistemas clássicos de planeamento baseados em IA baseiam-se principalmente no uso de um operador do tipo STRIPS. Por este motivo, têm de lançar mão de diversas estratégias para contornar os problemas decorrentes desta escolha. Note que a maioria das contribuições apresentadas nos sistemas pioneiros ou clássicos -- proteção de metas já atingidas, críticas para detectar conflito, regressão de metas, hipótese do

mundo fechado, ... -- estão relacionadas, de alguma forma, com formas de se corrigir este problema. Até mesmo, nos sistemas que integram planeamento e execução, são necessários os diferentes mecanismos propostos -- tabela triângulo, árvore de submetas e grafo de decisão, ... -- tratam da questão de se contextualizar os efeitos da aplicação de uma determinada ação definida no plano.

Ao invés disso, nossa proposta está fundamentada na elaboração de um modelo conceitual usando a teoria da Rede de Petri. A nossa proposta de estruturação faz uma separação clara dos tipos de conhecimento existentes num problema de “planning”: o conhecimento do domínio da aplicação e o meta-conhecimento para dirigir o processo de busca de soluções. A rede de Petri é usada para representar: (1) o problema de “planning” em si (o “planning self-model”), e (2) o domínio do problema (o “application model”). Estas redes de Petri constituem um “mapa rodoviário” ao alcance da mão do Planeador e descrevem todo o processo de “planning”. O “planning self-model” faz a representação *do meta-planeamento*, ou seja, das diversas estratégias possíveis para se construir o plano. O “application model” promove um fechamento sobre o domínio do problema. Ele não corresponde a nenhum plano em particular mas, sim, à combinação de todos os planos factíveis ou não que constituem soluções para o problema.

Além disso, a nossa proposta é compatível com a associação do planeamento com a execução, como mostra a Figura 36. Um plano corrente pode ser mantido factível, reagindo semi-reativamente às “perturbações” que ocorrem no ambiente onde o sistema atua.

No exemplo de Mundo de Blocos, tratamos de um problema exemplo clássico e conhecido onde o modelo de mundo pode ser fechado através da enumeração completa dos estados do sistema e, até mesmo, a análise da rede de Petri do domain-K não apresenta maiores dificuldades porque esta é muito simples. Pode-se-ia argumentar que os resultados obtidos poderiam ser obtidos apenas para estes problemas “toy”. Por este motivo, para mostrar que a nossa proposta de metodologia é válida e aplicável para problemas de grande porte do mundo real, descrevemos, no capítulo 5, uma aplicação que desenvolvemos para um Sistema Roteador de Helicópteros.

CAPÍTULO 5

5 O ROTEADOR DE HELICÓPTEROS

O Roteador de Helicópteros consiste de um sistema para elaborar a programação de vôos de helicópteros entre uma base localizada em terra (“on-shore”) e plataformas de petróleo localizadas no mar (“off-shore”). Este tipo de transporte consiste em levar passageiros a partir do Aeroporto no litoral, denominado de base, até os helipontos localizadas nas plataformas no mar e recolher passageiros a partir destes e levá-los até a base. A Figura 38, seguinte, ilustra o Sistema Roteador de Helicópteros.

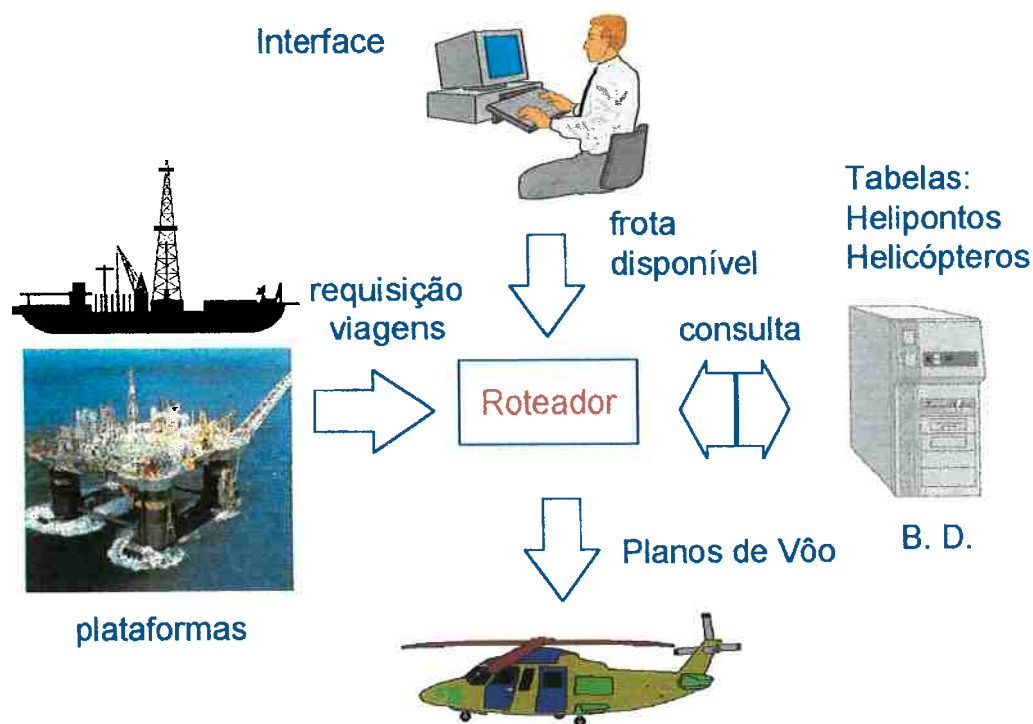


Figura 38 - O Sistema Roteador de Helicópteros

Este sistema é utilizado na rotina do dia-a-dia pelo programador de vôos. Para cada horário de partida do vôo a partir da base, existe uma demanda de passageiros de ida para as plataformas e de volta para a base. O objetivo é atender a demanda no menor tempo de atendimento e com o menor custo, considerando a disponibilidade de helicópteros. Para se ter uma idéia da sua complexidade, este problema envolve 50 helipontos, 16 helicópteros que podem ser de sete tipos diferentes, cinco horários de partida dos

vãos e até 20 mil passageiros transportados no mês. Seria, portanto, praticamente impossível fazer uma enumeração completa dos estados do sistema como no exemplo do Mundo de Blocos.

Os dados de entrada para o Roteador de Helicópteros são: dia e horário de partida do voo a partir da base, dados dos helipontos nas plataformas, dados da frota de helicópteros e as requisições de viagem. Os dados dos helipontos são: nome do heliponto, abcissa do heliponto, ordenada do heliponto. Os dados da frota são: tipo de helicóptero, quantidade de helicópteros disponíveis de cada tipo, capacidade dos helicópteros - número de assentos para passageiros, velocidade do helicóptero (km/h) e custo do helicóptero por hora de voo (U\$/h). As requisições de viagem consistem de demanda de passageiros -- de ida e de volta -- para cada heliponto.

Os dados de saída do sistema são: dados das rotas geradas, o custo total (U\$) da rotas geradas, a distância total percorrida (km), o número de passageiros transportados e o custo unitário por passageiro (U\$/passag.). Os dados de cada rota gerada são: o tipo de helicóptero atribuído, o horário de partida do voo, a seqüência de helipontos na rota, a grade de horários para cada heliponto na rota, a distância entre helipontos, o tempo de viagem em cada trecho, o custo de cada trecho, o número de passageiros de ida e de volta para cada heliponto e a ocupação em cada trecho da rota.

Mostramos anteriormente, no capítulo 4, como uma abordagem que associa uma análise da estrutura em Rede de Petri e IA pode melhorar o desempenho na solução do problema geral de "planning". Nesta abordagem, o primeiro passo no desenvolvimento de um sistema de "planning" baseado em IA consiste na síntese de um modelo que representa o domínio do problema baseado em Rede de Petri. A partir da análise da estrutura da RdP deste modelo do domínio, podemos obter critérios de escolha da próxima ação a ser efetuada no plano que está sendo elaborado e, deste modo, dirigir o processo de busca da solução. Para ilustrar isto, no capítulo 4, usamos o exemplo do Mundo de Blocos. Desta forma, tem-se um sistema planejador que elabora soluções com mais qualidade, evitando a Anomalia de Sussmann.

Aplicamos a metodologia descrita no capítulo 4 para desenvolver o nosso Roteador de Helicópteros. Desta maneira, conseguimos obter um sistema mais flexível que apresenta um planejador que elabora soluções com mais qualidade. Podemos comparar o serviço de transportes por helicóptero com a idéia de ‘Célula Virtual’ num Sistema de Manufatura Flexível com esquema de gestão “just-in-time”, i.e., sem estoque de produtos finais ou intermediários. A ordem de serviço (que pode ser comparada com demanda de viagem) deve ser atendida imediatamente através de operações efetuadas pela Célula Flexível (ou pela frota de helicópteros). A diferença está em que na Célula de Manufatura, as peças é que são roteadas, enquanto que no problema dos Helicópteros, são os recursos que são roteados.

O problema de fazer o “scheduling” de uma Célula de Manufatura com única máquina, com custos associados com a troca de ferramentas entre tarefas, em que estes custos dependem da seqüência de tarefas, é equivalente ao Problema do Caixeiro Viajante (PCV) [Kusiak 1990]. A formulação do problema de Scheduling Job Shop para uma Célula de Manufatura, como um Problema de Caixeiro Viajante (PCV) está descrito no item 5.6.1.2, mais adiante.

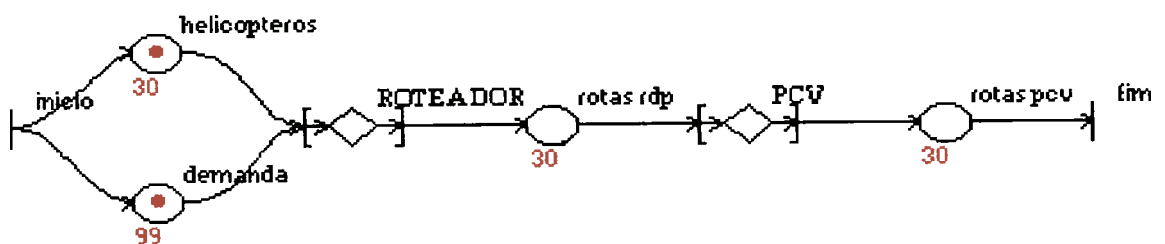


Figura 39 - Os Componentes do Roteador de Helicópteros

5.1 As componentes do Roteador de Helicópteros

O modelo conceitual (*mc*) do problema de “planning”, i.e., o “planning self model” foi desenvolvido usando Rede de Petri estendida Ghenesys. A Figura 39, acima, ilustra as componentes do Sistema Roteador de Helicópteros usando elementos da rede estendida Ghenesys. O sistema consiste de um Roteador baseado em Rede de Petri e um Otimizador das rotas geradas. O Roteador baseado em Rede de Petri constrói simulta-

neamente todas as rotas necessárias para atender todas requisições de viagens existentes para determinado horário de voo, de uma maneira incremental. O otimizador de rotas é baseado no algoritmo 3-opt para o Problema do Caixeiro Viajante (PCV).

Para um determinado horário de partida, havendo demandas por viagens e recursos suficientes para atender estas, o sistema usa o Roteador para elaborar as rotas iniciais (ou rotas RdP). Em seguida, estas rotas iniciais são melhoradas aplicando-se o Otimizador. O Otimizador analisa uma série de permutações para cada rota e define as rotas finais (ou rotas PCV), observando as limitações de capacidade tanto de uma forma agregada assim como em cada trecho da rota escolhida. Entretanto, como no exemplo que mostramos no item 5.4.1, este Otimizador, muitas vezes, é redundante porque o Roteador baseado em Rede de Petri já obtém uma solução inicial que já é a melhor solução para uma determinada rota.

No problema exemplo do Mundo de Blocos, descrito anteriormente no capítulo 4, fizemos a elaboração do “application model” fazendo uma enumeração completa dos estados do sistema para o caso de 3 blocos, e usando raciocínio por indução para o caso de n blocos. No sistema Roteador de Helicópteros, a representação do conhecimento do domínio (o domain-K) não é elaborado nem por enumeração dos estados e nem por indução. Ao invés, utilizamos a idéia de grupos (“clusters”) de plataformas, que são formados dinamicamente em função da existência ou não de demandas de viagem de / para aqueles helipontos. Estes grupos de plataformas devem ser preferencialmente roteados num mesmo voo. Mas, estes grupos podem ser quebrados caso haja limitação de capacidade nos voos, ou seja, estes grupos são flexíveis. Os “clusters” foram inspirados na idéia de células virtuais usadas nos SFFs.

5.2 - O Tratamento de subredes pelo sistema

Através da observação das coordenadas de localização dos helipontos no mar, elaboramos um agrupamento inicial destes em duas subredes: subrede A e subrede B. Mais especificamente, o sistema considera como sendo parte da subrede A toda plataforma

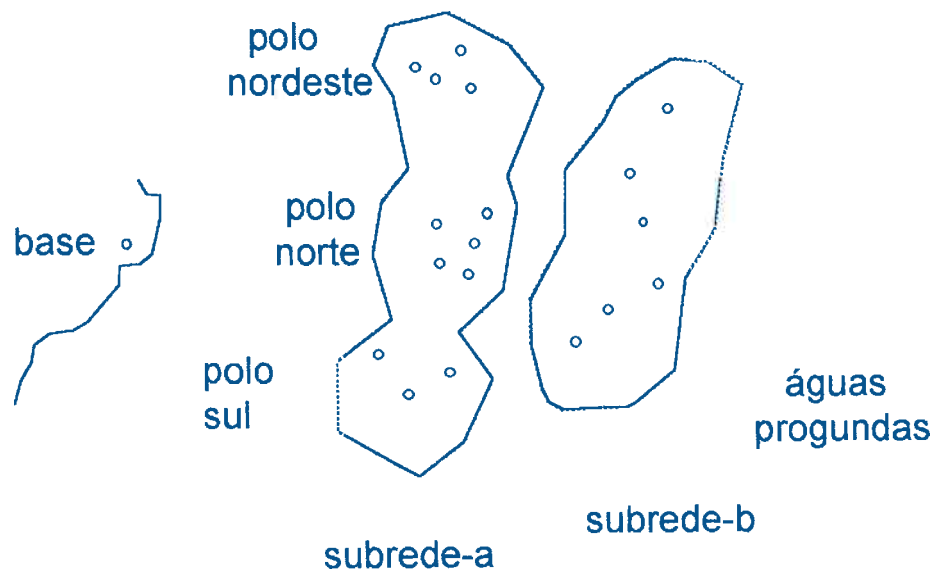


Figura 40 - A Localização dos helipontos no mar (sem escala)

com as coordenadas*¹: abcissa < 7510000 e ordenada < 343000. Fazem parte da subrede B, toda plataforma com as coordenadas: abcissa >= 7510000 e ordenada >= 343000. A Figura 40, acima, ilustra a localização geográfica dos helipontos nestas subredes. A subrede A é constituída pelas plataformas localizadas no polo nordeste (PNE), polo norte (PN) e polo sul (PS). A subrede B é constituída pelas plataformas localizadas em águas profundas (AP).

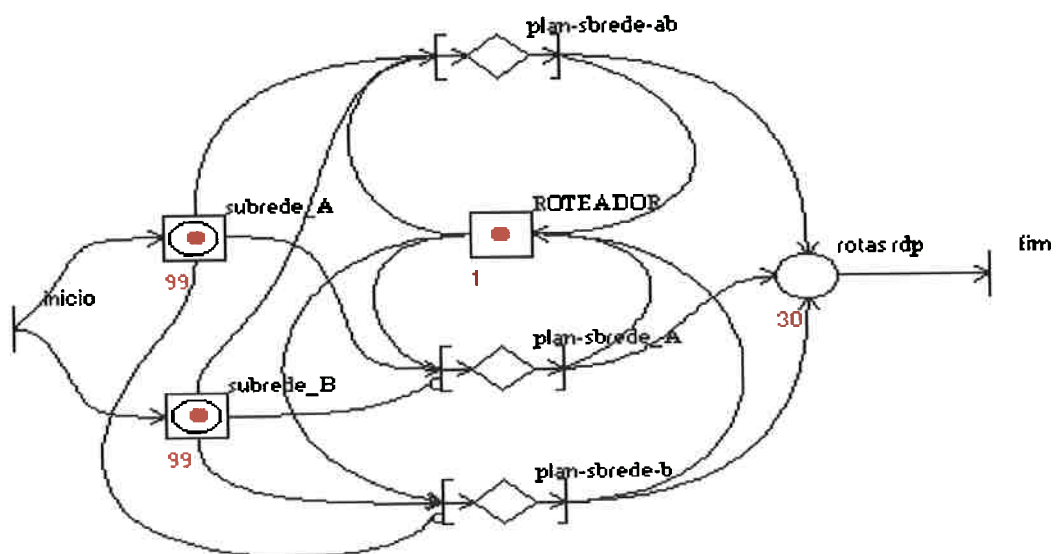


Figura 41 - O tratamento de subredes pelo sistema

*¹ Veja exemplo de coordenadas dos helipontos na Tabela 1, na pag. 146.

A Figura 41, acima, como o sistema trata as requisições de viagens de ida e de volta para plataformas pertencentes a estas subredes.

Para economizar tempo de processamento, o sistema Roteador de Helicópteros constrói a rede de Petri que representa o “application model”, dinamicamente, cada vez que o sistema é usado, fazendo “assertz” de termos que representam os elementos da RdP, ou seja, os lugares, as transições e arcos de ligação. O “application model” é construído “virtualmente” na memória RAM do computador, apenas para os helipontos para os quais existe uma demanda de viagem de ida e/ou de volta, a partir da base. Através da Figura 41, podemos ver que, devido ao uso de portas (“gates”) inibidoras, o sistema irá construir a Rede de Petri para a subrede A se houverem apenas demandas para a subrede A. Do mesmo modo, o sistema irá construir a Rede de Petri para a subrede B se houverem apenas demandas para a subrede B. Caso existam requisições de viagens para ambas as subredes, então, o sistema elabora a Rede de Petri considerando a totalidade dos helipontos.

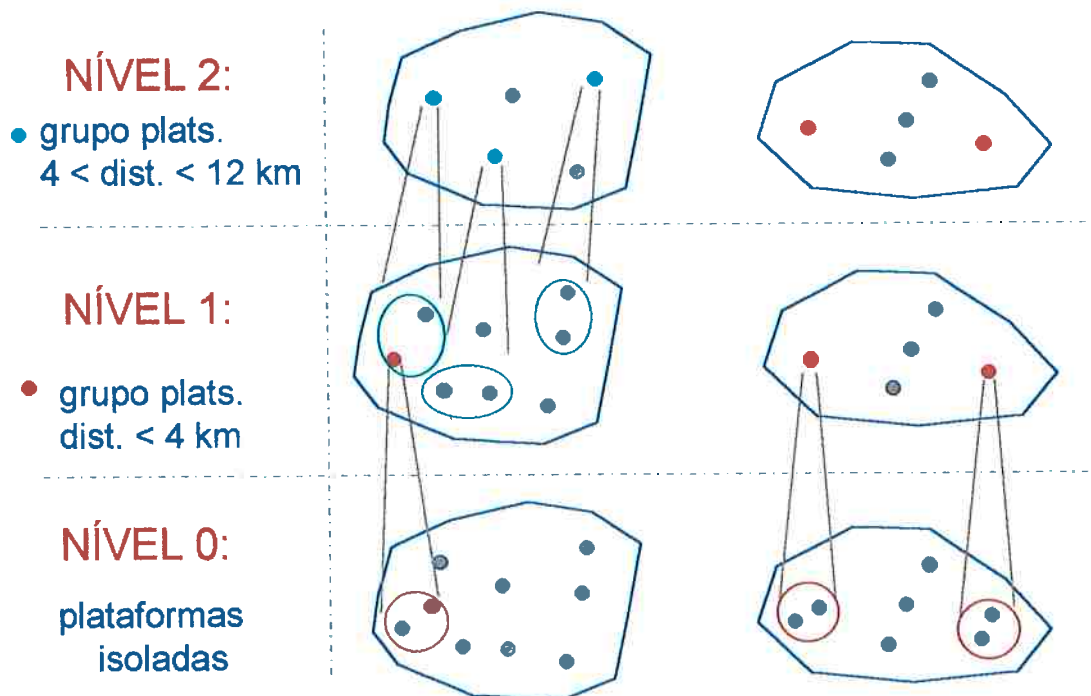


Figura 42 A Estruturação do Conhecimento do Domínio (domain-K)

5.3 - A Definição dos Níveis de Abstração para o Problema Corrente

Uma vez definida a subrede a ser tratada, o sistema passa ao processo de síntese da Rede de Petri estendida Ghenesys, que representa o domínio do aplicação (ou seja, do “application model”). Porque esta RdP pode tornar-se muito grande se incluirmos todos os helipontos existentes, o sistema faz a síntese da RdP dinamicamente na hora de fazer o planejamento das rotas, levando em conta apenas as requisições de viagens existentes. Existe um módulo de Agrupamento das plataformas acoplada à rotina de Síntese da RdP (Veja a Figura 44, mais adiante).

A Rede de Petri estendida Ghenesys suporta a representação de diferentes níveis de abstração do problema. Para tirar vantagem desta capacidade na representação do problema do Roteador elaboramos o modelo conceitual do problema usando diversos níveis de abstração. Para cada nível de abstração, definem-se modelos do domínio correspondentes. Nos modelos de alto nível, os lugares são constituídos por grupos de helipontos com coordenadas geográficas de localização próximas. O emprego destes níveis de abstração foi muito útil porque podemos rotear passageiros para ida ou de volta de um grupos de helipontos para um mesmo vôo -- desde que o helicóptero tenha capacidade suficiente para tanto.

Os níveis de abstração usados foram três: Nível 0, Nível 1 e Nível 2, como mostra a Figura 42, na página anterior. Os níveis de abstração são formados para plataformas com demandas de viagem existentes. No Nível 0 ou nível “ground” os boxes da RdP são constituídos pelas plataformas tratadas isoladamente (●). No Nível 1, os boxes representam plataformas isoladas (●) e grupos de plataformas distantes entre si até 4 km (●). No Nível 2, os boxes representam plataformas isoladas (●), grupos de plataformas distantes entre si até 4 km (●) e grupos entre 4 km e 12 km (●). Estes valores são empíricos e foram fixados com base em testes realizados. Evidentemente, estes níveis podem ou não ser formados para um dado horário de vôo porque dependem das demandas existentes naquele horário. Entretanto, uma vez definidos os níveis de abstração, faz-se a síntese da Rede de Ghenesys que representa o conhecimento do domínio (o domain-K). Esta rede Ghenesys para o domain-K é tal que os grupos de helipontos - (●), (●) - nos ní-

veis mais altos de abstração são representados por um elemento-estático-composto (O) (ou subrede-estática), e a demanda do grupo é o somatório das demandas para os seus helipontos componentes. No nível de abstração mais baixo (ou nível “ground”), só existem plataformas isoladas que são representadas por boxes simples.

O Sistema Roteador de Helicópteros, apresenta não só uma hierarquia de planos, mas também uma hierarquia de modelos do domínio da aplicação. O sistema elabora um plano para cada nível de abstração operando sobre o modelo lógico do domínio que corresponde àquele nível de abstração, i.e., à granularidade dos “clusters” de helipontos daquele nível. A solução para o nível de abstração mais alto torna-se uma meta para o sistema de baixo nível. A condição de término do processo de “planning” é atingida quando o plano para o Nível 0 (ou nível “ground”) é obtido.

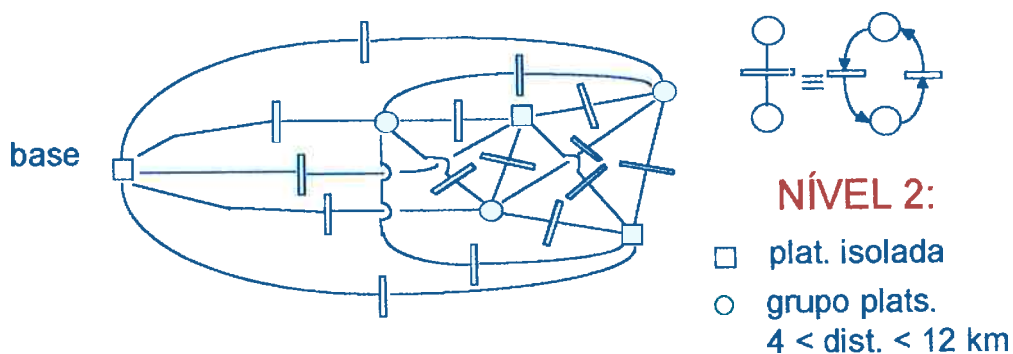


Figura 43 Rede Ghenesys para um “cluster” de plataformas situado no Nível 2

A Figura 43, acima, ilustra a Rede de Petri estendida Ghenesys que representa um “cluster” de plataformas situado no Nível 2, da Figura 42. O domínio deste problema pode ser representado por uma Rede Ghenesys, onde os helipontos (ou grupo de helipontos) constituem os boxes (ou boxes compostos) e as transições são formadas pelo produto cartesiano destes boxes. Em adição à estrutura da rede de Petri, regras adicionais que levam em conta a capacidade dos helicópteros são utilizadas para escolha da próxima ação a ser efetuada num determinado plano em elaboração.

O sistema faz a síntese da Rede Ghenesys que representa todo o domínio do problema dinamicamente, cada vez que for fazer o planeamento das rotas, levando em conta

apenas as requisições de viagens existentes. Observe a semelhança da rede de Petri mostrada na Figura 43 com a rede usada para representar o domain-K para o Mundo de Blocos, mostrada na Figura 29.

A Figura 44, seguinte, mostra a representação usando Rede de Petri estendida Ghenesys para representar o processo de agrupamento de helipontos e de síntese do modelo do domínio. A síntese do modelo do domínio envolve ainda o cálculo das distâncias que correspondem à cada transição. Estas distâncias permitem ao Roteador fazer o cálculo do tempo de vôo estimado e, em consequência, o custo estimado da inserção do heliponto correspondente na rota.

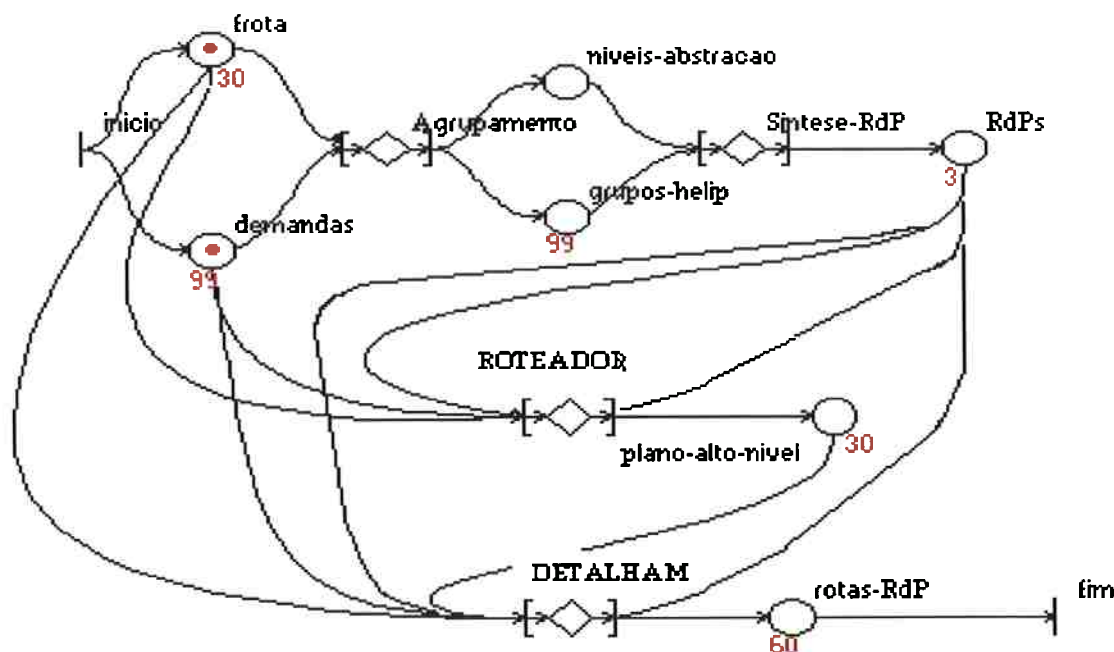


Figura 44 - O Agrupamento de helipontos e a síntese do modelo do domínio

5.4 O Roteador baseado em Rede de Petri

Dependendo da demanda existente para determinado horário, podemos ter até três níveis de abstração para o problema corrente. Entretanto, o processamento do Roteador é sempre o mesmo, não importa qual seja a combinação de níveis de abstração definida para o problema. Como mostra a Figura 44, acima, o sistema gera o plano para o nível de abstração mais alto existente e, então, faz sucessivamente o detalhamento para o nível mais baixo até chegar ao Nível 0 (ou nível “ground”). O Roteador de Helicópteros ope-

ra de uma forma “top-down” para elaborar o plano inicial P_0 e opera de uma forma “bottom-up” para fazer a modificação do plano corrente P_1 para tratar as “perturbações”.

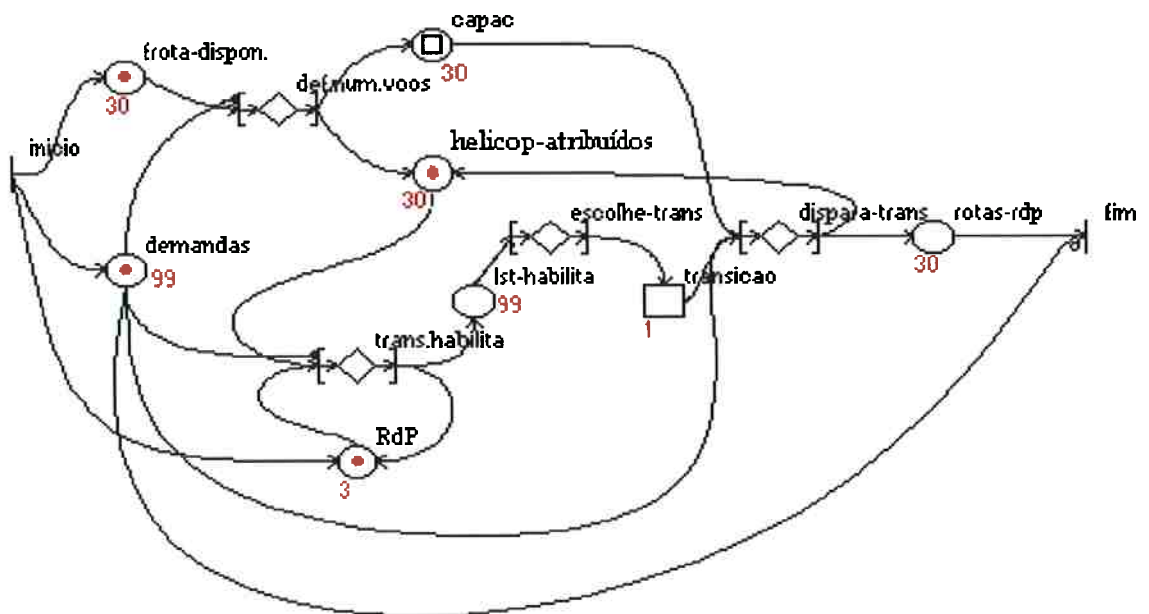


Figura 45 - O Roteador baseado em Rede de Petri

A Figura 45, acima, ilustra uma representação do Roteador, usando da Rede de Petri estendida Ghenesys. O plano é elaborado incrementalmente pelo Construtor de Planos ou Roteador. O Construtor de Planos é um Ambiente de Rede de Petri*² que opera sobre a rede que representa o conhecimento do domínio - o “application model”. Conseqüentemente, todas as rotas são elaboradas em paralelo.

O Roteador sempre é aplicado para o nível de abstração definido como sendo o mais alto para o problema corrente. Para este nível, já foram definidos: o horário de partida dos vôos, os grupos de helipontos, a frota de helicópteros disponíveis e já foi sintetizada a Rede Ghenesys que representa o domínio do problema (o “application model”). A condição inicial do sistema (ou “baseline”) é o de se ter todos os helicópteros prontos para iniciar a viagem na base e nenhuma requisição de viagem ainda não foi

*² Note que o Construtor de Planos baseado em Rede de Petri utiliza apenas a estrutura da Rede de Petri do domain-K. Ele não explora a dinâmica da rede e nem faz a evolução de marcas na rede. Veja no cap. 3, item 3.6, as dificuldades para se efetuar o mapeamento da RdP em lógica.

atendida. A condição final do sistema (ou “goal”) é o de se ter todos os helicópteros retornados na base, depois ter atendido a todas as requisições de viagem.

Inicialmente, o Roteador irá definir o número de vôos para resolver o problema corrente. Esta definição pode levar em conta uma reserva de capacidade, normalmente, de um assento livre por helicóptero. Em seguida, o Roteador atribui (“allocates”) os helicópteros para cada vôo. Para escolha do tipo de helicóptero a ser atribuído é empregado um critério muito simples; os helicópteros são ordenados segundo uma função de custo unitário (US\$) por passageiro, por km percorrido. Ao se atribuir o tipo de helicóptero para uma rota, define-se também a capacidade - número de assentos disponíveis - para esta rota.

Então, o Roteador faz a atribuição do primeiro heliponto na rota. Como o objetivo é o de atender a demanda existente no menor tempo de atendimento, usamos a heurística do *espalhamento máximo dos helicópteros* para fazer a escolha do primeiro heliponto na rota de cada helicóptero que foi alocado. Os resultados computacionais descritos no item 5.6, mais adiante, mostram que esta regra heurística é valiosa não só para se gerar rotas com menor tempo de atendimento mas, também, para tornar o sistema mais flexível para fazer o tratamento das “perturbações” ao plano corrente.

No processo típico do Planejador -- para o *i*-ésimo heliponto na rota -- o ambiente de Rede de Petri considera todas as rotas em paralelo. O Roteador é um Ambiente de Rede de Petri que opera sobre a RdP que representa o domain-K. O Roteador promove uma “competição” entre os helicópteros -- inspirado na a estratégia de busca oportunística [Hayes-Roth 1979] -- para fazer a escolha da próxima ação a ser realizada (que corresponde, na modelagem feita, à uma transição da RdP que representa o domain-K). Para isso, faz o cálculo do indicador que está associado ao critério corrente para a escolha da ação.

Para o *i*-ésimo heliponto, usamos o critério de *mínimo custo de inserção*. Este critério foi inspirado na regra heurística do *vizinho-mais-próximo*, usada no Problema do Caixeiro Viajante (PCV) [Timlin et Pulley 1992]. Observe que a escolha da próxima

ação a ser realizada implica, também, na escolha da rota a ser estendida que está associada com esta ação. Então, faz-se a inserção do heliponto correspondente na rota escolhida e uma atualização do modelo de mundo (o procedimento de atualização do modelo de mundo está baseado nas tradicionais listas de adição e de eliminação, do STRIPS). Além do critério corrente medido pelo indicador, o Roteador considera a capacidade versus lotação em termos agregados e em cada trecho da rota gerada.

Para contornar o problema do mapeamento da RdP que foi descrito no item 3.4, não tentamos elaborar um conjunto de cláusulas de Horn “equivalentes” à Rede de Petri. Ao contrário, os elementos da RdP são preservados exatamente como são, i.e., os lugares e as transições da RdP são representadas através de uma estrutura de dados na forma de termos que são gravados na Base de Dados Global do sistema que foi escrito em Visual PROLOG.

Quando ainda existem requisições de viagens não atendidas e não existe nenhuma rota (de nenhum dos helicópteros) que possa ser estendida para incluí-las, possivelmente existe uma falta de capacidade nos helicópteros para rotear os helipontos em grupos num mesmo helicóptero. Daí, faz-se o desagrupamento dos grupos ainda não roteados, para que os helipontos sejam roteados em separado. Desagrupar significa não só eliminar a requisição para o grupo e inserir requisições para os helipontos em separado, mas também em se modificar a topologia (da estrutura) da RdP que representa o domínio. Isto implica em descartar a RdP do domain-K e elaborar novamente todas atividades da RdP e recalcular as distancias entre os lugares da RdP modificada. Quando não existir mais nenhuma requisição de viagem a ser atendida, todos os vôos são terminados, inserindo-se o trecho de retorno à base na rota.

5.4.1 - O Detalhador Baseado em Rede de Petri

Como descrevemos no capítulo 4, item 4.5.1., a nossa metodologia elabora um plano para cada nível de abstração. Um plano elaborado para o nível de abstração mais alto torna-se meta para o nível de abstração mais baixo.

Como mostra a Figura 44, depois que o sistema gera o plano de mais alto nível para o nível de abstração mais alto existente, faz sucessivamente o detalhamento para o nível mais baixo até chegar ao Nível 0 (ou nível “ground”). O processamento do Roteador é sempre o mesmo, não importa qual seja a combinação de níveis de abstração definida para o problema.

Cada rota do plano de mais alto nível é detalhada de cada vez, desmembrando os grupos deste nível em helipontos e grupos de helipontos na rota. O Detalhador de Rotas usa como entrada a rota de mais alto nível, gerando a rota detalhada (rota_Det). Note que o Detalhador deve tratar do problema capacitado onde a ordem parcial dentro do grupo é muito importante. Em outras palavras, o Detalhador deve elaborar o roteamento dentro do grupo, observando a capacidade versus lotação em cada trecho da rota detalhada. Após o término do detalhamento das rotas, calcula distância e duração de cada rota. Para o Nível 0, calcula ainda o custo unitário por passageiro em cada rota. O sistema invoca o Otimizador de Rotas baseado em PCV, após terminar o processo de detalhamento das rotas para cada nível de abstração mais baixo.

Tabela 1 - Demandas do caso P0612AM

Heliponto	Demanda Ida	Demanda Volta	Ordenada (m)	Abcissa (m)
ss20	4	3	390.033,000	7.529.733,000
reel	3	2	369.094,000	7.529.746,000
pch1	2	1	347.501,000	7.518.651,000
pna1	2	3	353.254,000	7.518.078,000
ppg1	2	1	362.828,000	7.538.445,000
pvm2	1	3	366.700,000	7.547.000,000
total	14	13		

5.4.2 - Como Funciona o Roteador

Seja o problema descrito com o número 1 na Tabela 4 das páginas 160-161, i.e., caso P0612AM e recurso (helicóptero) do tipo Bell-212 (b-212), que tem uma capacidade de 11 assentos. O caso P0612AM é um arquivo de requisições de viagens definido como mostra a Tabela 1, acima.

A demanda total de ida é de 14 passageiros e a de volta é de 13 passageiros, de modo que são necessários dois vôos. Existem dois níveis de abstração: Nível 2 e Nível 0. Os grupos de plataformas que pertencem ao Nível 2 e foram roteados como tal são: “pch1+ pna1”, distantes entre si de 5,78 km e “ppg1+pvm2”, distantes entre si de 9,39 km. Estes grupos tem demanda de ida e volta de (4 , 4) e (3, 4), respectivamente.

Portanto, as metas para este Nível 2 são:

metas:

heliponto: ss20	ida: 4	volta:3	heliponto: pch1+pna1	ida: 4	volta:4
heliponto: reel	ida: 3	volta:2	heliponto: ppg1+pvm2	ida: 3	volta:4

O Roteador para elabora a solução para o nível 2, cujas rotas (rotas RdP) são:

nível: 2 rota: 1 custo (U\$): 546.54
 helicóptero: b-212 capac.:11 custo (U\$/h): 374.00 veloc.(km/h): 185.2

horário	trecho	dista(km):	tempo voo:
08:00:00	mcae-pch1+pna1	135.32	00:43:50
08:43:50	pch1+pna1-mcae	135.32	00:43:50
		270.637	01:27:40

passageiros: mcae - pch1+pna1 - mcae

ida:	-	4	-
volta:	-	4	-
trecho:	4	-	4

nível: 2 rota: 2 custo (U\$): 726.38
 helicóptero: b-212 capac.:11 custo (U\$/h): 374.00 veloc.(km/h): 185.2

horário	trecho	dista(km):	tempo voo:
08:00:00	mcae-ppg1+pvm2	150.32	00:48:42
08:48:42	ppg1+pvm2-reel	13.68	00:04:25
08:53:07	reel-ss20	20.94	00:06:47
08:59:54	ss20-mcae	174.75	00:56:36
		359.695	01:56:31

passageiros: mcae - ppg1+pvm2 - reel - ss20 - mcae

ida:	-	3	-	4	-	3	-
volta:	-	4	-	3	-	2	-
trecho:	10	-	11	-	10	-	9

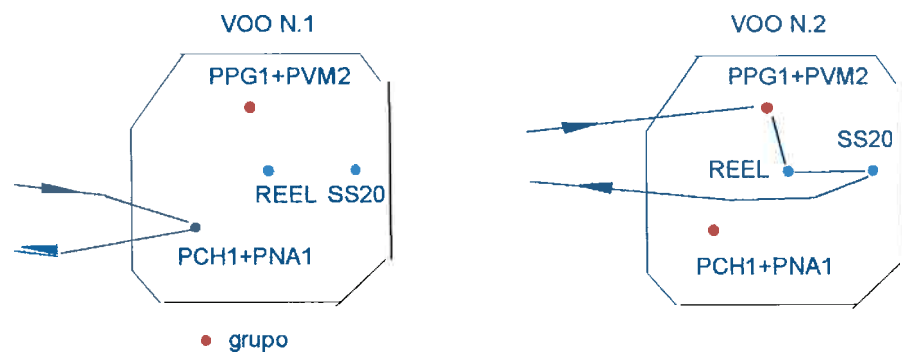


Figura 46 - O Plano RdP para o Nível 2 de abstração

A Figura 46, acima, ilustra as rotas (rota 1 e rota 2) do plano RdP para o Nível 2 de abstração. Para a rota 1, o número de passageiros no trecho inicial de ida é 4. O número máximo e mínimo de passageiros em cada trecho é 4. Para a rota 2, o número de passageiros no trecho inicial de ida é 10. O número máximo de passageiros em cada trecho é 11 e o número mínimo de passageiros em cada trecho é 9. Observe que a capacidade nunca ultrapassa a lotação em termos agregados e em cada trecho da rota gerada. Obtidas as rotas RdP, o sistema invoca o Otimizador baseado em PCV para este nível de abstração. Veja o item 5.5.1.

Não existem grupos de helipontos para o Nível 1, portanto o Detalhamento se faz a partir das rotas já otimizadas a partir do Nível 2 para o nível de detalhamento Nível 0. As metas para o Nível 0 correspondem às metas na Tabela 1:

metas:

heliponto: ss20	ida: 4	volta:3
heliponto: reel	ida: 3	volta:2
heliponto: pch1	ida: 2	volta:1
heliponto: pna1	ida: 2	volta:3
heliponto: ppg1	ida: 2	volta:1
heliponto: pvm2	ida: 1	volta:3

Para este problema de teste, as rotas detalhadas como solução para o Nível 0 são:

nível: 0 rota: 1 custo (U\$): 558.21
 helicóptero: b-212 capac.:11 custo (U\$/h): 374.00 veloc.(km/h): 185.2

horário	trecho	dista(km):	tempo voo:
08:00:00	mcae-pch1	132.43	00:42:54
08:42:54	pch1-pna1	5.78	00:01:52
08:44:46	pna1-mcae	138.21	00:44:46
		276.419	01:29:33

passageiros:	mcae	-	pch1	-	pna1	-	mcae
ida:	-	2	-	2	-	-	
volta:	-	1	-	3	-	-	
trecho:	4	-	3	-	4		

nivel: 0	rota: 2	custo (U\$):	729.36		
helicoptero: b-212	capac.:11	custo (U\$/h):	374.00	veloc.(km/h):	185.2
horário	trecho	dista(km):	tempo voo:		
08:00:00	mcae-ppg1	147.99	00:47:56		
08:47:56	ppg1-pvm2	9.39	00:03:02		
08:50:59	pvm2-ss20	29.03	00:09:24		
09:00:23	ss20-reel	20.94	00:06:47		
09:07:10	reel-mcae	153.82	00:49:49		
		361.169	01:57:00		

passageiros:	mcae	-	ppg1	-	pvm2	-	ss20	-	reel	-	mcae
ida:	-	2	-	1	-	3	-	4	-	-	
volta:	-	1	-	3	-	2	-	3	-	-	
trecho:	10	-	9	-	11	-	10	-	9		

Para a rota 1, o número de passageiros no trecho inicial de ida é 4. O número máximo de passageiros em cada trecho é 4 e o número mínimo de passageiros em cada trecho é 3. Para a rota 2, o número de passageiros no trecho inicial de ida é 10. O número máximo de passageiros em cada trecho é 11 e o número mínimo de passageiros em cada trecho é 9. Observe que a capacidade nunca ultrapassa a lotação em termos agregados e em cada trecho da rota gerada. Obtidas as rotas RdP, o sistema invoca o Otimizador baseado em PCV para este Nível 0.

5.5 - O Otimizador baseado no PCV

Faz a otimização de todas as rotas elaboradas pelo Roteador (rotas RdP) para um nível determinado nível de abstração. Para isto, emprega o algoritmo das 3-permutações para o Problema do Caixeiro Viajante (PCV) [Timlin et Pulley 1992]. As permutações podem ser de quatro tipos. A Figura 47, seguinte, ilustra os tipos de permutações de rotas implementadas no Otimizador.

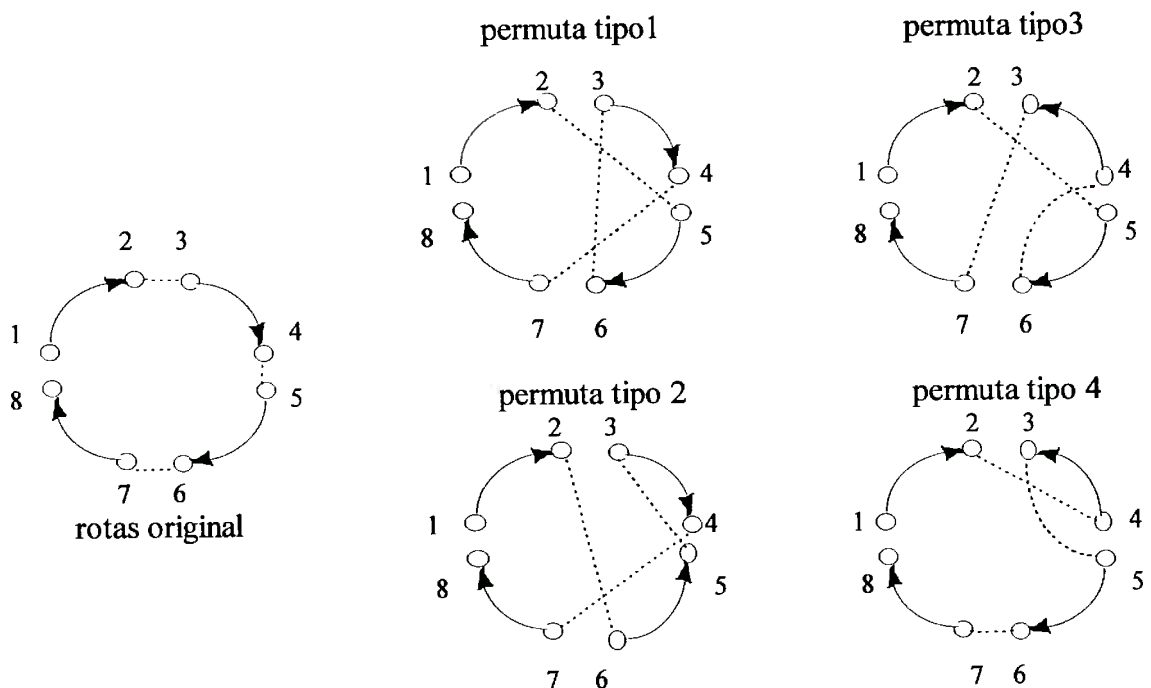


Figura 47 - Os Tipos de Permutação de rotas

Escolhe-se uma seqüência de nós consecutivos que constituem uma *janela*. Determina-se, então, se esta janela pode ser removida de sua posição corrente e pode ser reinserida entre dois nós da rota de modo que a distância total seja diminuída, sem que o limite de capacidade do helicóptero seja ultrapassado em nenhum trecho da rota permutada. Seja uma rota original com quatro janelas: 1-2, 3-4, 5-6 e 7-8. Estas janelas podem ser formadas por uma seqüência qualquer de helipontos, até mesmo por um heliponto isolado. Na permutação do tipo 1, a janela 3-4 é permutada com a janela 5-6, resultando na rota permutada 1-2, 5-6, 3-4, 7-8. Na permutação do tipo 2, a janela 3-4 é permutada com a janela 5-6 que tem o sentido de travessia invertido, resultando na rota

permutada 1-2, 6-5, 3-4, 7-8. Na permutação do tipo 3, a janela 3-4 tem o sentido de travessia invertido e é permutada com a janela 5-6 resultando na rota permutada 1-2, 5-6, 4-3, 7-8.

Na permutação do tipo 4, a janela 3-4 tem o sentido de travessia invertido e é permutada permanecendo na posição original, resultando na rota 1-2, 4-3, 5-6, 7-8.

Porém, não basta que permutação da rota corresponda a uma distância menor que a rota corrente; a permutação da rota deve ser factível quanto a capacidade do aparelho atribuído para esta rota não só em termos agregados como também em cada trecho da rota. O processo de permuta é repetido até que não exista mais nenhuma troca factível - busca gulosa ("greedy search"). Após efetuar todas permutações, o sistema verifica se é interessante e factível quanto à capacidade do aparelho fazer a inversão do sentido da rota. Após obtidas as rotas otimizadas, calculam-se as durações de todas as rotas.

5.5.1 - Como Funciona o Otimizador

Considere o exemplo dado no item 5.4.2. Cada vez que o sistema elabora / detalha uma solução para um determinado nível de abstração, ele invoca o Otimizador. Este emprega o algoritmo das 3-permutações de rotas baseado em PCV. O sistema tem a facilidade de gravar em arquivo as rotas finais elaboradas e otimizadas para qualquer nível de abstração do problema que está no banco de dados global do Visual PROLOG. Fazendo isto para o Nível 2, pode-se observar que estas rotas PCV são tais que a rota 1 é a mesma rota 1, na página 147, criada pelo Roteador baseado em RdP, apresentando o mesmo custo estimado de U\$ 546.54, percorrendo uma distância estimada de 270,637 km, com uma duração de 01:27:40, como mostramos a seguir:

nível: 2	rota: 1	custo (U\$):	546.54	
helicoptero:	b-212	capac.:11	custo (U\$/h):	374.00
			veloc.(km/h):	185.2
horário	trecho	dista(km):	tempo voo:	
08:00:00	mcae-pch1+pna1	135.32	00:43:50	
08:43:50	pch1+pna1-mcae	135.32	00:43:50	
		270.637	01:27:40	

passageiros: mcae - pch1+pna1 - mcae
 ida: - 4 -
 volta: - 4 -
 trecho: 4 - 4

Quanto à rota 2, esta foi otimizada. Observe que podemos particionar a rota RdP para a rota 2, na página 147, para compor as janelas como: 1-2: mcae-ppg1+pvm2, 3-4: reel, 5-6: ss20, 7-8: mcae. Ao aplicarmos a permutação do tipo 1, cf. Figura 47, obtém-se a rota permutada : 1-2: mcae-ppg1+pvm2, 5-6: ss20, 3-4: reel, 7-8: mcae, que é a rota 2 otimizada para este nível de abstração.

nível: 2 rota: 2 custo (U\$): 713.86
 helicóptero: b-212 capac.:11 custo (U\$/h): 374.00 veloc.(km/h): 185.2
 horário trecho dista(km): tempo voo:
 08:00:00 mcae-ppg1+pvm2 150.32 00:48:42
 08:48:42 ppg1+pvm2-ss20 28.41 00:09:12
 08:57:54 ss20-reel 20.94 00:06:47
 09:04:41 reel-mcae 153.82 00:49:49
 353.492 01:54:31

passageiros: mcae - ppg1+pvm2 - ss20 - reel - mcae
 ida: - 3 - 3 - 4 -
 volta: - 4 - 2 - 3 -
 trecho: 10 - 11 - 10 - 9

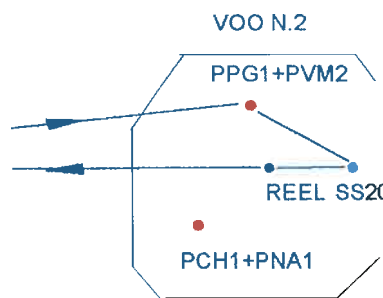


Figura 48 - A rota 2 otimizada para o Nível 2

A rota 2 otimizada para o Nível 2 tem uma distância estimada de 353,492 km que são percorridos em 01:54:31, enquanto a rota 2 RdP, na página 147, tem uma distância estimada de 359,695 km que são percorridos em 01:56:31. Deste modo, a distância total (estimada) das rotas para o Nível 2 foi reduzida de 630,332 km para 624,129 km e a duração da viagem foi reduzida em 2 minutos pelo Otimizador ao permutar a rota 2.

Finalmente, consultando do mesmo modo a solução otimizada gravada em arquivo para o Nível 0, pode-se observar que estas rotas são as mesmas rotas elaboradas pelo Detalhador baseado em Rede de Petri. descritas nas páginas 148-149, i.e., neste caso o Roteador baseado em RdP já gerou as rotas otimizadas. Esta solução para o Nível 0 tem custo total de U\$ 1287.57 e implica numa distancia total (exata) de 637,588 km. Além da listagem da rotas elaboradas e do número de passageiros na rota, o sistema apresenta outros indicadores acerca das rotas, como: tempo de atendimento de passageiros na ida e na volta e custo das viagens em cada trecho da rota.

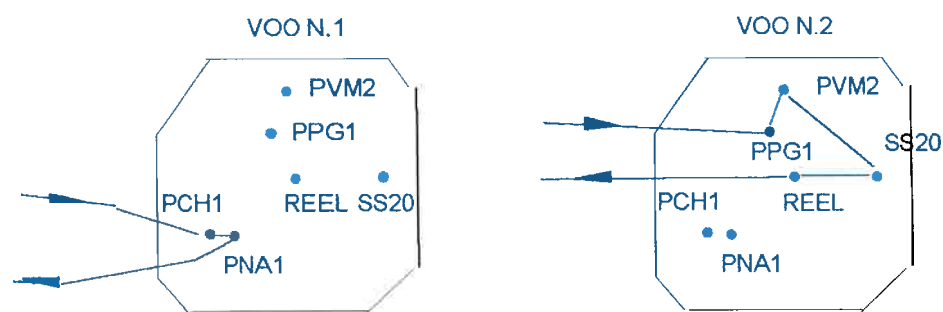


Figura 49 - O Plano PCV para o Nível 0 (“ground”) de abstração

problema: P0612AM horário: 08:00

grade de horário: como mostrado nas páginas 148-149.

rota: 1 Duração das Viagens (hh:mm:ss)

	trecho:	ida:	volta:	
mcae-pch1	00:42:54	00:42:54		tempo médio atendim. ida : 00:43:50
pch1-pna1	00:01:52	00:44:46	00:46:38	tempo médio atendim. volta : 00:45:14
pna1-mcae	00:44:46		00:44:46	
total	01:29:33			

rota: 2 Duração das Viagens (hh:mm:ss)

	trecho:	ida:	volta:	
mcae-ppg1	00:47:56	00:47:56		tempo médio atendim. ida : 00:58:59
ppg1-pvm2	00:03:02	00:50:59	01:09:03	tempo médio atendim. volta : 00:59:37
pvm2-ss20	00:09:24	01:00:23	01:06:01	
ss20-reel	00:06:47	01:07:10	00:56:36	
reel-mcae	00:49:49		00:49:49	
total	01:57:00			

tempo médio atendim. global ida: 00:54:40

tempo médio atendim.global volta: 00:55:12

rota: 1	Custo Unitário (U\$ / passag.)			rota: 2	Custo Unitário (U\$ / passag.)		
	trecho:	ida:	volta:		trecho:	ida:	volta:
mcae-pch1	66.86	66.86		mcae-ppg1	29.89	29.89	
pch1-pna1	3.89	70.75	73.67	ppg1-pvm2	2.11	31.99	46.18
pna1-mcae	69.78		69.78	pvm2-ss20	5.33	37.32	44.07
				ss20-reel	4.23	41.55	38.74
				reel-mcae	34.51		34.51

RESUMO NOSSO ROTEADOR:

custo total (U\$): 1287.57
 distância total: 637.588 km
 n.passag.: 27
 custo unitário (U\$/passag.): 47.69

No resumo acima, estão mostrados estes indicadores da solução para o Nível 0, para o problema de número 1. A Figura 49, na página anterior, ilustra as rotas (rota 1 e rota 2) do Plano PCV para o Nível 0 (“ground”) de abstração - que é idêntico ao Plano RdP.

5.6 - Resultados Computacionais

Foram usados cinco conjuntos de requisições de viagens, gravados em arquivo na forma de termos na linguagem Visual PROLOG, que formam casos de teste. Estes estão indicados na Tabela 2, a seguir.

Tabela 2 - Casos de teste do Roteador

Caso de teste	Data e Hora do Vôo
P0612AM	06/12/95 08:00
P0612PM	06/12/95 18:00
P0712AM	07/12/95 09:00
P0712PM	07/12/95 19:00
P2204AM	22/04/95 08:00

Estes casos foram testados usando-se diferentes combinações da frota de helicópteros cadastrados. O tipos e características dos helicópteros cadastrados estão indicados na Tabela 3, a seguir.

Tabela 3 - Características dos Helicópteros Cadastrados

TipoH	N. Assentos	Custo (U\$/h)	Veloc. (km/h)
b-212	11	374.00	185,20
b-222	7	428.00	244,46
b-412	13	506.00	222,24
dauphin	12	608.00	259,28
s-61	26	603.00	203,72
s-76	12	454.00	237,06
twin	5	332.00	207,42

5.6.1 - O Sistema de Referência

Para efeito de comparação, utilizamos como referência um sistema existente de Programação de Helicópteros escrito em FORTRAN para mainframe IBM e utiliza a abordagem clássica baseada no algoritmo de Clark & Wright, [Clark et Wright 1964], [Brinati 1993] resolvendo uma seqüência de k PCVs - Problema do Caixeiro Viajante - sucessivos, onde k é o número de vôos e não está baseado em IA. O objetivo principal neste sistema é o de obter soluções de mínimo custo.

Existe uma crítica possível a esta abordagem usada pelo sistema de referência, porque o roteamento de helicópteros deve tratar do problema capacitado [Timlin et Pulley 1992]. É óbvio que, se não houvesse nenhuma restrição de capacidade nos aparelhos, a solução para o PCV implementada no mainframe sempre obteria a solução ótima (a solução teria um único vôo), i.e., a solução que corresponde ao caminho mínimo e ao custo mínimo. Entretanto, ao abordar o problema capacitado através da solução de k PCVs resolvidos sucessivamente, de uma forma procedimental, devido à natureza combinatória deste tipo de problema, tem-se que as possíveis soluções para os $k-1$ PCVs sucessivos dependem da solução dada ao primeiro PCV. Ao dividir o problema original em subproblemas, o sistema mainframe não é mais capaz de resgatar a dependência que existe entre estes subproblemas.

A principal desvantagem desta abordagem usando um algoritmo de otimização está na excessiva rigidez da solução dada pelo sistema. O módulo planejador não está acoplado com o módulo de execução do plano. Em consequência, mesmo para tratar de pequenas modificações ou “perturbações” ao plano, é preciso refazer todo o processo a partir do princípio.

O nosso Roteador é capaz de tratar adequadamente a dependência que existe entre estes subproblemas. Ao contrário do sistema comercial de referência, ele elabora simultaneamente a solução para todas as k rotas em paralelo. Além disso, é mais flexível porque foi concebido desde o princípio para tratar adequadamente modificações ao plano corrente sem a necessidade de refazer todo o processo de planejamento.

5.6.1.1 O Problema de Atribuição na PO

Na Pesquisa Operacional (PO), o Problema do Caixeiro Viajante (PCV) constitui um caso especial do problema de programação linear denominado como Problema de Atribuição (ou “assignment problem”).

Suponha que um gerente tem três vagas de emprego, e_1 , e_2 e e_3 e três pessoas candidatas para preencher estas vagas, p_1 , p_2 e p_3 . Queremos fazer a atribuição (“assignment”) de modo a maximizar o valor total do desempenho levando-se em conta que a produtividade de $([c_{ij}] (i=1,2,3, j=1,2,3))$ varia em cada par de atribuição pessoa-emprego $[x_{ij}]$. O procedimento trivial de se fazer a enumeração completa de todas as possíveis soluções é fácil de se fazer para três empregos, mas, para 10 empregos, seria necessário considerar mais de três milhões ($10!$) de possíveis atribuições. Se cada candidata pudesse ser avaliada manualmente em, por exemplo apenas um segundo, esta tarefa iria exigir um tempo total de processamento de 800 horas.

Seja $C = [c_{ij}]$ a produtividade associada com a atribuição da pessoa i ($i = 1, \dots, n$) à vaga de emprego j ($j = 1, \dots, n$). Seja $[x_{ij}]$ a variável de decisão associado a cada par de atribuição candidato-emprego, tal que:

$$[x_{ij}] = \begin{cases} 1 & \text{caso o candidato } p_i \text{ seja atribuído para a vaga de emprego } e_j \\ 0 & \text{caso contrário} \end{cases}$$

Lembrando que um problema de maximização pode ser sempre traduzido para um problema de minimização equivalente, ou seja, $\max z = \min (-z)$. O problema geral de atribuição é definido na PO [Spivey et Thrall 1970], como o problema de:

$$\text{minimizar } z = \sum_i \sum_j c_{ij} x_{ij}$$

sujeito às restrições:

$$\sum_{j=1}^n x_{ij} = 1 \quad (i = 1, \dots, n)$$

$$\sum_{i=1}^n x_{ij} = 1 \quad (j = 1, \dots, n)$$

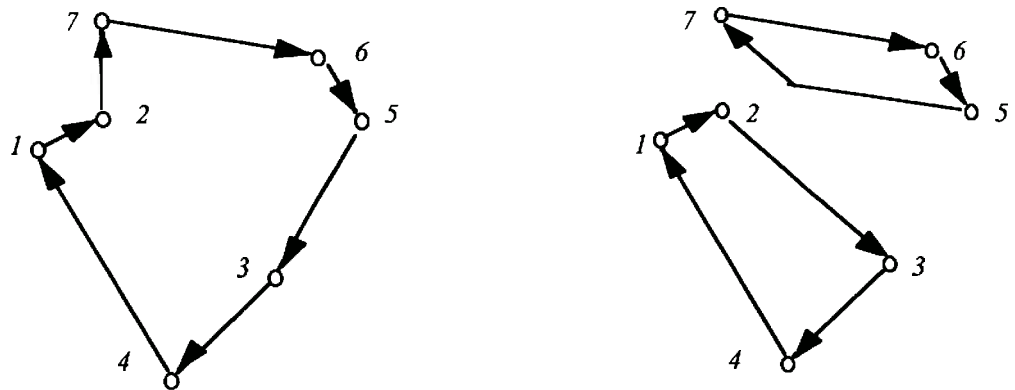
$$x_{ij} \geq 0$$

Uma característica interessante deste problema de atribuição é que uma solução ótima $\mathbf{x}^* = [x_{ij}^*]$ para o problema de atribuição, como definido acima, resulta sempre numa solução inteira em que os valores de cada variável $x_{i,j}$ sempre pertence a $\{0, 1\}$, mesmo que esta limitação não seja explicitamente incluída no equacionamento [Spivey et Thrall 1970]. Isto torna o problema de atribuição relativamente fácil de ser resolvido (porque pode ser resolvido usando-se o método de resolução simplex, usual para resolver problemas contínuos, onde $\mathbf{x} \in \mathbf{R}^N$).

Entretanto, isto não ocorre com o Problema do Caixeiro Viajante (PCV), que não apresenta esta propriedade. Por isto, o PCV exige algoritmos especializados para o tratamento da programação inteira para a sua resolução, por exemplo o método clássico na PO, Ramificar e Podar (“Branch & Bound”).

5.6.1.2 O Problema do Caixeiro Viajante na PO

Um dos problemas mais conhecidos na Pesquisa Operacional (PO) é o Problema do Caixeiro Viajante (PCV) que quer visitar um número n de cidades, sem repetir nenhuma delas, partindo e retornando à sua cidade de origem, com um mínimo custo (ou distância).



(a) uma solução factível

(b) uma solução infactível

Figura 50 - Exemplo do Problema de Caixeiro Viajante (PCV)

Nesse caso, temos que $C = [c_{ij}]$ é o custo de se ir da cidade i ($i = 1, \dots, n$) para a cidade j ($j = 1, \dots, n$). Seja $[x_{ij}]$ a variável de decisão associado inclusão de cada trecho (ou *pernada*, no caso do plano de vôos) $i-j$ na rota, tal que:

$$[x_{ij}] = \begin{cases} 1 & \text{caso o trecho } i-j \text{ seja incluído na rota} \\ 0 & \text{caso contrário} \end{cases}$$

O Problema do Caixeiro Viajante (PCV) é definido na PO [Kusiak 1990], como o problema de:

$$\text{minimizar } z = \sum_i \sum_j c_{ij} x_{ij} \quad (1)$$

sujeito às restrições:

$$(2) \quad \sum_{j=1}^n x_{ij} = 1 \quad (i = 1, \dots, n)$$

$$(3) \quad \sum_{i=1}^n x_{ij} = 1 \quad (j = 1, \dots, n)$$

$$(4) \quad u_i - u_j + n x_{ij} \leq n - 1 \quad (i = 2, \dots, n, j = 2, \dots, n, i \neq j)$$

$$(5) \quad x_{ij} \in \{0, 1\} \quad (i = 1, \dots, n, j = 1, \dots, n)$$

$$(6) \quad u_i \geq 0 \quad (i = 1, \dots, n)$$

A equação (1), corresponde ao custo total (ou a distância total) associada à rota gerada. A equação (2) assegura que num dado “tour”, apenas uma cidade i é visitada antes de cada cidade j . A equação (3) assegura que apenas uma cidade j é visitada após cada cidade i . A equação (4) elimina ciclos ou subsequências não conectadas da rota gerada. A equação (5) estabelece que $x_{ij} \in \{0, 1\}$. Finalmente, a equação (6) define que u_i é uma variável auxiliar, não negativa.

A Figura 50.a, na página anterior, mostra um exemplo de rota factível para o problema envolvendo sete cidades. Entretanto, usando-se a formulação geral do problema de atribuição, descrita acima, pode também ocorrer uma solução que é infactível para o PCV, porque apresenta sub-rotas (ou “tours”, ou ciclos) não conectados, como mostra a Figura 50.b. Portanto, a definição do problema tem de ser modificada para não permitir estes casos.

Em termos puramente matemáticos, a solução do problema de programação inteira acima seria de fácil resolução, já que o espaço das possíveis soluções é finito para um determinado número de cidades. O problema seria o de escolha trivial entre os muitos caminhos possíveis de modo a minimizar os custos. Entretanto, é a natureza combinatória característica deste problema que torna sua solução complexa -- um problema simples envolvendo apenas 13 cidades tem mais de seis bilhões ($13!$) de possíveis caminhos.

Apresentamos a seguir os resultados computacionais obtidos com o Roteador implementado usando a linguagem Visual PROLOG para Windows 3.11. Como referência para a linguagem PROLOG, veja [Clocksin et Mellish 1987], [Sterling et Shapiro 1986] e [Araribóia 1989]. Para obter referência do Visual PROLOG, veja [PDC 1996].

5.6.2 O Custo e a Distância das Rotas Geradas

A Tabela 4, nas páginas 160-161, mostra os principais indicadores obtidos na análise comparativa entre o Nosso Roteador e o sistema Mainframe de referência. Cada teste consiste de um arquivo de casos e de recursos disponíveis. Os indicadores são o número de rotas geradas para atender toda demanda existente, o custo total (U\$) das rotas geradas e a distância total (km) percorrida pelos helicópteros em todas as rotas geradas, para o nosso Roteador e também para o sistema em mainframe.

Tabela 4 - Análise dos Resultados Nosso Roteador X Mainframe

ANÁLISE DOS RESULTADOS NOSSO ROTEADOR X MAINFRAME									
CASO	RECURSOS	N.Pass.	número rotas		custo total (U\$)		distância total (km)		
			ROTEADOR	Mainframe	ROTEADOR	Mainframe	ROTEADOR	Mainframe	
1 P0612AM	b-212	27	2	2	1287,57	1345,08	637,588	666,067	
2	b-222	27	3	3	1649,96	1651,13	942,403	943,091	
3	b-412	27	2	2	1451,67	1434,60	637,588	630,092	
4	dauphin	27	2	2	1495,12	1573,84	637,588	671,161	
5	s-76	27	2	2	1221,06	1223,20	637,588	638,746	
6	twin	27	4	3	1990,70	1569,88	1243,705	980,819	
7	b212+b222	27	2	2	1263,93	1263,01	672,377	666,067	
8	b212+b412	27	2	2	1358,71	1366,43	637,588	630,092	
9	b212+dph	27	2	2	1405,14	1459,41	637,588	671,161	
10	b-212+s76	27	2	2	1258,74	1250,92	637,588	638,746	
11	b212+twin	27	2	2	1227,37	1217,14	680,262	666,067	
12	dph+b222	27	2	2	1369,03	1383,93	672,377	671,161	
13	dph+twin	27	2	2	1335,03	1335,93	680,262	671,161	
14	s61+twin	27		2		1567,46		670,840	
15	s76+b222	27	2	2	1230,25	1179,76	672,377	638,746	
16 P0612PM	b-212	27	2	2	972,60	980,65	481,620	485,608	
17	b-222	27	3	3	1266,16	1257,95	723,192	718,513	
18	b-412	27	2	2	1096,56	1084,52	481,620	476,332	
19	dauphin	27	2	2	1129,38	1138,73	481,620	485,608	
20	s-76	27	2	2	922,36	912,25	481,620	476,332	
21	twin	27	4	3	1504,99	1198,24	940,258	748,625	
22	b212+b222	27	2	2	953,88	919,10	504,831	485,608	
23	b212+b412	27	2	2	1031,63	1029,05	481,620	476,332	
24	b212+dph	27	2	2	1054,72	1055,24	481,620	485,608	
25	b-212+s76	27	2	2	948,68	934,73	481,620	476,332	

Tabela 4 - continua....

CASO	RECURSOS	NumPass	número rotas		custo total (U\$)		distância total (km)		Custo unitario (U\$/pass)	
			ROTEADOR	Mainframe	ROTEADOR	Mainframe	ROTEADOR	Mainframe	ROTEADOR	Mainframe
26	b212+twi	27	2	2	917,21	894,69	504,831	485,608	33,97	32,77
27	dph+b222	27	2	2	1088,72	1002,59	504,831	485,608	38,47	37,13
28	dph+twi	27	2	2	1002,05	968,18	504,831	485,608	37,11	35,86
29	s61+twi	27		2		1168,41		508,789		
30	s76+b222	27	2	2	926,69	876,82	504,831	476,332	34,32	32,47
31	F0712AM	33	2	2	1244,96	1511,27	616,490	748,362	37,73	45,80
32	b-222	33	3(0)	3	1651,07	1667,66	943,041	952,533	50,03	50,54
33	b-412	33	2	2	1403,64	1630,21	616,490	716,004	42,53	49,40
34	dauphin	33	2	2	1445,64	1754,87	616,490	748,362	43,81	53,18
35	s-76	33	2	2	1180,66	1371,26	616,490	716,004	35,78	41,56
36	twi	33	5	5	2390,09	2482,69	1493,233	1551,116	72,43	75,23
37	b212+b222	33	3	3	1683,48	1951,78	924,896	1057,104	51,01	59,14
38	b212+b412	33	2	2	1306,45	1533,12	616,490	716,004	39,59	46,46
39	b212+dph	33	2	2	1357,88	1632,46	616,490	748,362	41,45	49,47
40	b-212+s76	33	2	2	1220,05	1410,60	616,490	716,004	36,97	42,75
41	b212+twi	33	3(0)	3	1644,82	1897,32	955,106	1086,977	49,84	57,49
42	dph+b222	33	3	3	1761,24	2074,20	924,896	1057,104	53,37	62,86
43	dph+twi	33	3	3	1722,58	2019,73	955,106	1086,977	52,20	61,20
44	s61+twi	33		3		2296,07		1114,999		
45	s76+b222	33	3	2	1658,56	1309,25	924,896	716,004	50,26	39,67
46	F0712FM	33	2	2	1237,68	1409,50	612,882	697,966	37,51	42,71
47	b-222	33	3(0)	3	1559,26	1566,56	880,599	894,784	47,25	47,47
48	b-412	33	2	2	1394,45	1525,02	612,466	669,802	42,26	46,21
49	dauphin	33	2	2	1436,18	1436,18	612,466	612,466	43,52	43,52
50	s-76	33	2	2	1172,98	1282,76	612,466	669,802	35,54	38,87
51	twi	33	5	5	2237,69	2284,24	1398,019	1427,130	67,81	69,22
52	b212+b222	33	3	3	1600,92	1676,19	877,738	907,505	48,51	50,79
53	b212+b412	33	2	2	1300,82	1428,40	612,466	669,802	39,42	43,28
54	b212+dph	33	2	2	1359,42	1317,76	612,882	612,466	41,19	39,93
55	b-212+s76	33	2	2	1210,88	1321,90	612,466	669,802	36,69	40,06
56	b212+twi	33	4	4	1981,47	1924,39	1175,430	1117,217	60,04	58,31
57	dph+b222	33	2(0)	2	1220,03	1220,02	612,466	612,466	36,97	36,97
58	dph+twi	33	3	3	1592,04	1641,87	879,008	910,148	48,24	49,75
59	s61+twi	33		3		1904,01		912,624		
60	s76+b222	33	2(0)	2	1113,51	1221,11	612,466	669,802	33,74	37,00
61	P2204AM	29	2	2	1092,08	1092,46	540,783	540,967	37,66	37,67
62	b-222	29	3	3	1390,56	1404,58	788,530	802,264	47,61	48,43
63	b-412	29	2	2	1231,26	1233,59	540,783	541,804	42,46	42,54
64	dauphin	29	2	2	1268,11	1393,00	540,783	594,041	43,73	48,03
65	s-76	29	2	2	1036,67	1137,68	540,783	594,041	35,71	39,23
66	s-61	29	1	1	915,84	915,84	309,412	309,411	31,58	31,58
67	twi	29	4	4	1628,03	1683,82	1017,126	1052,004	55,14	58,06
68	b212+b222	29	2	2	1038,72	1027,88	551,556	540,957	35,82	35,44
69	b212+b412	29	2	2	1154,66	1178,09	540,783	541,804	39,82	40,62
70	b212+dph	29	2	2	1188,96	1299,80	540,783	594,041	41,00	44,82
71	b-212+s76	29	2	2	1066,71	1167,54	540,783	594,041	36,78	40,26
72	b212+twi	29	2(0)	2	966,73	991,79	551,556	540,957	34,37	34,20
73	dph+b222	29	2	2	1127,24	1222,88	551,556	594,041	38,87	42,17
74	dph+twi	29	2	2	1036,25	1179,88	551,556	594,041	37,42	40,69
75	s76+b222	29	2	2	1010,35	1090,62	551,556	594,041	34,84	37,61

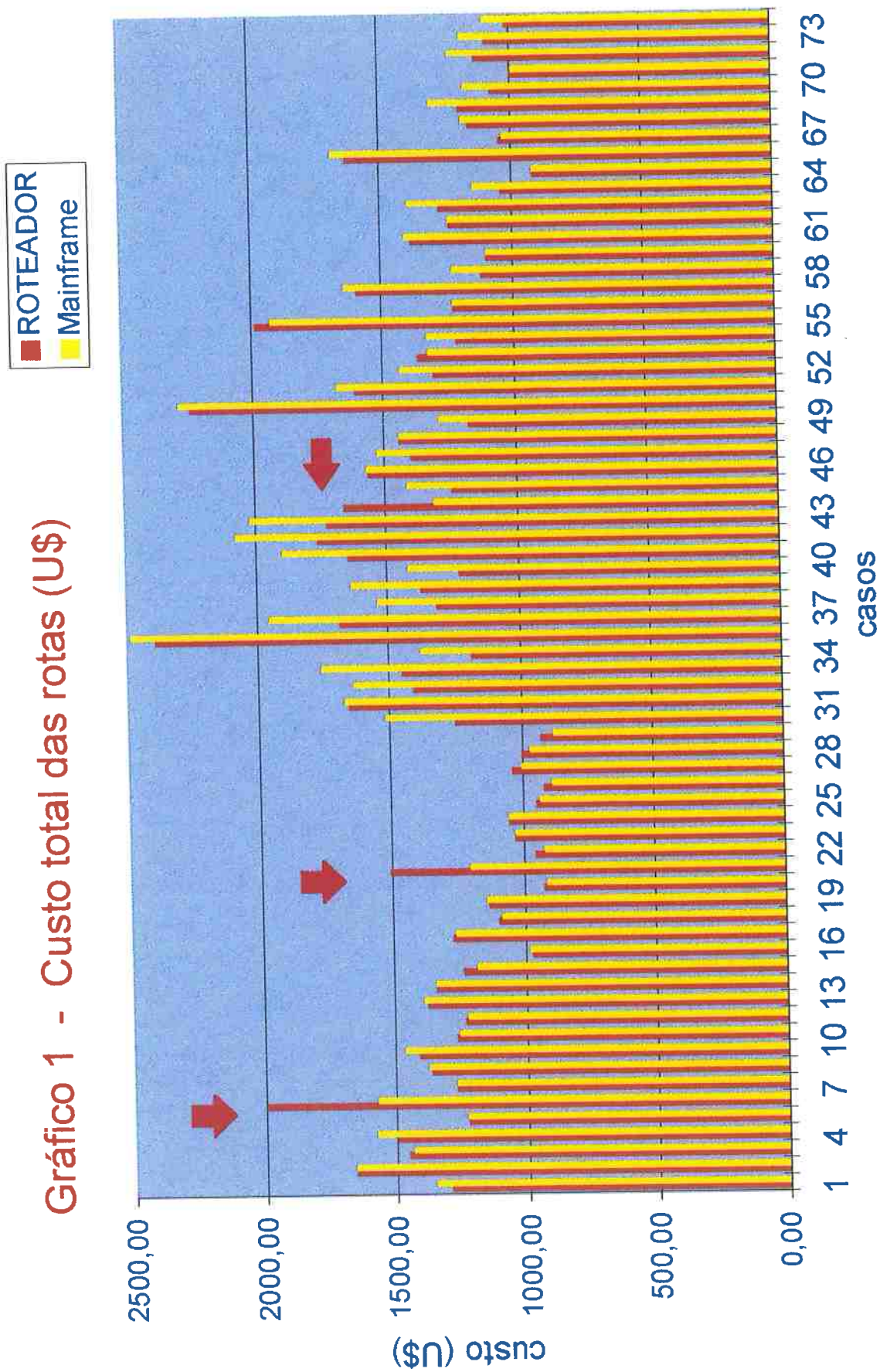
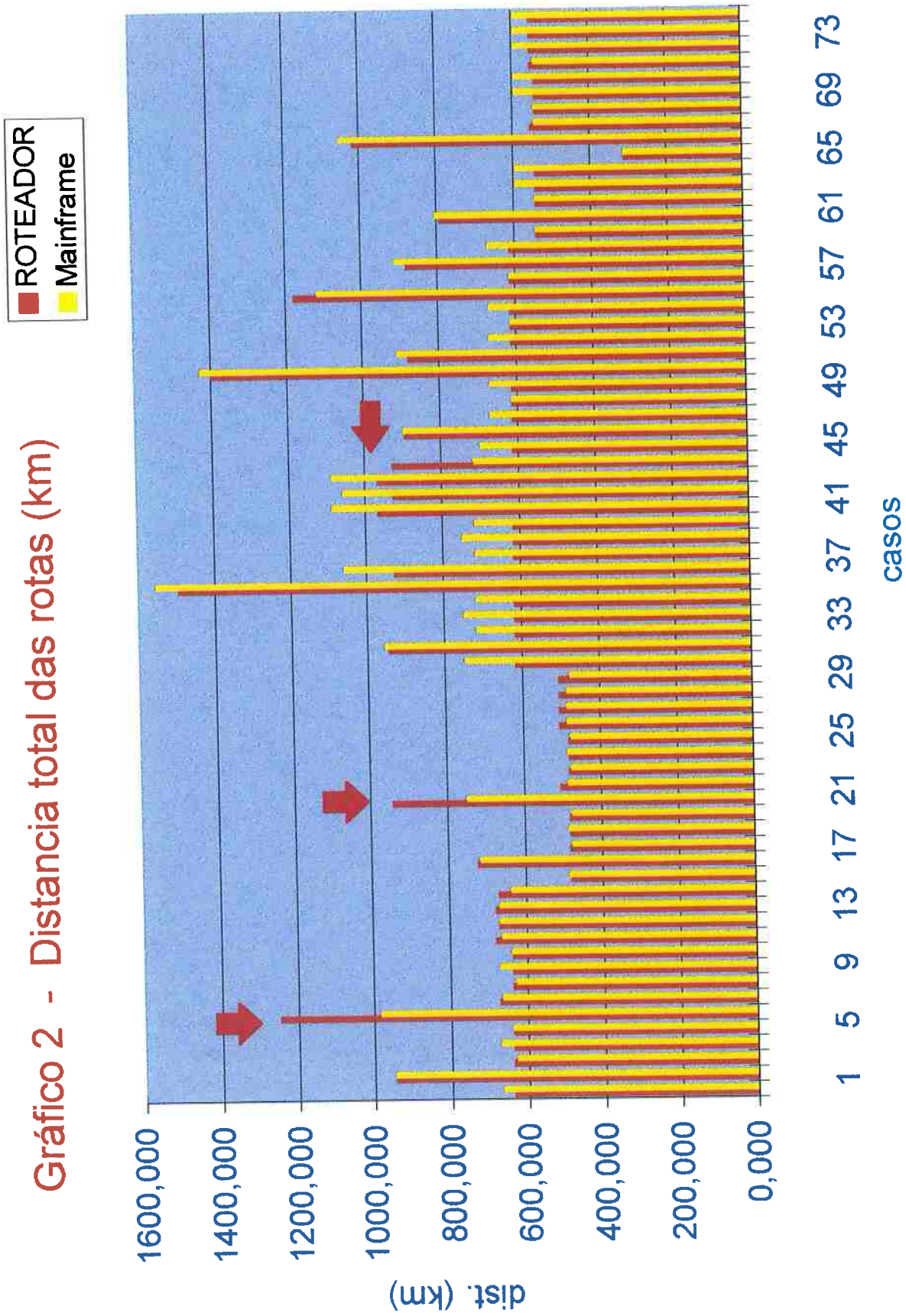


Gráfico 2 - Distancia total das rotas (km)



Por exemplo, seja o problema de número 1, que deveria atender requisições de viagens gravadas no arquivo P0612AM. (Veja os casos de teste na Tabela 2, na página 154). Só poderiam ser utilizados helicópteros do tipo Bell-212 (b-212). O helicóptero do tipo b-212 tem 11 assentos, voa a uma velocidade de 185,20 km / h e a hora de vôo custa U\$ 374.00 / h. A solução apresentada pelo nosso Roteador foi: número de rotas geradas: 2 rotas; custo total das rotas U\$ 1,287.56; distância total das rotas 637,588 km. A solução apresentada pelo sistema mainframe de referência foi: número de rotas geradas: 2 rotas; custo total das rotas U\$ 1,345.08; distância total das rotas 666,067 km.

De um modo geral, a Tabela 4 apresenta valores próximos para os indicadores de custo total e distância total das rotas geradas pelo Nosso Roteador e pelo sistema comercial de referência. Isto pode ser melhor observado nos Gráficos 1 e 2. O Gráfico 1, na página 162, ilustra o custo total das rotas (U\$). Do mesmo modo, o Gráfico 2, na página 163, mostra a distância total das rotas (km).

O casos de teste em que o nosso Roteador apresentou uma solução pior em termos de custo das rotas geradas (e também em termos de distância percorrida) foram: 6, 21 e 45. Nestes casos o nosso Roteador utilizou um vôo a mais em relação à solução dada pelo sistema de referência porque não usamos uma abordagem de otimização do custo das rotas. Ao invés, o nosso objetivo foi o de minimizar o tempo de atendimento.

5.6.3 Os Custos Unitários das Rotas Geradas

A Tabela 5, na página seguinte, indica os custos unitários (U\$ / passageiro) para cada uma dos casos de teste constantes da Tabela 4. No Gráfico 3, na página 166, ilustramos o custo unitário das rotas (U\$/passag.). Este indicador permite se fazer a comparação segundo uma única dimensão, porque em cada rota da Tabela 2 foi empregado um tipo de helicóptero diferente. E cada tipo de helicóptero tem características diferentes de capacidade, velocidade e de custo. De um modo geral, a Tabela 5 apresenta valores próximos para os indicadores de custo unitário das rotas geradas pelo Nosso Roteador e pelo sistema mainframe de referência.

Tabela 5 - A Comparação dos Custos Unitários das rotas

Custo unitário (U\$/pass.)		
	Roteador	Mainframe
1	47,69	49,82
2	61,11	61,15
3	53,77	53,13
4	55,37	58,29
5	45,22	45,30
6	73,73	58,14
7	46,81	46,78
8	50,32	50,61
9	52,04	54,05
10	46,62	46,33
11	45,46	45,08
12	50,70	51,26
13	49,45	49,48
14		
15	45,56	43,69

Custo unitário (U\$/pass.)		
	ROTEADOR	Mainframe
16	36,02	36,32
17	46,89	46,59
18	40,61	40,17
19	41,83	42,18
20	34,16	33,79
21	55,74	44,38
22	35,33	34,04
23	38,21	38,11
24	39,06	39,08
25	35,14	34,62
26	33,97	32,77
27	38,47	37,13
28	37,11	35,86
29		
30	34,32	32,47

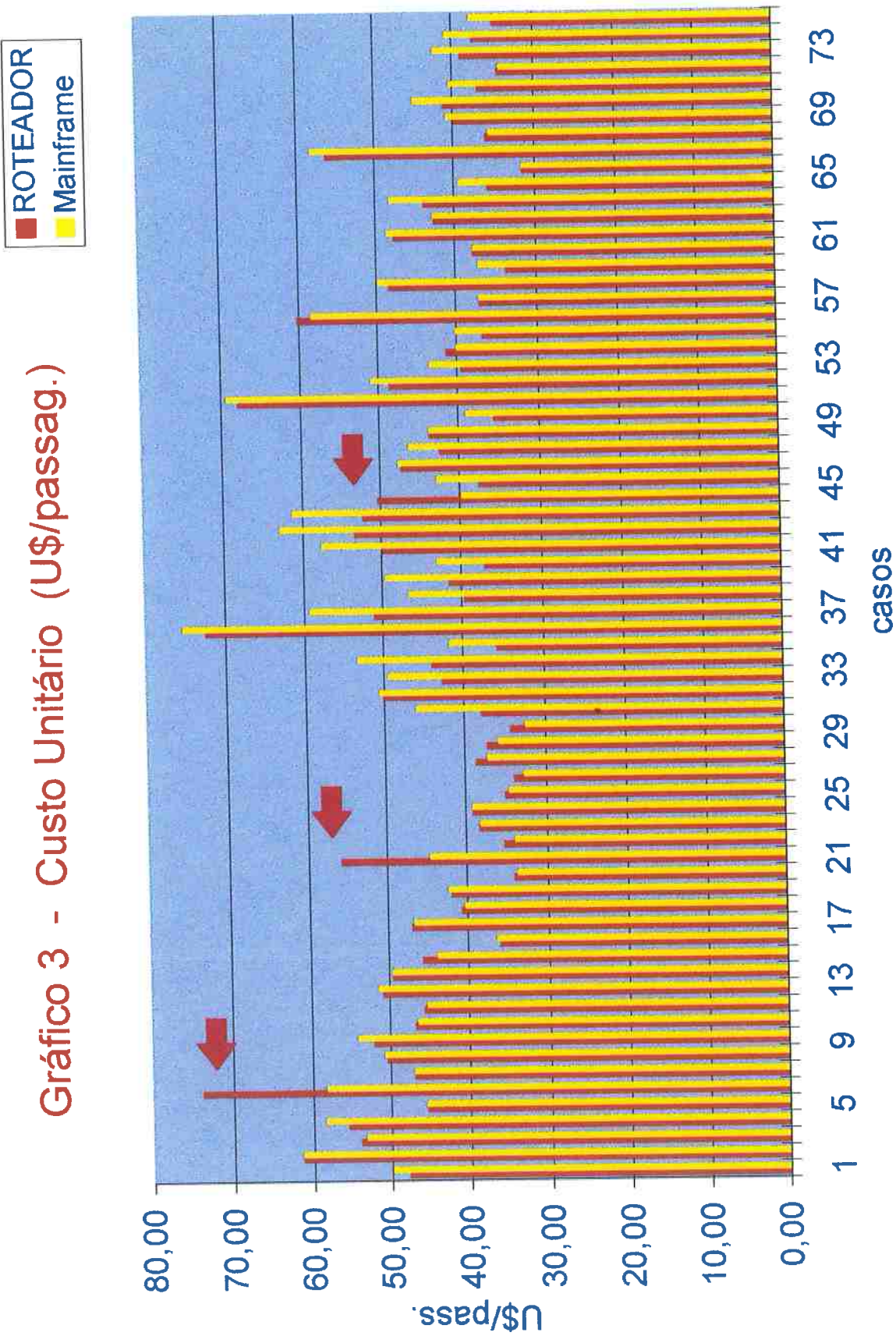
Custo unitário (U\$/pass.)		
	Roteador	Mainframe
31	37,73	45,80
32	50,03	50,54
33	42,53	49,40
34	43,81	53,18
35	35,78	41,55
36	72,43	75,23
37	51,01	59,14
38	39,59	46,46
39	41,45	49,47
40	36,97	42,75
41	49,84	57,49
42	53,37	62,85
43	52,20	61,20
44		
45	50,26	39,67

Custo unitário (U\$/pass.)		
	Roteador	Mainframe
46	37,51	42,71
47	47,25	47,47
48	42,26	46,21
49	43,52	43,52
50	35,54	38,87
51	67,81	69,22
52	48,51	50,79
53	39,42	43,28
54	41,19	39,93
55	36,69	40,06
56	60,04	58,31
57	36,97	36,97
58	48,24	49,75
59		
60	33,74	37,00

Custo unitário (U\$/pass.)		
	Roteador	Mainframe
61	37,66	37,67
62	47,61	48,43
63	42,46	42,54
64	43,73	48,03
65	35,71	39,23
66	31,58	31,58
67	56,14	58,06
68	35,82	35,44
69	39,82	40,62
70	41,00	44,82
71	36,78	40,26
72	34,37	34,20
73	38,87	42,17
74	37,42	40,69
75	34,84	37,61

O casos de teste em que o nosso Roteador apresentou uma solução pior em termos de custo unitário das rotas geradas foram: 6, 21 e 45. Nestes casos o nosso Roteador utilizou um vôo a mais em relação à solução dada pelo sistema de referência.

O motivo porque o nosso Roteador apresenta um desempenho tão favorável em relação ao sistema mainframe de referência é porque o problema do roteamento de helicópteros deve tratar do problema capacitado. Ao abordar o problema capacitado através da solução de k PCVs resolvidos sucessivamente, de uma forma procedimental, devido à natureza combinatória deste



tipo de problema, tem-se que as possíveis soluções para os $k-1$ PCVs sucessivos dependem da solução dada ao primeiro PCV. Ao dividir o problema original em subproblemas, o sistema mainframe não é mais capaz de resgatar a dependência que existe entre estes subproblemas [Timlin et Pulley 1992]. (Veja item 5.6.1).

Ao contrário, o nosso Roteador, pode elaborar uma solução para todas a k rotas em paralelo, porque a Rede de Petri permite isto, e é capaz de tratar adequadamente a dependência que existe entre estes subproblemas. Isto explica porque, eventualmente, o nosso Roteador consegue obter uma solução melhor do que a solução apresentada pelo sistema mainframe de referência, por exemplo como nos casos: 1,2,4,5,9,16,19,....

Observando a Tabela 4, podemos verificar que ela não tem resultados indicados para o nosso Roteador nos casos de teste de números 14, 29, 44 e 60. Estes casos envolvem o helicóptero do tipo s-61 (que tem limitação de pouso em determinadas plataformas). O nosso Roteador ainda não contempla todas as limitações consideradas pelo sistema comercial. Não representamos esta limitação nesta implementação do nosso Roteador. Deste modo, a comparação não seria representativa porque, devido a essa simplificação, o nosso Roteador tenderia a gerar rotas com menor número de vôos. Entretanto, este tipo de limitação pode ser incorporada, sem nenhum tipo de problema, ao nosso Roteador.

Além disso, analisando os resultados, podemos observar que nos problemas de teste de números 6 e 21 (e 45) o nosso Roteador apresentou soluções que requerem 4 vôos (3 vôos), enquanto que o sistema mainframe de referência apresentou soluções que requerem apenas 3 vôos (2 vôos). Verificamos, então que estes casos envolvem helicópteros pequenos do tipo twin (b-222). Concluimos que apesar de ser útil na maioria dos casos de teste, o esquema empregado em nosso Roteador para rotear grupos de helipontos usando diversos níveis de abstração torna-se menos eficaz - quanto à minimização do custo - quando a frota disponível conta apenas com helicópteros com pequena capacidade como o twin (tem capacidade para apenas 5 assentos) porque o nosso Roteador prioriza o tempo de atendimento e a flexibilidade para se modificar o plano corrente. Por exemplo, no problema de número 6, o Roteador obteve uma solução de custo US\$ 1,990.70 contra uma solução mainframe de US\$ 1,569.88 de custo, percorrendo 1.243,705 km de distância total contra 980,818 km percorridos na solução comercial de referência.

5.6.4 Os Tempos de Atendimento das Rotas Geradas

Entretanto, até mesmo esta suposta falha do nosso Roteador pode ser justificada pois o objetivo do sistema não é o de obter o mínimo custo mas o de atender a demanda no menor tempo de atendimento, considerando a disponibilidade de helicópteros. É claro, que por utilizar um número maior de vôos, nos problemas de teste de números 6 e 21 o nosso Roteador logrou obter um tempo de atendimento menor. As Tabelas 6.a e 6.b, a seguir, mostram, respectivamente, o tempo médio de atendimento das soluções (para problema de número 6) obtidas usando o nosso Roteador e o sistema comercial de referência.

Tabela 6.a - O Tempo de atendimento para problema teste n. 6

ROTEADOR	I D A		V O L T A	
	passag.	tempo atendim.	passag.	tempo atendim.
pch1	2	00:38:18	1	0:41:39
pna1	2	00:39:58	3	0:39:58
ppg1	2	00:42:48	1	0:46:54
pvm2	1	00:45:31	3	0:44:11
reel	3	00:44:29	2	0:44:29
ss20	4	00:50:33	3	0:50:33
Média	14	00:44:31	13	0:44:44

Tabela 6.b - O Tempo de atendimento para problema teste n. 6

Mainframe	I D A		V O L T A	
	passag.	tempo atendim.	passag.	tempo atendim.
pch1	2	00:38:18	1	00:54:08
pna1	2	00:50:11	3	00:39:58
ppg1	2	00:45:32	1	00:46:54
pvm2	1	00:48:15	3	00:44:11
reel	3	00:44:29	2	00:45:40
ss20	4	00:50:33	3	00:50:33
Média	14	00:46:34	13	00:45:53

Podemos verificar, que a média aritmética do tempo de atendimento na ida, considerando todos passageiros, para o nosso Roteador é de 00:44:31 e de 00:46:34 para o sistema comercial de referência. Portanto, obtivemos uma redução de 2 minutos na duração de tempo de ida. A

média aritmética do tempo de atendimento na volta para o nosso Roteador é de 00:44:44 e de 00:45:53 para o sistema de referência. Portanto, obtivemos uma redução de 1 minuto no tempo de duração de volta. O tempo de atendimento é um indicador do nível de serviço do ponto de vista do passageiro. Desta forma, o tempo de atendimento de ida considerado é como o tempo de duração do vôo a partir da base até o heliponto de destino do passageiro atendido e o tempo de atendimento de volta é considerado como o tempo de duração do vôo a partir do heliponto de embarque do passageiro até a base.

O tempo de atendimento vale tanto para passageiro assim como para toda a carga transportada de / para as plataformas off-shore. A limitação de transporte que deve ser considerada é, na verdade, a capacidade de carga do aparelho. O sistema considera como um “passageiro” uma pessoa ou uma carga com o peso padrão de oitenta (80) quilos. Por este motivo, um aparelho pode ter a sua capacidade esgotada mesmo tendo alguns dos seus assentos não ocupados. A vantagem desta possibilidade de conversão é que, desta forma, o sistema Roteador pode tratar igualmente tanto as “perturbações” ao plano corrente com respeito a passageiros assim como com respeito às cargas.

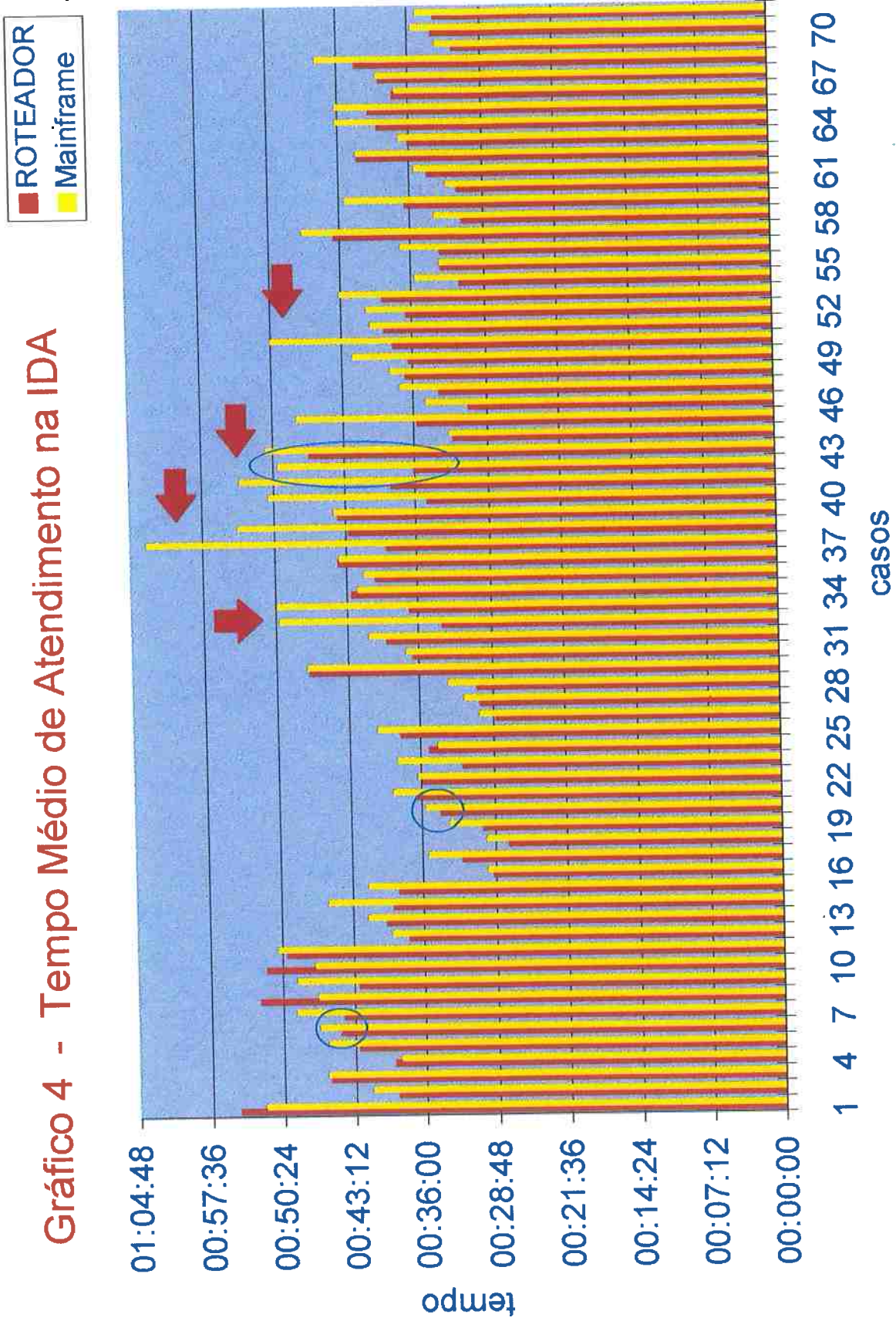
Tabela 7 - Tempo de Atendimento dos Passageiros de Ida e de Volta

ANÁLISE DOS RESULTADOS NOSSO ROTEADOR X MAINFRAME								
	número rot		tempo médio atend. IDA		tempo med. atend. VOLTA			
RECURSOS	ROT.	Mainfr.	ROTEADOR	Mainframe	ROTEADOR	Mainframe		
b-212	2	2	00:54:40	00:52:08	00:55:12	00:59:44	1	
b-222	3	3	00:38:45	00:41:17	00:39:16	00:38:34	2	
b-412	2	2	00:45:33	00:45:45	00:46:00	00:49:59	3	
dauphin	2	2	00:39:02	00:38:24	00:39:25	00:40:34	4	
s-76	2	2	00:42:42	00:45:40	00:43:07	00:43:54	5	
twin	4	3	00:44:31	00:46:34	00:44:44	00:45:53	6	
b212+b222	2	2	00:44:14	00:48:53	00:51:18	00:57:48	7	
b212+b412	2	2	00:52:34	00:46:46	00:52:52	00:50:32	8	
b212+dph	2	2	00:42:37	00:48:49	00:43:24	00:51:03	9	
b-212+s76	2	2	00:51:55	00:47:01	00:52:09	00:44:38	10	
b212+twin	2	2	00:49:51	00:50:41	00:55:04	00:58:53	11	
dph+b222	2	2	00:37:24	00:39:09	00:41:34	00:41:26	12	
dph+twin	2	2	00:39:44	00:41:29	00:42:40	00:44:10	13	
s76+b222	2	2	00:39:01	00:45:32	00:43:51	00:43:50	15	
b-212	2	2	00:38:24	00:41:22	00:40:31	00:39:10	16	
b-222	3	3	00:28:55	00:29:29	00:29:56	00:31:20	17	
b-412	2	2	00:32:00	00:35:20	00:33:46	00:33:22	18	
dauphin	2	2	00:27:26	00:29:33	00:28:56	00:27:59	19	
s-76	2	2	00:30:00	00:33:08	00:31:39	00:31:16	20	
twin	4	3	00:34:06	00:35:31	00:34:05	00:35:29	21	

Tabela 7 - continua

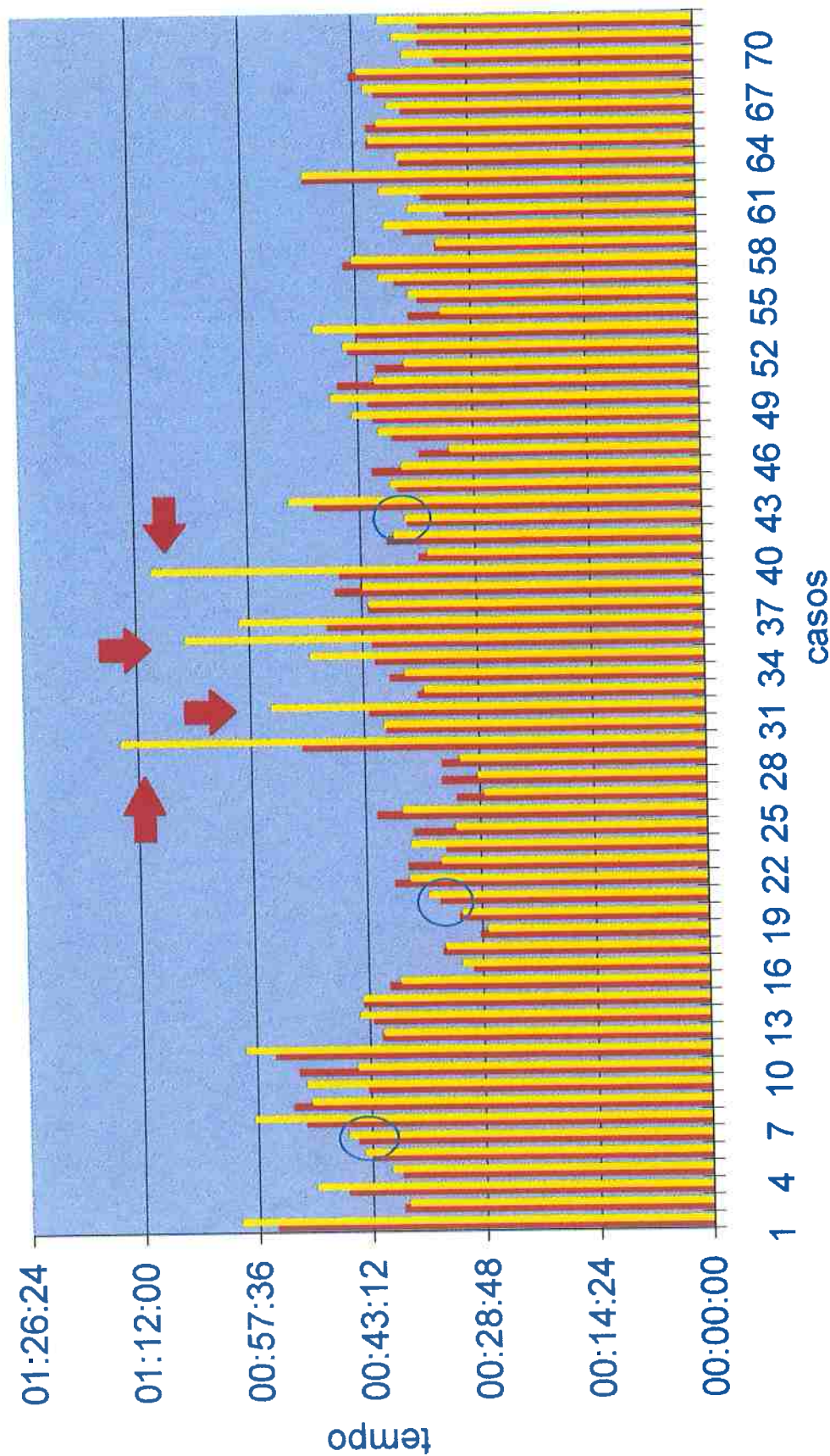
RECURSOS	ROT.	Mainfr.	ROTEADOR	Mainframe	ROTEADOR	Mainframe	
b212+b222	2	2	00:36:27	00:38:44	00:39:42	00:37:50	22
b212+b412	2	2	00:35:51	00:36:10	00:38:02	00:33:48	23
b212+dph	2	2	00:31:49	00:38:15	00:33:12	00:37:35	24
b-212+s76	2	2	00:35:03	00:34:13	00:37:15	00:31:52	25
b212+twin	2	2	00:38:01	00:40:12	00:41:39	00:38:35	26
dph+b222	2	2	00:28:28	00:30:01	00:31:40	00:28:13	27
dph+twin	2	2	00:30:02	00:31:30	00:33:36	00:28:58	28
s76+b222	2	2	00:30:20	00:33:00	00:33:33	00:31:13	30
b-212	2	2	00:47:05	00:47:13	00:50:58	01:13:59	31
b-222	3(0)	3	00:36:34	00:37:05	00:40:21	00:40:37	32
b-412	2	2	00:39:14	00:40:55	00:42:28	00:54:54	33
dauphin	2	2	00:33:38	00:49:55	00:36:24	00:35:35	34
s-76	2	2	00:36:47	00:50:09	00:39:49	00:38:03	35
twin	5	5	00:42:41	00:42:03	00:41:35	00:49:40	36
b212+b222	3	3	00:40:13	00:41:17	00:41:52	01:05:37	37
b212+b412	2	2	00:43:56	00:43:49	00:47:31	00:58:40	38
b212+dph	2	2	00:39:01	01:03:06	00:42:20	00:42:13	39
b-212+s76	2	2	00:42:57	00:53:58	00:46:26	00:43:00	40
b212+twin	3(0)	3	00:43:54	00:44:13	00:45:50	01:09:45	41
dph+b222	3	3	00:34:49	00:50:47	00:35:56	00:34:51	42
dph+twin	3	3	00:38:30	00:53:43	00:39:54	00:38:59	43
s76+b222	3	2	00:36:05	00:49:44	00:37:19	00:37:31	45
b-212	2	2	00:46:47	00:51:03	00:48:50	00:52:08	46
b-222	3(0)	3	00:32:02	00:32:25	00:38:29	00:39:14	47
b-412	2	2	00:35:37	00:47:47	00:41:29	00:37:54	48
dauphin	2	2	00:30:32	00:34:37	00:35:34	00:31:47	49
s-76	2	2	00:33:24	00:37:16	00:38:54	00:40:40	50
twin	5	5	00:36:46	00:38:14	00:41:14	00:43:46	51
b212+b222	3	3	00:36:18	00:42:03	00:41:47	00:46:24	52
b212+b412	2	2	00:37:59	00:50:17	00:45:32	00:41:00	53
b212+dph	2	2	00:38:49	00:40:10	00:40:48	00:37:16	54
b-212+s76	2	2	00:36:30	00:40:33	00:44:12	00:44:44	55
b212+twin	4	4	00:38:58	00:43:12	00:43:11	00:48:28	56
dph+b222	2(0)	2	00:31:09	00:35:27	00:36:37	00:32:37	57
dph+twin	3	3	00:33:01	00:33:05	00:35:30	00:36:33	58
s76+b222	3	3	00:33:03	00:36:55	00:38:19	00:40:13	60
b-212	2	2	00:43:45	00:46:51	00:44:33	00:43:34	61
b-222	3	3	00:30:44	00:33:32	00:33:12	00:33:04	62
b-412	2	2	00:36:27	00:42:31	00:37:08	00:39:24	63
dauphin	2	2	00:31:15	00:32:20	00:31:49	00:36:31	64
s-76	2	2	00:34:11	00:35:22	00:34:48	00:39:56	65
s-61	1	1	00:41:16	00:41:16	00:49:37	00:49:37	66
twin	4	4	00:36:03	00:36:56	00:37:29	00:37:40	67
b212+b222	2	2	00:39:13	00:43:20	00:41:24	00:41:11	68
b212+b412	2	2	00:40:04	00:43:21	00:41:26	00:40:10	69
b212+dph	2	2	00:37:33	00:37:21	00:37:11	00:38:47	70
b-212+s76	2	2	00:38:55	00:39:12	00:40:27	00:41:41	71
b212+twin	2(0)	2	00:41:21	00:45:18	00:43:28	00:42:31	72
dph+b222	2	2	00:31:26	00:33:06	00:32:52	00:36:52	73
dph+twin	2	2	00:33:34	00:35:28	00:34:55	00:37:56	74
s76+b222	2(0)	2	00:33:15	00:34:57	00:34:52	00:39:45	75

Gráfico 4 - Tempo Médio de Atendimento na IDA



■ ROTEADOR
■ Mainframe

Gráfico 5 - Tempo Médio de Atendimento na VOLTA



Apresentamos a Tabela 7, na páginas 169-170, para ilustrar a média aritmética do tempo de atendimento para todos passageiros de Ida e para todos passageiros de Volta obtidos para o nosso Roteador e para o sistema de referência para os números de problemas da Tabela 4, exceto para os de números 14, 29, 44 e 60.

De um modo geral, a Tabela 7 apresenta valores mais favoráveis ao nosso Roteador para os indicadores de tempo de atendimento de passageiros de ida e de volta em relação ao sistema mainframe de referência. Isto pode ser melhor observado nos Gráficos 4 e 5. O Gráfico 4, na página 171, ilustra o tempo de atendimento de passageiros na ida (hh:mm:ss). Do mesmo modo, o Gráfico 5, na página 172, mostra o tempo de atendimento de passageiros na volta (hh:mm:ss).

No caso 6 do teste, obtivemos uma redução de 2 minutos no tempo de atendimento de ida. No caso 21, obtivemos uma redução de 1 minuto no tempo neste tempo. Usando o Gráfico 4 podemos visualizar os casos em que a Tabela 7 indica que houve uma grande redução neste tempo de atendimento nas viagens de ida: caso 39 (menos 24 minutos), caso 34 (menos 16 minutos), caso 42 (menos 15 minutos), caso 43 (menos 15 minutos).

No caso de teste 6 obtivemos uma redução de 1 minuto no tempo de atendimento de volta. No caso 21, obtivemos também uma redução de 1 minuto neste tempo. Através do Gráfico 5 podemos visualizar os casos em que, de acordo com a Tabela 7, houve uma grande redução neste tempo de atendimento nas viagens de volta: caso 41 (menos 24 minutos), caso 31 (menos 23 minutos), caso 37 (menos 23 minutos), caso 38 (menos 12 minutos), caso 33 (menos 12 minutos), etc.

Calculando-se a média aritmética do indicador de tempo de atendimento, a partir da Tabela 7, obtém-se a média do tempo de atendimento na ida de 00:44:01 para o nosso Roteador contra 0:48:08 para o sistema de referência e a média do tempo de atendimento na volta de 00:47:05 para o nosso Roteador contra 00:49:03 para o sistema de referência. Deste modo, temos um bom indicador mostrando que o nível de qualidade do serviço melhorou em média, praticamente, de quatro minutos nas viagens de ida e de dois minutos nas viagens de volta.

5.6.5 - O Comportamento Semi-Reativo

O Roteador de Helicópteros apresenta algumas das funcionalidades que caracterizam o comportamento semi-reactivo suportado pela metodologia de desenvolvimento proposta no Capítulo 4. O sistema de transporte por helicópteros pode ser considerado como um ambiente dinâmico, porque, estão ocorrendo continuamente mudanças no sistema em consequência de eventos que não foram causados ou controlados pelo planejador. No sistema dinâmico, a ocorrência destas mudanças é a regra geral; a exceção é não ocorrer nenhuma mudança. Portanto, a reatividade -- i.e., a capacidade de modificar continuamente um plano corrente, mantendo-o factível todo o tempo -- é uma característica desejada para o Roteador de Helicópteros. A arquitetura do Roteador de Helicópteros segue a proposta ilustrada pela Figura 37, no Capítulo 4. O sistema é semi-reactivo porque trata apenas dos casos em que ocorre uma inclusão de novas requisições de viagens às rotas existentes no plano corrente. O cancelamento de requisições de viagens previstas anteriormente é tratado simplesmente eliminando-se a sua execução do plano corrente. O sistema considera apenas casos onde as modificações necessárias no plano original possam ser consideradas como *perturbação*^{*1}, i.e., não causam mudanças muito drásticas no plano.

Estas mudanças ou *perturbações* ao plano corrente podem ser: (i) não disponibilidade do tipo de aparelho alocado pelo Roteador; (ii) impossibilidade de pouso em determinado heliponto previsto na rota, (iii) cancelamento de uma requisição de viagem prevista anteriormente; (iv) inclusão de nova requisição ao plano corrente em andamento, etc.

Considere o exemplo do problema descrito como de número 1 na Tabela 2 da página 156, e referenciado nos itens 5.4.2 e 5.5.1. Suponha que não exista helicóptero disponível para efetuar a viagem prevista pelo Roteador para a rota 1 e que tem-se o uso de um aparelho do tipo bell-222 uma como alternativa possível.

*¹ O modelo conceitual feito em rede de Petri efetua um fechamento sobre o domínio do problema. Estas "perturbações" podem causar uma modificação da estrutura da rede de Petri que representa o domain-K, eliminando ou incluindo novos objetos. A perturbação deve ter uma extensão limitada para não tornar o modelo do domínio não aderente à realidade.

O sistema informa se esta troca de aparelhos é factível ou não. Para isto, utiliza-se a função *Inicialização* do Menu do sistema na opção *Helicópteros*, tornando o número de aparelhos b-222 disponíveis igual a um. Em seguida, utiliza-se a função *Modifica-Plano-Corrente* do Menu do sistema e a opção troca de aparelho. Selecione a rota 1 como sendo a rota a ser modificada. Digite b-222 como o novo tipo de aparelho para a rota 1. O sistema irá emitir uma mensagem informando que este tipo de troca é factível. e pede uma confirmação para efetuar a substituição em definitivo. Sendo confirmado, a troca é feita e são recalculados os tempos de atendimento dos passageiros, a duração total do vôo e o custo total do vôo. A rota 1, assim modificada pode ser vista, utilizando-se a função *Exibir-Plano-Corrente* do Menu do sistema.

rota modificada: 1 custo (U\$): 483.95
 helicóptero: b-222 capac.: 7 custo (U\$/h): 428.00 veloc.(km/h): 244.5

08:00:00	mcae-pch1	132.43	00:32:30
08:32:30	pch1-pna1	5.78	00:01:25
08:33:55	pna1-mcae	138.21	00:33:55
		276.419	01:07:50
passageiros:	mcae - pch1 - pna1 - mcae		
ida:	- 2 - 2 -		
volta:	- 1 - 3 -		
trecho:	4 - 3 - 4		

Em relação à solução nas páginas 148-149, a duração do vôo para a rota 1 diminui de 1:29:33 para 1:07:50 e o custo total da solução do problema diminui de U\$ 1,287.57 para U\$ 1,213.31 - observe que a distância percorrida de 276,419 km permanece a mesma.

Suponha agora uma segunda perturbação, desta vez que ocorreu um cancelamento de uma requisição de viagem para pvm2 (demanda de ida = 1 e volta =3), prevista anteriormente pelo Roteador para a rota 2. Utilize novamente a função *Modifica-Plano-Corrente* do Menu do sistema e a opção *Eliminar-heliponto-na-rota*. Selecione agora a rota 2 como sendo a rota a ser modificada. Digite pvm2 como sendo o heliponto a ser eliminado da rota e confirme esta modificação. Sendo confirmada, a troca é feita e são recalculados os tempos de atendimento dos passageiros, a duração total do vôo e o custo total do vôo. O pla-

no corrente, após estas duas modificações, pode ser visto utilizando-se a função *Exibir-Plano-Corrente* do Menu do sistema Para este número de problema de teste, as rotas modificadas são:

rota 1: ... o mesmo mostrado na página anterior

rota modificada: 2 custo (U\$): 709.46
 helicóptero: b-212 capac.:11 custo (U\$/h): 374.00 veloc.(km/h): 185.2

horário	trecho	dista(km):	tempo voo:
08:00:00	mcae-ppg1	147.99	00:47:56
08:47:56	ppg1-ss20	28.57	00:09:15
08:57:12	ss20-reel	20.94	00:06:47
09:03:59	reel-mcae	153.82	00:49:49
		351.317	01:53:49

passageiros:	mcae	-	ppg1	-	reel	-	ss20	-	mcae
ida:	-		2	-	4	-	3	-	
volta:	-		1	-	3	-	2	-	
trecho:	9	-	8	-	7	-	6	-	

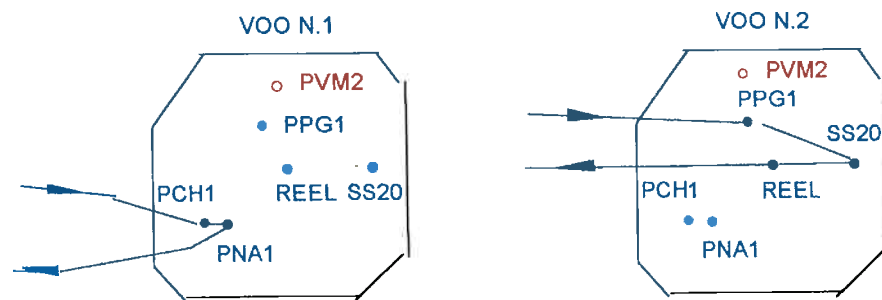


Figura 51 - O Plano modificado para eliminar heliponto na rota 2

A Figura 51, acima, mostra o Plano modificado para tratar a eliminação de heliponto na rota 2. Neste caso, a duração total do voo para a rota 2 diminuiu de 1:57:00 para 1:53:49, a distância percorrida diminuiu de 361,169 km para 351,317 km e o custo diminuiu de U\$ 729.36 para U\$ 709.46 e o custo total da solução para o problema diminuiu de U\$ 1,213.31 para U\$ 1,193.42.

A implementação da perturbação que corresponde à inclusão de uma nova requisição de viagem ao plano corrente ainda não foi implementada no nosso Roteador. Entretanto, esta funcionalidade é suportada pela metodologia e pode ser implementada sem problemas. Como comentário, ressaltamos que a perturbação em questão exige que se faça a modifica-

ção do modelo que representa o domínio, i.e., em todos os níveis de abstração da Rede Ghenesys. Isto corresponde a rever todos os passos do processo de planejamento, levando em conta agora, o modelo modificado que representa o domínio e o plano existente com o objetivo de preservá-lo o mais possível.

Para implementar os tipos de modificação já disponíveis no nosso Roteador, utilizamos o nível mais baixo (Nível 0) de abstração. O tratamento para os níveis de abstração mais altos da rede Ghenesys, e os planos correspondentes foram eliminados, simplesmente, porque deixaram de ter a aderência com o problema a partir da modificação feita.

O tratamento da inclusão de novas tarefas requer a modificação da topologia da rede Ghenesys que representa o domain-K.

Uma alternativa para tornar o modelo conceitual que representa o domain-K menos susceptível à modificações freqüentes está no uso das Redes de Petri Auto-Modificáveis (RdPAM^{*2}) [Valk 1978], [Sasaki 1996]. Imagina-se que, incorporando a funcionalidade de apresentar arcos mutantes^{*3} das RdPAM à rede estendida Ghenesys, possamos obter uma modelagem capaz de tratar adequadamente (sem necessidade de modificar a rede) o fato de, eventualmente, um helicóptero já programado seja colocado em manutenção de emergência ou que uma determinada plataforma não apresente condições de pouso e decolagem.

É claro que, no caso do Roteador de Helicópteros, o fato de o modelo conceitual do domain-K ser preservado não implica que o replanejamento não seja necessário. Pelo contrário, é preciso replanejar porque o Roteador é um sistema semi-reativo e, por este motivo, apresenta menos automatismo do que um sistema reativo puro. No sistema Roteador, o *mc* representa o domínio e não o plano. Acreditamos que não é o caso de se elaborar planos de

*² A RdPAM é uma variante da Rede de Petri Ordinária em que os pesos dos arcos são indicados através de variáveis que estão relacionados com a quantidade de marcas em lugares pré-definidos. Desta forma, existe uma dependência da evolução do sistema em função da marcação atual da rede. A RdPAM é utilizada, principalmente, para modelar processos de recuperação de erros no controle de sistemas flexíveis de manufatura, em caso de quebra de equipamentos.

*³ Através de um artifício de modelagem, por exemplo, usando portas-habilitadoras e porta-inibidoras -- para não eliminar a possibilidade de se reduzir a rede estendida Ghenesys a uma rede Elementar no nível de abstração "ground".

contingência, que sofre restrições de muitos autores porque a sua relação de custo-benefício é bastante questionável. Reconhecemos, portanto, o benefício da incorporação das funcionalidades das RdPAM no modelo conceitual seria maior para um sistema de “planning” e execução do plano em tempo-real.

5.6.5 - O Potencial de Ganho Econômico

Reportando, ainda, à Tabela 4, podemos obter os valores médios para os problemas de testes 1 a 75 exceto os de números 14, 29, 44 e 60. Os valores médios de custo total para estes problemas é de U\$ 1,318.71 para o nosso Roteador e de U\$ 1,366.46 para o sistema de referência. Os valores médios das distâncias das rotas é de 682,176 km para o nosso Roteador e de 701,659 km para o sistema de referência. Deste modo, o nosso Roteador elaborou em média soluções que percorrem 19,5 km menos do que as soluções elaboradas pelo sistema de referência. A média aritmética do custo unitário, considerando todos os passageiros, para o nosso Roteador é de U\$ 44.27 / passageiro e de U\$ 45.87 / passageiro para o sistema de referência. Este resultado indica que, além de melhorar o tempo de atendimento, podemos obter uma redução nos custos de transporte de U\$ 1.60 / passageiro. Para a demanda estimada de até 20.000 passageiros por mês, existe a possibilidade de uma redução de custo de até U\$ 384,000.00 por ano.

Estes resultados nos permitem afirmar que podemos obter soluções próximas e, eventualmente, até melhores do que as soluções do sistema comercial de referência utilizando a abordagem que combina Rede de Petri e IA. Isto apesar de o nosso Roteador não usar otimização matemática -- usa apenas IA associado com a análise da estrutura da Rede de Petri para dirigir o processo de busca da solução.

Além disso, outros resultados mostram uma melhoria média no tempo de atendimento de 4 minutos nas viagens de ida para as plataformas e de 2 minutos nas viagens de retorno para a base, melhorando assim, o nível do serviço de transporte por helicópteros.

Finalmente, mas não menos importante, destacamos a flexibilidade alcançada com o uso de IA para tratar “perturbações” no plano corrente. O nosso Roteador apresenta vantagens porque pode ser estendido para suportar implementação em ambiente “on-line”, apresentando comportamento semi-reativo como descrevemos no capítulo 6. Conseqüentemente, o nosso Roteador de Helicópteros -- desenvolvido segundo metodologia proposta no capítulo 4 -- pode ser caracterizado como um sistema mais flexível que apresenta um planejador que elabora soluções com mais qualidade.

CAPÍTULO 6

6

CONCLUSÃO E TRABALHOS FUTUROS

A literatura mostra que a elaboração de planos lineares simples construídos usando tão somente abordagens ingênuas baseado nos métodos fracos (busca em largura, busca em profundidade), no estilo tentativa e erro, não logram sucesso. A resolução destes problemas constitui um desafio permanente devido a sua complexidade -- os algoritmos para sua solução conhecidos são NP-completos. O que se busca é obter sistemas de planejamento mais eficazes (que consigam, efetivamente, atingir uma boa solução para o problema) e mais eficientes (que esta boa solução seja obtida no tempo de resposta exigido pelo problema). Para se obter um melhor desempenho computacional nestes sistemas, é necessário se adotar mecanismos auxiliares para estruturar o problema.

Além disso, queremos desenvolver sistema de “planning” mais flexíveis, capazes de tratar adequadamente as modificações ao plano corrente sem refazer todo o processo de planejamento. Com este objetivo propusemos, neste trabalho, a estruturação do problema de planejamento baseada em IA e no formalismo das redes de Petri [Silva et Shimad 1995]. A estruturação proposta permite a construção de um método para desenvolvimento de sistemas de planejamento mais flexíveis, com qualidade. Esta proposta apresenta uma alternativa à representação STRIPS para expressar o modelo de ação. Vimos no capítulo 2, que a escolha deste tipo de representação causa inúmeros problemas.

6.1 O Uso de Modelos Conceituais

A nossa proposta está baseada em IA e no formalismo das redes de Petri, como descrito em [Rillo 1988]. As vantagens desta abordagem são:

- (i) separação do conhecimento do domínio do meta-conhecimento de “planning”,
- (ii) representação gráfica de todos os estados do domínio do problema,
- (iii) representação algébrica equivalente na forma de equação de estado,
- (iv) análise de propriedades da rede de Petri,

- (v) decomposição estrutural do problema,
- (vi) aprendizado de estratégias de busca de soluções,
- (vii) representação do meta-conhecimento em rede de Petri,
- (viii) Ambiente de Redes de Petri para Construtor de Planos,
- (ix) módulo supervisor da execução do plano em rede de Petri,
- (x) replanejamento na falha,
- (xi) capacidade semi-reativa,
- (xii) metodologia aplicável a qualquer domínio da aplicação.

Separação do conhecimento do domínio do meta-conhecimento de “planning”:

A estruturação proposta faz uma separação clara dos tipos de conhecimento existentes num problema de “planning”: o conhecimento do domínio da aplicação e o meta-conhecimento para dirigir o processo de busca de soluções.

Representação gráfica de todos os estados do domínio do problema:

A idéia da representação gráfica do problema tem aqui o valor equivalente ao da (idéia) “máxima” associada aos diagramas utilizados nos métodos de desenvolvimento de sistemas”, por exemplo, DFD, MER, etc, ou seja, “um gráfico vale por mil palavras. A rede de Petri promove a modelagem como Sistema de Eventos Discretos (SED) pode capturar a característica principal de um sistema produtivo que é o seu comportamento dinâmico [Miyagi 1990]. O modelo conceitual em RdP efetua um “fechamento” sobre o domínio da aplicação. A Rede Ghensys suporta a representação de diferentes níveis de abstração. Esta característica permite se fazer a estruturação do domain-K de uma forma hierárquica em que cada nível de abstração apresenta um grau de detalhamento diferenciado.

Representação algébrica equivalente na forma de equação de estado:

A rede de Petri é uma entidade matemática com representação através de equações algébricas. Neste trabalho utilizamos o ambiente HIPER, que consiste de um ambiente para edição, análise e simulação da rede estendida Orientada a Objetos Ghensys. Os módulos de análise e simulação da rede Ghensys são inteiramente baseados na equação de estado. (Veja o cap. 3, item 3.5.4).

Análise de propriedades da rede de Petri:

A Rede de Petri é um ente matemático que possui diversas propriedades intrínsecas. Queremos conhecer e explorar estas propriedades da RdP do domain-K, para o benefício do sistema. Desta forma, a nossa proposta de estruturação do problema de “planning” caracteriza como um processo de “solução por metáfora” (Veja o cap. 1, item 1.7).

Decomposição estrutural do problema:

A nossa proposta de estruturação pode efetuar uma abstração estrutural do problema de “planning”, como descrito em [Mädler 1992]. A análise das propriedades estruturais da RdP pode sugerir formas de se fazer a decomposição do problema em subproblemas independentes. Ilustramos isto através do exemplo do Mundo de Blocos.

Aprendizado de estratégias de busca de soluções:

O sistema é capaz de aprender estratégias de planejamento, independente do domínio da aplicação, como descrito em [Day 1992]. Este aprendizado de possíveis estratégias de busca da solução é feita através da análise das propriedades estruturais das redes de Petri. Por exemplo, no exemplo do Mundo de Blocos, o uso de estratégias diferenciadas para tratar dos problemas com o estado inicial e o estado meta na mesma subrede e em subredes distintas. Além disso, arranjos típicos na rede de Petri que denotam de transições concorrentes, por exemplo, no uso de recursos compartilhados, exigem a associação destas transições com regras heurísticas obtidas, em geral, do domínio da aplicação. Este conhecimento pode ser associado às técnicas de busca da usadas na IA e na teoria da seqüenciação aplicadas na engenharia de produção.

Representação do meta-conhecimento em rede de Petri:

O meta-conhecimento (seja o resultado obtido a partir da análise estrutural da rede de Petri, seja o conhecimento da teoria de “planning”, seja uma regra heurística (IA) obtida diretamente do domínio do problema), são representados através da rede de Petri Estendida Orientada a Objetos Ghemesys. Desta forma, temos uma representação homogênea de todos os tipos de conhecimento do sistema em Redes de Petri: o conhecimento do domínio e o meta-conhecimento da estratégia.

O Ambiente baseado em Redes de Petri:

O nível mais baixo do “planning self-model” deve incluir uma representação do Construtor de Planos, que é um ambiente de Redes de Petri que opera sobre a rede que representa o conhecimento do domínio (ou domain-K). O Construtor de Planos promove uma “competição” entre as ações plausíveis para fazer a escolha da próxima ação a ser realizada (que corresponde, na modelagem feita, à uma transição da RdP que representa o domain-K). Para isso, faz o cálculo do valor de um indicador que está associado ao critério corrente para a escolha da ação. Note que o Construtor de Planos baseado em Rede de Petri utiliza apenas a estrutura da Rede de Petri do domain-K. Ele não explora a dinâmica da rede e nem faz a evolução de marcas na rede.

Módulo supervisor da execução do plano em rede de Petri:

Quanto à monitoração da execução do plano, podemos afirmar que este pode ser efetuado, com vantagens, usando-se a representação do plano em redes de Petri, como foi usado em [Paiva 1993], ao invés de se fazer uma descrição dos estados de execução / habilitação de uma tarefa como em [McDermott 1978].

Replanejamento na falha:

Quanto ao replanejamento na falha, a representação do plano na forma de rede de Petri pode ser vista como uma alternativa aos mecanismos de tabela triângulo usada em STRIPS+PLANEX [Fikes, Hart et Nilsson 1972] e ao mecanismo da árvore de submetas e do grafo de decisão, utilizados por [Hayes 1975]. Observe que a nossa proposta tem a vantagem da representação homogênea do conhecimento em Redes de Petri.

Capacidade semi-reativa:

A arquitetura proposta para o sistema está descrita no cap. 4, item 4.9. O Sistema Planejador Semi-Reativo apresenta dois módulos: um módulo Supervisor e Executor do Plano e um módulo Planejador e Revisor do Plano. Ao contrário dos sistemas tradicionais de planejamento baseados em IA, o sistema proposto monitora continuamente o ambiente onde atua o sistema e a execução o sucesso (ou não) das ações programadas no plano corrente. As modificações ocorridas no ambiente pode causar a falha de uma ação durante a sua execução ou, até mesmo, esta nem pode ser iniciada. Estas modificações são

tratadas como “perturbações” às quais o módulo Planejador e Revisor deve reagir, mantendo o plano corrente continuamente factível. O sistema é semi-reactivo porque sempre tem um plano completo corrente (ao invés de um plano parcial corrente) e, além disso, no caso de cancelamento de uma tarefa anteriormente prevista, simplesmente retira esta tarefa do plano corrente, sem retomar o processo de “planning”.

Metodologia aplicável a qualquer domínio da aplicação:

Em nossa abordagem, é a análise da estrutura das redes de Petri que permitem o aprendizado de heurística de busca para dirigir o processo de planejamento. Por este motivo, a metodologia pode ser reaplicada para um outro domínio qualquer de problema. Isto ocorre porque o Construtor de Planos, que é um ambiente de Rede de Petri, opera sobre a representação do problema em redes de Petri, e não sobre a sua interpretação para um determinado domínio da aplicação.

6.2 - A Análise dos Resultados Obtidos

No capítulo 4, desenvolvemos a proposta de estruturação, ilustrando cada passo com o exemplo do Mundo de Blocos. Os resultados foram obtidos a partir da implementação em Visual PROLOG. Utilizando a metodologia proposta, conseguimos obter um sistema eficaz (que sempre tem sucesso em obter um plano) e eficiente (que obtém o plano num tempo de processamento pequeno). Usando os resultados da análise estrutural da rede de Petri que representa o domain-K, fizemos uma decomposição em subproblemas independentes que permitem o uso de uma mesma estratégia de busca da solução para progredir monotonicamente na direção da solução do problema.

Mostramos, no Apêndice II, que resultados adicionais desta abordagem permite a resolução do problema do Mundo de Blocos generalizado para n blocos. Afirmamos que se existe um modelo lógico do domínio, então, também existe um modelo em Rede de Petri. Ao contrário do “application model” para três blocos, que foi obtido por enumeração completa dos estados, o mc do domain-K para o problema generalizado foi obtido num processo de raciocínio por indução.

A representação do modelo de ação em nossa proposta não utiliza mais o mecanismo do tipo adição/eliminação da hipótese STRIPS. Desta forma, evitamos os problemas inerentes à esta representação: (i) o “frame problem”, (ii) o problema da interdependência entre ações. No problema do Mundo de Blocos, podemos evitar a ocorrência da Anomalia de Sussman, sem necessidade de se elaborar antes um plano com falhas para depois corrigi-lo como é feito em [Sussman 1977], [Waldinger 1975], [Lifschitz 1987], [Sacerdoti 1975]. Ao contrário, um plano sem falhas foi elaborado desde o princípio.

Os problemas exemplo combinatórios constituem um bom campo de prova (“test-bed”) para os sistemas de “planning”. A nossa proposta de estruturação constitui um método apropriado para representar os aspectos “salientes” do domínio destes problemas, sendo capaz de capturar todos estados de interesse destes sistemas. No Apêndice II, descrevemos como a proposta pode ser aplicada como sendo um “planning schemata” para a resolução de problemas combinatórios. Ilustramos os resultados obtidos de programas que elaboramos para diversos problemas exemplo deste tipo: o problema das jarras, o problema do fazendeiro e o problema dos missionários e canibais.

Com o avanço da informática e da automatização das fábricas, observa-se um esforço cada vez maior para integrar as redes de automação com as redes administrativas. Este movimento é conhecido como re-engenharia de processos e integração de sistemas corporativos. Nesta perspectiva, tornam-se importantes os sistemas de “planning” e “scheduling” dos Sistemas de Manufatura Flexíveis, levando-se em conta os objetivos que são estabelecidos na rede administrativa (de informática). No Apêndice I, descrevemos uma implementação que fizemos usando a linguagem Visual PROLOG para o problema de “Scheduling Job Shop” na automação da fábrica, com a finalidade de mostrar que a nossa proposta de solução pode também ser aplicada na resolução de problemas neste domínio. O “application model” foi construído usando a rede estendida Ghenesys, com diversos níveis de abstração.

Os resultados descritos acima foram obtidos apenas para estes problemas “toy”. Para mostrar que a nossa proposta de metodologia é válida e aplicável para problemas de

grande porte*¹ do mundo real apresentamos, no capítulo 5, uma aplicação completa de um sistema Roteador de Helicópteros. O sistema Roteador foi concebido para ser flexível, capaz de tratar adequadamente modificações ao plano corrente, sem a necessidade de refazer todo o processo de planejamento. A flexibilidade é obtida associando-se a regra heurística (IA) do máximo espalhamento dos helicópteros e a idéia de “cluster” de plataformas, que são representadas através dos diversos níveis de abstração da rede de Petri estendida Ghenesys.

O Sistema Roteador de Helicópteros, apresenta não só uma hierarquia de planos, mas também uma hierarquia de modelos do domínio da aplicação. O Construtor de Planos é um Ambiente de Rede de Petri que opera sobre a rede que representa o domain-K.

O sistema elabora o plano inicial de uma forma “top-down” e opera para fazer a modificação do plano corrente para tratar as “perturbações” de uma forma “bottom-up”. O planejamento de uma maneira “top-down” permite explorar a estratégia oportunística [Hayes-Roth 1979] que consiste numa “competição” entre os helicópteros para fazer a escolha da próxima ação a ser realizada.

Para se fazer uma comparação dos resultados computacionais obtidos pelo sistema implementado, utilizamos como referência um sistema comercial existente que implementa o algoritmo de Clark & Wright [Clark et Wright 1964].

Efetuamos testes comparativos em 61 casos de teste, cujos resultados estão ilustrados nas tabelas 4, 5 e 7. Os resultados comparativos foram tais que o nível de qualidade do serviço melhorou em média, praticamente, de quatro minutos nas viagens de ida e de dois minutos nas viagens de volta. O nosso roteador elaborou em média soluções que percorrem 19,5 km menos do que as soluções elaboradas pelo sistema de referência. A média aritmética do custo unitário, para o nosso roteador é de U\$ 44.27 / passag. e

* ¹ O conceito de um sistema grande e complexo é muito impreciso porque não existe uma definição formal de um “sistema grande”. Utilizamos aqui um conceito empírico, baseado no Projeto de Redes de Petri, de que um sistema “grande e complexo” é aquele que apresenta mais de 20 condições e mais de 20 eventos e, onde a matriz de incidência correspondente não é esparsa.

de US\$ 45.87 / passag. para o sistema de referência. Este resultado indica uma redução nos custos de transporte de US\$ 1.60 / passageiro.

O destaque fica por conta da confirmação prática da capacidade semi-reativa do Roteador de Helicópteros, descrita no item 5.5.1. Por exemplo, se for necessário trocar o tipo de aparelho, o sistema verifica se o helicóptero alternativa é factível. Caso positivo, a troca é feita e são recalculados os tempos de atendimento dos passageiros, a duração total do vôo e o custo total do vôo. O sistema suporta um segundo tipo de perturbação; o cancelamento de uma requisição de viagem prevista anteriormente. O heliponto é eliminado da rota e são recalculados os tempos de atendimento dos passageiros, a duração total do vôo e o custo total do vôo.

A perturbação que corresponde à inclusão de um novo heliponto numa rota existente ainda não foi implementada. Entretanto, esta funcionalidade é suportada pela metodologia e pode ser implementada sem problemas. Como comentário, ressaltamos que, em nossa proposta, a perturbação em questão exige que se faça a modificação do modelo que representa o domínio, i.e., da Rede estendida Ghenesys.

6.3 Outras Perspectivas de Fusão IA / Redes de Petri:

A revisão do modelo conceitual para refletir a ocorrência de uma “perturbação” está relacionada com o natureza dinâmica do problema de “planning”. [Silva 1992] mostra que a axiomatização de sistemas dinâmicos requer o uso de lógicas mais complexas, como é a lógica modal. A Lógica Dinâmica [Fischer 1979], [Harel 1979] é uma forma de lógica complementar derivada da lógica modal alética, criada para representar a lógica de programas através de uma sucessão de estados representados por proposições lógicas. A lógica modal alética [Costa 1992], [Hughes et Creswell 1968], cujos operadores modais são: necessidade (\square) e possibilidade (\diamond), abordam um sistema dinâmico através do conceito de mundos possíveis.

Por mundo possível entende-se todos os mundos acessíveis a partir do mundo real. No mundo acessível a partir do mundo real existem objetos com suas propriedades e relações entre estes. No mundo acessível na Semântica dos Mundos Possíveis de Kripke [Hintikka 1962], [Costa 1992], [Hughes et Creswell 1968] não só estas propriedades e relações podem se alterar como também pode-se remover objetos existentes no mundo real (adicionando-se ou não objetos que não existem no mundo real).

No aspecto prático, o paradigma dos mundos possíveis corresponde a um sistema dinâmico por eventos discretos habitado por múltiplos agentes. Num sistema por eventos discretos a transição entre os estados do mundo se faz através da ocorrência de eventos discretos. Alguns destes eventos acontecem sem o controle do agente. Os modelos de mundo (cada agente = um modelo de mundo) e são os mundos acessíveis a partir de um mundo real (do modelo de mundo de um agente em particular) .

Questão: Dados os mundos possíveis (correspondendo aos modelos de mundo dos outros agentes), como estabelecer o valor verdade num determinado mundo acessível (modelo de mundo de outro agente) de uma proposição acerca de um objeto (p. ex. uma "feature" de outro agente) que não existe no domínio do mundo real (no modelo de mundo do agente) ?

É claro que o modelo de mundos acessíveis totalmente irrestrito, é mais próxima do mundo dinâmico que queremos modelar. Entretanto, neste modelo, o valor verdade de algumas proposições é meio imprevisível. Não é possível se decidir o valor verdade de uma proposição num determinado mundo acessível, a menos que se restrinja o escopo dos objetos que podem ser removidos ou adicionados. Isto significa que, só de consegue estabelecer o valor verdade num determinado mundo (acessível) de uma proposição acerca de um objeto (p. ex. uma "feature" de um determinado agente) que não existe no domínio do mundo (real) se o escopo destes objetos for conhecido (limitado) de antemão.

Portanto, mesmo os sistemas lógicos mais complexos, exigem que a extensão de uma "perturbação" seja limitada para que este seja capaz de tratá-la adequadamente.

Como uma alternativa para não se utilizar a lógica modal na representação de sistemas dinâmicos, [Doyle 1979] apresenta a abordagem dos sistemas de manutenção da verdade, que vamos discutir em seguida.

6.3.1 - O Problema da Manutenção da Verdade

Os sistemas baseados em IA, usualmente, constroem modelos computacionais da aplicação. Para manter estes modelos consistentes em sistemas dinâmicos -- quando ocorre a chegada de uma nova informação e muda o estado do domínio da aplicação que foi representado -- o sistema tem de eliminar ou modificar parte destes modelos. Uma modificação, freqüentemente, pode causar outras mudanças porque o sistema freqüentemente constrói uma parte do modelo em função de outras partes do modelo.

Recentemente, tem havido um interesse crescente em IA (para aplicações de CAD Inteligente) no estudo do Sistema de Manutenção da Verdade (TMS - "True Maintenance System") [Doyle 1979]. O TMS é um sistema que efetua funções para gravar e manter *razões para crença* do programa. Tais razões para crença são úteis para se construir explicações do porque das decisões do programa e também para dirigir as ações do mecanismo de inferência (MI) na elaboração de uma solução. Para escolher suas ações, o MI deve ser capaz de suportar hipóteses e, posteriormente, revisá-las quando surgir uma contradição negando estas hipóteses.

[Doyle 1979] apresenta a idéia do agente racional que deve apresentar a noção de crença, de desejo e de intenção. No TMS, a única componente real usada para se fazer o raciocínio é o conjunto corrente de razões para crença. *Razão para crença* é definida como "crença verdadeira justificada" ou "argumentação racional", em oposição ao simples estabelecimento do valor verdade de uma sentença. Neste sentido, uma sentença P não é mais questionada acerca do seu valor verdade mas, sim quanto a constituir uma razão aceitável (*válida*) correntemente. Assim, ou uma sentença P :

- tem pelo menos uma razão presentemente aceitável (*válida*) e, deste modo, pertence ao conjunto corrente de crenças, ou

- não tem nenhuma razão presentemente aceitável (não tem razão alguma, ou apenas razões não aceitáveis) e, deste modo, não pertence ao conjunto corrente de crenças.

Se uma sentença P pertence ao conjunto de crenças corrente, dizemos que P está *in*. Se P não pertence ao conjunto de crenças corrente, dizemos que P está *out*. Uma razão para crença (ou justificativa) é formada por um par ordenado de outras crenças, tal que a crença deduzida está *in* por força apenas desta razão se cada crença no primeiro conjunto do par ordenado está *in* e cada crença no segundo conjunto deste par ordenado está *out*. Uma hipótese H é uma crença que independe de outras crenças.

Ex. H ({}, {}).

O TMS guarda dois tipos de dados para fazer a manutenção da verdade: os nós (crenças) e as justificativas (razões para crença). [Doyle 1979] apresenta apenas dois procedimentos para isto: o SL-justification (justificativa lista-de-suporte) e CP-justificatiosn (justificativa prova-condicional). A justificativa SL é a mais simples. Uma justificativa SL para um nó é formado por um par ordenado de listas. Um nó tem a sua razão para crença justificada se cada nó no primeira lista está *in* e cada nó na segunda lista está *out*. Exemplo:

Nó	Comando	Justificativa	Comentário
N-1	Operação(Op) = Op1	(SL () (N-2))	uma hipótese
N-2	Operação(Op) = Op2		ainda não justificado
N-3	Tempo(Op) = 6 hr	(SL (N-37 N-1) ())	
N-4	Tempo(Op) = 3 hr	(SL (N-41 N-2) ())	

No exemplo acima, tem-se que N-2 não está *in* porque não está justificado ainda. O Nó N-1 está *in* porque a primeira lista está vazia e N-2 que está na segunda lista está *out*. Daí, N-3 está *in* porque a N-1 na primeira lista já está *in* e supondo que N-37 está *in*, já que a segunda lista está vazia. Assim, temos N-1 e N-3 no conjunto de crenças correntes que representam que, por exemplo, é crença corrente de que Op1 foi iniciada com tempo de 6 hr.

Agora, suponha que por uma mudança não monotônica, temos uma nova informação, de que Op2 foi iniciada (representado pelos nós N-7 e N-8) e devemos rejeitar a

hipótese N-1 e N3 (*out*), enquanto que N2 e N-4 passam a ser *in*, supondo que N-412 está *in*.

<i>Nó</i>	<i>Comando</i>	<i>Justificativa</i>	<i>Comentário</i>
N-2	Operação(M) = Op2	(SL (N-7 N-8) ())	justificado

Agora, no conjunto de crenças correntes temos N-2 e N-4 que representam que, por exemplo, é crença corrente de que Op2 foi iniciada com tempo de 3 hr.

[Doyle 1979] faz uma crítica da monotonicidade implícita nos sistemas tradicionais. Nestes, a crença é igual à verdade, e a verdade nunca muda. Nesta visão simplista, o processo de inferência deve sempre derivar novos conhecimentos a partir dos conhecimentos anteriormente já existentes. Desta forma, a única ação possível que resta ao MI é a de aumentar o conjunto corrente de crenças com mais crença. Isto é a causa de três problemas conhecidos que estão interligados: (a incapacidade de representar) o senso comum, o “frame problem” e (a dificuldade de mudar rapidamente) o controle.

Por exemplo, freqüentemente, elaboramos uma solução supondo que um determinado objeto que apresenta determinadas características (“features”) é persistente no ambiente. Caso ele não esteja presente, conseguimos rapidamente adaptar a solução à esta nova situação. Claramente, vemos pelo exemplo acima que o raciocínio baseado no senso comum é um caso de raciocínio não-monotônico.

A nossa crença da situação (ou estado) corrente também muda de uma forma não-monotônica. Considerando o tempo através da ocorrência de uma sucessão de eventos discretos, vemos que o estado corrente pode ser representado por um conjunto de crenças correntes e o estado anterior por um conjunto de crenças anteriores. O que é importante observar é que o conjunto de crenças não evolui conforme uma seqüência monótona de ocorrência de eventos, por exemplo, a partir do estado anterior para o estado corrente porque muitas ações modificam o valor verdade do que supomos verdadeiro num determinado estado, sem o nosso controle. Este problema de representar e atualizar corretamente o estado do mundo é conhecido por “frame problem”. O “frame problem”

foi discutido no item 4.1.2. [Hintikka 1962] apresenta uma forma de semântica dos mundos possíveis para a lógica modal de conhecimento e de crença.

O problema do controle é o problema de determinação da estratégia de solução do problema, i.e., o que o mecanismo de inferência (MI) deve fazer em seguida. Ao invés de tomar suas decisões cegamente, ao acaso, [Doyle 1979] sugere que devemos usar o MI para fazer inferências acerca de quais inferências fazer em seguida, i.e., o MI deve ser um agente racional. O do agente racional que deve apresentar a noção de crença, de desejo e de intenção. Usando esta representação reflexiva do MI, i.e., um modelo de si mesmo ("self model"), as regras de inferência tornam-se regras de auto-modificação das crenças do MI (e, como consequência, também, de seus desejos e intenções).

6.3.2 A Evolução dos Estados Usando Rede de Petri

[Doyle 1979] apresenta uma forma de dedução da crença, baseada no cálculo proposicional. Entretanto, apesar de correta, esta pode ser explosiva combinatoriamente para tratar da manutenção da verdade em bases de conhecimento muito grandes.

A nossa proposta pode ser vista como uma alternativa para o uso da Lógica Dinâmica [Silva 1992] e para o sistema de manutenção da verdade (TMS) [Doyle 1979], descrita no item 6.1.2, acima. A nossa representação faz um modelamento dos sistemas vistos através da ocorrência de eventos discretos.

Observe que [Doyle 1979] usa termos que são verdadeiros (mas não constituem uma *crença* -- apenas uma crença potencial) pode ser visto como a topologia da rede de Petri -- a topologia da rede representa todas as soluções possíveis e nenhuma delas em particular. No TMS, apenas a crença justificada como verdadeira é que determina a situação (o estado) corrente. Do mesmo modo, na rede de Petri, apenas a marcação corrente é que determina o estado corrente. Portanto, concluímos que ambas abordagens conseguem capturar a mudança dos estados de um sistema (dinâmico) que se modifica de uma forma não-monotônica. Ambas representam todos os estados potenciais e também um estado corrente em particular.

O uso da representação em rede de Petri tem a vantagem da visualização gráfica da evolução destes estados discretos dos sistemas, além de resolver igualmente os três problemas apontados no item 6.1.1, acima: o senso comum, o “frame problem” e o controle.

No caso do exemplo envolvendo a persistência de um objeto (exemplo de senso-comum), a rede de Petri pode representar a presença ou não de objetos através da existência ou não de marcas em lugares da RdP e efetuar o disparo das transições de acordo com a distribuição destas marcas.

Do mesmo modo, faz-se a resolução do “frame problem” (ou seja, o problema da explosão combinatória das sentenças para se definir explicitamente o que muda e o que permanece após a execução de uma ação), para o problema de “planning”, fazendo-se a representação do problema em rede de Petri. Ao invés de definir inúmeras sentenças em lógica proposicional, basta olhar a dinâmica expressa pela marcação da rede de Petri.

O uso de um modelo de si mesmo (“self model”) do MI, proposto em [Doyle 1979], para o controle do que fazer em seguida, coincide com a nossa proposta de elaborar um modelo conceitual (mc), descrito no cap. 4, usando a teoria da rede de Petri para representar o problema de “planning”. Este modelo conceitual constitui um “mapa rodoviário” ao alcance da mão do Planejador e descreve todo o processo de “planning”, inclusive estratégias para a elaboração da solução.

6.4 - Trabalhos Futuros

Consideramos como sendo necessárias, desejáveis e possíveis, as seguintes melhorias em relação à nossa proposta:

- evolução da rede de Petri para RdP “dobrada”
- programação usando linguagem Orientada a Objetos
- sistema de execução e de supervisão em tempo-real
- implementação do comportamento reativo puro
- axiomatização da rede de Petri em lógica

Evolução da rede de Petri para RdP “dobrada”:

Neste trabalho, utilizamos diferentes tipos de redes de Petri para representar o domain-K. Mais especificamente, usamos a rede de Petri do tipo C/E, no problema do Mundo do Blocos e usamos a rede de Petri estendida Ghenesys no Roteador de Helicópteros. Para evitar ter de trabalhar com diferentes tipos de rede ou com redes muito extensas, o ideal é usar uma única forma de rede dobrada mais expressiva -- rede com marcas individuais. Esta rede dobrada poderia ser: a própria estendida Ghenesys [Silva et all 1996], o E-MFG [Santos 1993], [Santos 1993a], ou redes de alto nível [Jensen 1996].

Uso de uma Linguagem Orientada a Objetos:

A implementação feita não usa uma linguagem de programação Orientação a Objetos. Tanto o problema exemplo do Mundo de Blocos, assim como o Sistema Roteador de Helicópteros e o Scheduling Job Shop foram implementados na linguagem Visual PROLOG para Windows. A rede estendida Ghenesys [Silva et all 1996], [Silva et Miyagi 1996] é uma rede de Petri Orientada a Objetos. A tradução da rede de Petri estendida Ghenesys para uma linguagem Orientada a Objetos preservaria melhor as características inerentes à tecnologia de Objetos, por exemplo, de herança, de encapsulamento, etc.

Sistema de execução e de supervisão em tempo-real:

O módulo que corresponde à Supervisão e à Execução do Plano está localizado na camada de Controle do SIM. Por este motivo, também é perfeitamente compatível com o modelamento por Rede de Petri.

Para acoplar o Planejador com um sistema de Execução do plano em tempo real, deve-se integrar o computador onde reside o Planejador com um sistema de controle sobre o dispositivo que irá executar o plano. O plano elaborado deve ser representado como uma rede de Petri [Paiva 1993]. Além disso, para permitir a implementação de procedimentos de recuperação de erros e procedimento de ajustes na atuação do dispositivo, deve existir um sistema supervisorio acoplado.

Existe uma pesquisa em andamento no LAS - Laboratório de Automação de Sistemas (ou "Mecatrônica") para desenvolvimento de um sistema supervisorio inteligente baseado em IA e Redes de Petri para aplicação em edifícios inteligentes.

Implementação do comportamento reativo puro:

[Georgeff et Lansky 1987] propuseram um sistema puramente reativo. Num sistema dinâmico, nem todas as ações que ocorrem no mundo estariam sob o controle do agente planejador. A dinâmica dos sistemas do mundo real deve-se, principalmente, à coexistência de múltiplos agentes, nem sempre cooperativos.

O APE [Hendler 1992] implementa a idéia de sistemas emergenciais que constitui uma evolução em relação aos sistemas reativos. [Hendler 1992] chama a atenção para uma diferença sutil que existe entre esta idéia de "emergência" e os sistemas reativos, porque nestes o comportamento reativo é dirigido pela ocorrência de uma falha, enquanto que nos sistemas "emergenciais", geralmente, esta falha não ocorre. No APE, apenas os aspectos procedimentais dos operadores APE estão especificados de forma semelhante às redes de Petri do tipo C/E. Entretanto, para que a reatividade se torne automática,

é preciso embutir (“embed”) a rede de Petri, traduzindo-a para axiomas em lógica.

Neste caso, é preciso rigor tanto no processo de modelagem assim como no processo de axiomatização da Rede de Petri.

A nossa proposta de fundamentar o desenvolvimento de sistemas baseado na elaboração do modelo conceitual em rede de Petri permanece válida nesta proposta. Inclusive, a nossa proposta de elaboração do modelo da aplicação (“application model”) e do modelo do planejador (“self model”) também continua válida.

Axiomatização da rede de Petri em lógica:

Para obter uma implementação mais formal da rede de Petri em lógica, deve-se fazer a implementação da tradução do modelo conceitual em lógica. Como já mencionamos no capítulo 3, existem duas propostas para se fazer a axiomatização da rede de Petri em lógica: a lógica dinâmica [Silva 1992], [Tuomin 1989] e a lógica linear [Valette 1992].

A Lógica Transacional Paralela (LTP) [Bonner et Kifer 1995], surge como uma possível alternativa para se fazer esta axiomatização das Redes de Petri. A LTP deriva da Lógica Transacional seqüencial (LT), que trata da mudança de estados em programas lógicos e em Banco de Dados. A LT seria aplicável a um grande leque de tarefas: na programação de transações e gatilhos em Bancos de Dados, na simulação e no “IA planning” de robôs, e na especificação de ações em sistemas dinâmicos.

A Lógica Transacional Paralela (LTP) foi proposta para tratar da programação e execução de processos de comunicação e de transações paralelas em Bancos de Dados. A LTP apresenta os conectivos da lógica de primeira ordem, \wedge e \neg e o quantificador \forall . A LTP apresenta dois novos conectivos adicionais, \otimes e $|$, e um operador modal, \odot . Os demais conectivos são obtidos em função dos anteriores: \vee , \leftarrow , \oplus , $\$$. Os conectivos \oplus

e $\$$ são conectivos duais de \otimes e $|$, respectivamente. Sejam α e β dois termos que representam duas transações (ou ações) sobre um Banco de Dados.

Então,

- i) \otimes denota uma conjunção seqüencial: $\alpha \otimes \beta$ significa que as transações α e β são executadas em seqüência.
- ii) $|$ denota uma conjunção paralela: $\alpha | \beta$ significa que as transações α e β são executadas em paralelo.
- iii) $\odot \alpha$ denota uma atomicidade: a execução de α não pode estar intercalada com nenhuma outra transação, ou seja, a ação α deve ser executada como um átomo.

Na LTP os itens dados são agrupados num Banco de Dados (ou estado). Intuitivamente, um item de dado representa qualquer tipo de objeto persistente que existe no mundo. O acesso ao Banco de Dados se faz através de dois procedimentos (ou oracles): o oracle de dados o oracle de transição. Um *oracle de dados* (O^d) é usado para retornar dados a partir de um BD e um *oracle de transição* (O^t) é usado para atualizar dados num BD. Um oracle de dados (O^d) provê uma semântica estática para um item de dado, enquanto que um oracle de transição (O^t) provê uma semântica dinâmica.

[Bonner et Kifer 1995] relatam que a LTP pode prover um repertório de semântica para Banco de Dados e um mecanismo acoplado para se adicionar novas semânticas. Os autores exemplificam inúmeros predicados dedicados para domínios específicos de aplicação:

- i) **Banco de dados relacionais:** Um oracle de dados (O^d) simplesmente retorna o valor de uma fórmula existente no Banco de Dados ou no estado (D), i.e., ($O^d(D) = D$). Um oracle de transição $O^t(D_1, D_2)$ promove a inserção ou eliminação de átomos de D_1 , resultando D_2 . Dois novos predicados *p.ins* e *p.del* efetuam atualizações no BD no estilo SQL (note que é usada uma notação do tipo objeto (“object-like”) no LTP).

Formalmente, $p.ins(x) \in \mathcal{O}^t(D_1, D_2)$ sse $D_2 = D_1 \cup \{p(x)\}$. Do mesmo modo, $p.del(x) \in \mathcal{O}^t(D_1, D_2)$ sse $D_2 = D_1 - \{p(x)\}$.

ii) Comunicação de Dados: O estado do mundo é representado por um conjunto de filas, cada uma representando um canal de comunicação. Cada fila é identificada por um nome, q , e contém uma lista de mensagens. É definido um oracle de dados (\mathcal{O}^d) *peek* que simplesmente faz a leitura de uma mensagem msg situada numa fila q no estado D , sem consumi-la. Formalmente, $peek(q, msg) \in \mathcal{O}^d(D)$ sse a mensagem msg está na cabeça da lista q no estado D . No domínio da comunicação de dados, São definidos dois oracles de transição $\mathcal{O}^t(D_1, D_2)$: *send* e *receive* que transmitem e recebem mensagens msg para/de um canal de comunicação q . Intuitivamente, *send* grava uma mensagem msg no final da fila de um canal de comunicação q , enquanto *receive* faz a leitura de uma mensagem m no canal q e a consome. Formalmente, define-se $send(q, msg) \in \mathcal{O}^t(D_1, D_2)$ sse $D_2 = D_1 \cup \{incluir\ msg\ no\ final\ da\ fila\ q\}$. E, $receive(q, msg) \in \mathcal{O}^t(D_1, D_2)$ sse $D_2 = D_1 \cup \{eliminar\ msg\ na\ cabeça\ da\ fila\ q\}$.

Vemos perspectivas favoráveis de uso da LTP e, especialmente, destes predicados de comunicação de dados na axiomatização de redes de Petri. O efeito do predicado *send* pode ser comparado com o efeito da aplicação da implicação clássica. Agora, note a semelhança que existe entre o comportamento do predicado *receive* e a implicação linear ((0—) [Girard 1987] -- a mensagem é eliminada do canal de comunicação depois que termina a sua leitura, ou seja, a mensagem só está disponível para leitura uma única vez. A implicação linear foi usada por [Valete et Courvoisier 1992] para axiomatizar o disparo de transições de uma rede de Petri. Observe também, a semelhança entre o predicado *peek* e a persistência de marcas que envolve as portas-habilitadoras e portas-inibidoras que existe na rede estendida Ghenesys.

iii) Regras de Horn Paralelas: [Bonner et Kifer 1995] definem as regras de Horn paralelas para se fazer a especificação de transações que combinam processos seqüenciais e paralelos. Uma *regra de Horn paralela* é denotada por: $\alpha \leftarrow \beta$, onde α é uma fór-

mula atômica e β é uma meta serial paralela. Uma meta serial paralela (“concurrent serial goal”) é uma fórmula tal que β :

- i) é uma fórmula atômica,
- ii) $\beta = (\beta_1 \otimes \dots \otimes \beta_k)$, onde cada β_i é uma meta serial paralela e $k \geq 0$,
- iii) $\beta = (\beta_1 \mid \dots \mid \beta_k)$, onde cada β_i é uma meta serial paralela e $k \geq 0$,
- iv) $\beta = \odot \beta$, onde cada β é uma meta serial paralela.

O corpo destas regras consistem de fórmulas atômicas conectadas através de conjunções do tipo seqüencial ou paralela. Cada fórmula atômica representa uma transação i.e., uma consulta ou uma atualização. Exemplo, seja a regra de Horn paralela:

$$a \leftarrow (b_1 \otimes b_2) \mid (c_1 \otimes c_2)$$

Intuitivamente, tem-se que “para executar a transação a , deve-se executar paralelamente as duas transações $(b_1 \otimes b_2)$ e $(c_1 \otimes c_2)$. Para executar $(b_1 \otimes b_2)$, deve-se executar primeiro a transação (b_1) e, depois executar em seqüência a transação (b_2) . Idem para a transação $(c_1 \otimes c_2)$ ”. Portanto, a cabeça de uma regra de Horn paralela identifica a regra, atribuindo-lhe um nome, enquanto que o corpo da regra faz a definição dos processos que a compõem.

Além disso, uma regra de Horn paralela deve apresentar o comportamento de uma transação, i.e., deve ser executada atômicamente. Uma regra de Horn paralela deve ser executada inteiramente, com sucesso em todas as suas transações, ou então falhar completamente. Isto é muito importante no domínio do Banco de Dados para não permitir que a execução incorreta de transações deixem o BD inconsistente.

[Bonner et Kifer 1995] observam que isto não é o que ocorre com outras linguagens de programação, como por exemplo o PROLOG. Suponha uma transação do tipo $t \leftarrow b_1 \otimes b_2$ envolvendo dois processos de atualização num BD. O processo de tradução para PROLOG consiste em substituir (\otimes) por $(,)$ e substituindo os oracles de

atualização do BD por “assertz” e “retract”. Em caso de ocorrer uma falha na segunda transação b_2 , o PROLOG não irá desfazer a atualização já feita com sucesso para a primeira transação b_1 e, conseqüentemente, acaba deixando o BD inconsistente.

A possibilidade de se especificar processos concorrentes e sequenciais na LTP é uma característica interessante desta lógica que merece uma investigação mais aprofundada. As Redes de Petri apresentam, justamente, esta mesma característica que as diferenciam das linguagens puramente declarativas (PROLOG, LISP) e das linguagens procedimentais.

Através da axiomatização do modelo conceitual do domínio da aplicação e do problema de “planning” representados através de Redes de Petri, podemos obter:

- i) Construtores de Planos melhores, usando sistemas de dedução automática (provadores de teoremas), por exemplo, usando esta LTP, possivelmente, operando sobre uma Base de conhecimentos que traduz o *mc* com mais fidelidade.
- ii) Execução automatizada do plano representado como Rede de Petri, inclusive com Redes de Petri “compiladas” em lógica e embutidas (“embedded”) nos sistemas de tempo-real para produzir o comportamento reativo de baixo nível, como descrito em [Hendler 1992]. Costuma-se dizer que um “especialista” humano possui regras “compiladas” e que não precisa mais “pensar” para realizar ações por reflexo e mesmo para realizar determinadas tarefas repetitivas.
- iv) Sistema inteligentes autônomos, que apresentem funcionalidades integradas de tempo-real para efetuar simultaneamente: a monitoração, o “planning” e “scheduling”, a execução e a supervisão da execução do plano, possivelmente fazendo as correções de rumo que se fizerem necessárias (“compliance”).
- v) e ainda, possivelmente, outras aplicações que não foram sequer mencionadas.

*REFERÊNCIAS
BIBLIOGRÁFICAS*

REFERÊNCIAS BIBLIOGRÁFICAS

- [Araribóia 1989], Grupo ARARIBÓIA. Inteligência artificial: um curso prático. Livros Técnicos e Científicos Ltda. 1989. Rio de Janeiro.
- [Baker 1974] BAKER, K.R. Introduction to Sequencing and Scheduling, John Wiley. 1974. New York.
- [Bonner et Kifer 1995] BONNER, A.J., KIFER, M. Concurrent Transaction Logic: Semantics and Proof Theory. (private communication). 1995.
- [Day 1992] DAY, David. Acquiring Search Heuristics Automatically for Constraint-Bases Planning and Scheduling. in: HENDLER, J. Proceedings of the First International Conference on Artificial Intelligence Planning Systems. pp. 45-51. 1992. Morgan Kaufmann. California. USA.
- [Doyle 1979] DOYLE, John. A TMS: True Maintenance System. Artificial Intelligence, 1979, vol. 12. pp. 231-273.
- [Brinati 1993] BRINATI, M.A., BOTTER R.B., et all. Heurísticas de Roteamento. Portos e Navios. jun 1993. pp 26-30.
- [Clark et Wright 1964] CLARK G. et WRIGHT J.W. Scheduling of vehicles from a central depot to a number of delivery points. Operations Research, 1964, vol. 12, pp. 568-581.
- [Clocksin et Mellish 1987] CLOCKSIN W.F., MELLISH C.S., Programming in Prolog. 1987. Springer-Verlag. Germany.
- [Costa 1992] COSTA, M. M. do Carmo, Introdução à Lógica Modal Aplicada à Computação. VIII Escola de Computação, 3 a 12.ago.1992. Gramado - RS. Instituto de Informática da UFRGS.
- [Drummond 1985], DRUMMOND, M.E. Refining and extending the procedural net. IJCAI 1985. In: ALLEN, J.; HENDLER, J.; TATE, A., coord. **Readings in planning**. San Mateo, California. Morgan Kaufmann, 1990. Cap. 10, p.667-669.
- [Fox 1984], FOX, M.; SMITH, S.F. ISIS - a knowledge-based system for factory scheduling. Expert Systems, July 1984, Vol.1, n.1. Id. Ibid. Cap. 5, p.336-360.
- [Fikes, Hart et Nilsson 1972], FIKES, R.E.; HART, P.; NILSSON, N.J. Learning and executing generalized robot plans. Artificial Intelligence 3(4), 1972. Id. Ibid. Cap.4, p.189-206.

- [Fikes et Nilsson 1971], FIKES, R.E.; NILSSON, N. STRIPS: A new approach to the application of theorem proving to problem solving. 1971. Id. Ibid. Cap.2, p.88-97.
- [Fischer 1979] FISCHER. M.J., LADNER R.E., Propositional Modal Logic of Programs, J. Comput. Sci., 18, 1979.
- [Frank et Schmidt 1993] FRANK, M, SCHMIDT, V. Petri Netze für Windows, Version 2.0 1993. Germany.
- [Georgeff 1987], GEORGEFF, M.P. Planning. Ann. Rev. Comput. Sci. 1987. In: ALLEN, J.; HENDLER, J.; TATE, A., coord. **Readings in planning**. San Mateo, California. Morgan Kaufmann, 1990. Cap.1, p.5-25.
- [Girard 1987] GIRARD, J.Y. Linear Logic. Theoretical Computer Science. 50. 1987.
- [Hayes 1973] HAYES, Patrick J. The frame problem and related problems in artificial intelligence. In: ALLEN, J.; HENDLER, J.; TATE, A., coord. **Readings in planning**. San Mateo, California. Morgan Kaufmann, 1990. Cap. 3, p.588-595.
- [Hayes 1975], HAYES, Philip J. A representation for robot plans. IJCAI, 1975. In: ALLEN, J.; HENDLER, J.; TATE, A., coord. **Readings in planning**. San Mateo, California. Morgan Kaufmann, 1990. Cap. 3, p.154-162.
- [Hayes-Roth 1979] HAYES-ROTH, Barbara et Frederick. A Cognitive Model of Planning. In: ALLEN, J.; HENDLER, J.; TATE, A., coord. **Readings in planning**. San Mateo, California. Morgan Kaufmann, 1990. Cap. 4, p.245-262.
- [Harel 1979] HAREL, D., First-Order Dynamic Logic. Lecture Notes on Computer Science, 68, 1979.
- [Hendler 1992], HENDLER, J. et SPECTOR, L. Planning and control across supervenient levels of representation. Department of Computer Science. University of Maryland. Third Draft: sept 1992, for International Journal on Intelligent & Cooperative Information Systems (IJICIS).
- [Haugeland 1985] HAUGELAND, J. Artificial Intelligence: The Very Idea. 1985. The MIT Press. Cambridge, Mass.
- [Hintikka 1962] HINTIKKA, J. Knowledge and Belief. 1962. Cornell University Press. Ithica. USA.

- [Hughes et Creswell 1968] HUGHES, G.E., CRESSWELL, M.J. An Introduction to Modal Logic, 1968. Methuen and Co. Ltd. London.
- [Jensen 1996] JENSEN, Kurt. Coloured Petri Nets: Basic Concepts, Analysis, Methods and Practical Use. Vol. 1/2. Springer 1996.
- [Kusiak 1990], KUSIAK, Andrew. Intelligent manufacturing systems. Prentice-Hall, Englewood Cliffs. 1990. New Jersey.
- [Lifschitz 1987], LIFSCHITZ, V. On the semantics of STRIPS. Stanford University, 1987. In: ALLEN, J.; HENDLER, J.; TATE, A., coord. **Readings in planning**. San Mateo, California. Morgan Kaufmann, 1990. Cap.7, p.523-530.
- [Maher 1990], MAHER, M. Process models for design synthesis. AI Magazine, Winter 1990.
- [Mädler 1992] MÄDLER, Fritz. Towards Structural abstraction. in: HENDLER, J. Proceedings of the First International Conference on Artificial Intelligence Planning Systems. pp. 163-171. 1992. Morgan Kaufmann. California. USA.
- [McCarthy et Hayes 1969] Mac CARTHY J. HAYES P.J., Some Philosophical Problems from the Standpoint of Artificial Intelligence. 1969. Mach. Intell. 4, pp. 463-502.
- [McDermott 1978], McDERMOTT, D. Planning and acting. Cognitive Science 2(2), 1978. In: ALLEN, J.; HENDLER, J.; TATE, A., coord. **Readings in planning**. San Mateo, California. Morgan Kaufmann, 1990. Cap4., p.225-244.
- [Mc Dermott 1982] Mc DERMOTT, D. A temporal logic for reasoning about processes and plans. 1982. Cognit. Sci., 6. pp.101-155.
- [Miyagi 1988] MIYAGI, Paulo E. Control System Design, Programming and Implementation for Discrete Event Production Systems by Using Mark Flow Graph. Tokyo, Jan 1988. Doctor Thesis. Tokyo Institute of Technology. Japan.
- [Miyagi 1988a], MIYAGI, P.E. et FURUKAWA, C. Mark flow graph e production flow schema: unificação e estruturação de sistemas de controle. 3°. CONAI, P.257-264, 1988. São Paulo.
- [Miyagi 1989], MIYAGI, P.E., HASEGAWA, K, et TAKAHASHI, K. Mark flow graph (MFG) para modelamento e controle de sistemas de eventos discretos. Monografias em Automação e Inteligência Artificial, vol.1, nº.1, 1989. Departamento de Engenharia Mecânica, Escola Politécnica. Universidade de São Paulo.

- [Miyagi 1990], MIYAGI, P.E. Modelagem e controle de sistemas produtivos - aplicação da teoria de redes de petri. Monografias do Departamento de Engenharia Mecânica nº. 55, 1990. Escola Politécnica, Universidade de São Paulo.
- [Miyagi 1993], MIYAGI, P.E. Automação de sistemas sequencias. Livro-texto, 1993. Curso de Engenharia Mecatrônica. Escola Politécnica. Universidade de São Paulo.
- [Miyagi 1993a], MIYAGI, P.E. Introdução ao projeto de sistemas por redes de petri. Livro-texto, 1993. Curso de Engenharia Mecatrônica. Escola Politécnica. Universidade de São Paulo.
- [Murata 1989] MURATA, T. Petri Nets: Proprieties, Analysis ans Applications. Proceed. of IEEE, vol. 77, n. 4, april 1989, USA.
- [Newell et Simon 1963], NEWELL, A.; SIMON, H.A. GPS, a program that simulates human thought. Computers and Thought, p. 279-293, 1963. In: ALLEN, J.; HENDLER, J.; TATE, A., coord. **Readings in planning**. San Mateo, California. Morgan Kaufmann, 1990. Cap. 2, p.59-66.
- [Paiva 1993] PAIVA, F.S. Sistema de Planejamento de Atividades de Uma Célula de Montagem. Dissertação (Mestrado). São Paulo. 1993. Escola Politécnica. Universidade de São Paulo.
- [PDC 1996] Prolog Development Center A / S, Visual PROLOG Version 4.0 - Language Tutorial, 1996. Denmark.
- [Peterson 1981] PETERSON, J.L. Petri Net Theory and the Modelling of Systems, 1981. Prentice-Hall.
- [Reisig 1985] REISIG. W., Petri Nets: An Introduction. 1985. Springer-Verlag. Berlin, Germany.
- [Rillo 1988] RILLO, Márcio. Aplicações de Redes de Petri em Sistemas de Manufatura. Tese (Doutorado). São Paulo. 1988. Escola Politécnica. Universidade de São Paulo.
- [Rosemberg 1995] ROSEMBERG, G. Elementary Net Systems. Lecture Notes of the Second International Course on Petri Nets for Latin America. Vol.1. Campina Grande. nov, 20-28, 1995. Universidade Federal da Paraiba. Brazil.
- [Sacerdoti 1974], SACERDOTI, E.D. Planning in a hierarchy of abstraction spaces. Artificial Intelligence 5, 1974. ALLEN, J.; HENDLER, J.; TATE, A., coord. **Readings in planning**. San Mateo, California. Morgan Kaufmann, 1990. Cap.2, p.98-108.

- [Sacerdoti 1975], SACERDOTI, E.D. The nonlinear nature of plans. IJCAI, 1975. Id. Ibid. Cap.3, p.162-170.
- [Santos 1993], SANTOS F °.D. J. Proposta de MFG estendido para modelagem e controle de sistemas integrados de manufatura. São Paulo 1993. Tese (Mestrado). Escola Politécnica. Universidade de São Paulo.
- [Santos 1993a] SANTOS F °, Diolino J., Realização do Controle de Tarefas de Veículos Autônomos de Transporte Através do MFG Estendido com Marcas Individuais. In: CONG-RESSO NACIONAL DE AUTOMAÇÃO, São Paulo, 1994, Anais : CONAI '94.
- [Sasaki 1996] SASAKI, E.M. Aplicação de Redes de Petri Auto-Modificáveis em Sistemas de Manufatura. Dissertação (Mestrado). São Paulo. 1996. Escola Politécnica. Universidade de São Paulo.
- [Shimad 1994] SHIMADA, L.M. Uma Metodologia baseada em Redes de Petri para o Planejamento Inteligente de Sistemas. EPUSP, 1994, Exame de Qualificação para Doutorado em Eng. Mecânica.
- [Silva 1992], SILVA, J.Reinaldo. Uma formalização do processo formal baseado em metáforas: sua aplicação na automatização de sistemas de eventos discretos. São Paulo 1992. Tese (Doutorado). Escola Politécnica. Universidade de São Paulo.
- [Silva 1992a], SILVA, J.Reinaldo. et PESSOA, F.J.B. Análise Semi-Automática de Mark Flow Graphs, Revista Robótica e Automatização, n°.10, nov 1992, pp. 25-30. Lisboa, Portugal.
- [Silva 1994], SILVA, J.Reinaldo., KAGOHARA, M.Y., TOLEDO, C.F.M., MIYAGI, P.E. Automatic Generation of Control Programs for Manufacturing Cells. Dept. of Mechanical Engineering, Escola Politécnica, Universidade de São Paulo. São Paulo 1994.
- [Silva 1994a] SILVA, J.R. COWAN, D.D. LUCENA, C.J.P. The Design Reusability Problem; A Case-Based Approach to the Design of Flexible Manufacturing Systems. Relatório Técnico USP, 1994.
- [Silva et Shimad 1995] SILVA, J. Reinaldo, SHIMADA, L. M. Um Sistema de Planejamento Semi-Reativo de Sistemas Produtivos Dinâmicos usando Redes de Petri. 2º Simpósio Brasileiro de Automação Inteligente. 13 a 15.set.1995. CEFET-PR. Curitiba.
- [Silva et Miyagi 1996] SILVA, J. Reinaldo, MIYAGI, Paulo E. A Formal Approach to PFS / MFG: a Petri Net Representation of Discrete Manufacturing Systems. Studies in Informatics and Control, Vol. 5, N. 2, june 1996. pp. 131-141.

- [Silva et all 1996] SILVA, J. Reinaldo. MIYAGI, Paulo E. SEBASTIÃO, J. Ricardo. RODRIGUES, Maurício S. A High Level Integrated Petri Net Environment for the Design of FMS. Proceedings of the IASTED - International Conference on Modeling, Simulation and Optimization. May, 6-9, 1996. Gold Coast. Australia.
- [Spivey et Thrall 1970] SPIVEY, W.A., THRALL, R.M. Linear Optimization. Holt, Rinehart and Winston. 1970. USA.
- [Sterling et Shapiro 1986] STERLING, L. SHAPIRO E., The Art of Prolog - Advanced Programming Techniques. 1986. The MIT Press. USA.
- [Sussman 1974], SUSSMAN, G.J. The virtuous nature of bugs. MIT AI Lab, 1974. In: ALLEN, J.; HENDLER, J.; TATE, A., coord. **Readings in planning**. San Mateo, California. Morgan Kaufmann, 1990. Cap.3, p.111-117.
- [Tate 1977], TATE, A. Generating project networks. IJCAI 1977. Id. Ibid. Cap.5, p.291-296.
- [Timlin et Pulley 1992] TIMLIN, M.T.F, PULLEYBLANK, W.R. Precedence Constrained Routing and Helicopter Scheduling: Heuristic Design. may-jun 1992, Interfaces 22:3. pp. 100-111.
- [Tuomin 1989] TUOMINEN, H., Elementary Net Systems and Dynamic Logic. Lecture Notes in Computer Science, 424, Springer-Verlag, 1989.
- [Valette 1992] VALETTE, R. Les réseaux de petri. LAAS/CNRS. Toulouse, mai 1992.
- [Valette et Courvoisier 1992] VALETTE, R. COURVOISIER, M. Petri Nets and Artificial Intelligence. Internationatl Workshop on Emerging Technologies for Factory Automation. North Queensland, Australia, aug 1992.
- [Valette et Silva 1989] VALETTE, R., SILVA. M. Petri Nets and Flexible Manufacturing. Lecture Notes in Computer Sciences. vol. 424. Springer-Verlag. 1989.
- [Valk 1978] VALK, R. Self-Modifying Nets, a Natural Extension of Petri Nets. Lecture Notes in Computer Science. vol. 62. Springer-Verlag, p. 464-476. 1978.
- [Vere 1983], VERE, S.A. Planning in time: windows and durations for activities and goals. IEEE 1983. In: ALLEN, J.; HENDLER, J.; TATE, A., coord. **Readings in planning**. San Mateo, California. Morgan Kaufmann, 1990. Cap.3, p.297-318.
- [Waldinger 1975], WALDINGER, R. Achieving several goals simultaneously. Machine Intelligence 8, 1975. Id. Ibid. Cap.3, pp. 118-139.

- [Wilensky 1981], WILENSKY, R. A model for planning in complex situations. *Cognition and Brain Theory* Vol. IV, n.4, Fall, 1981. Id. *Ibid.* Cap.4, p.263-274.
- [Wilkins 1984], WILKINS, D.E. Domain-independent planning: representation and plan generation. *Artificial Intelligence* 22) 1984. Id. *Ibid.* Cap.5, p.319-335.
- [Wilkins 1988] WILKINS, David E., *Practical Planning - Extending the Classical AI Planning Paradigm*. 1988. Morgan Kaufmann. San Mateo. California.