**Processos de decisão Markovianos
fatorados com probabilidades
imprecisas**

Karina Valdivia Delgado

TESE APRESENTADA
AO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DA
UNIVERSIDADE DE SÃO PAULO
PARA
OBTENÇÃO DO TÍTULO
DE
DOUTOR EM CIÊNCIAS

Programa: Ciência da Computação
Orientador: Prof. Dr. Leliane Nunes de Barros

Durante o desenvolvimento deste trabalho o autor recebeu auxílio financeiro da CAPES, do National ICT Australia (NICTA) e da Pró-Reitoria de Pós-Graduação da Universidade de São Paulo

São Paulo, janeiro de 2010

**Processos de decisão Markovianos
fatorados com probabilidades
imprecisas**

Este exemplar corresponde à redação
final da tese, devidamente corrigida
e defendida por Karina Valdivia Delgado,
aprovada pela Comissão Julgadora.

Banca Examinadora:

- Profa. Dra. Leliane Nunes de Barros (presidente) - IME-USP
- Profa. Dra. Anna Helena Reali Costa - EP-USP
- Prof. Dr. Fábio Gagliardi Cozman - EP-USP
- Prof. Dr. Marcus Vinicius Soledade Poggi de Aragão - PUC-RIO
- Profa. Dra. Graçaliz Pereira Dimuro - FURG

Dedicatória

À Karel, Elias e Sara; meus filhos, com amor e gratidão por sua compreensão e carinho.

À Juan e Dora; meus pais, pelo amor e orações.

À Felix pelo amor e compreensão.

Agradecimentos

À professora Leliane Nunes de Barros, pela orientação, sempre contribuindo com idéias muito valiosas para melhorar a tese e pela confiança e amizade que me brindou durante esses anos. Considero-a parte da minha família, pois sempre esteve preocupada com meu desenvolvimento pessoal e profissional.

Ao professor Scott Sanner do Centro de Pesquisa NICTA da Austrália, com quem tivemos a oportunidade de discutir a proposta de algumas soluções importantes dessa tese e que me recebeu na cidade de Canberra, Austrália, por 4 meses, com muita hospitalidade e constante interesse pela minha pesquisa.

Ao professor Fábio Gagliardi Cozman da EP-USP, pelos comentários valiosos relacionados à minha pesquisa e por ser o responsável por redescobrir os MDPIPs, há muito tempo adormecidos.

Aos professores do Departamento de Computação do IME-USP, que contribuíram para a minha formação nesse doutorado.

Agradeço ao IME e à USP pela oportunidade de realização do curso de doutorado nesta Instituição, bem como à Pró-Reitoria de Pós-Graduação da Universidade de São Paulo pelo apoio financeiro para apresentação de trabalhos internacionais.

À CAPES pelo apoio financeiro para realização desta pesquisa. Ao AAAI pelo apoio para participar do Doctoral Consortium do ICAPS 2008, onde surgiu a oportunidade de desenvolver parte do meu doutorado no NICTA, um dos maiores institutos internacionais de pesquisa. Ao NICTA pelo auxílio financeiro durante a visita de 4 meses naquele instituto de pesquisa australiano.

À meus amigos em São Paulo, Jesús, Christian, Vladi, Fabio, Edu, Thiago, José, Ricardo, Carlos, Pablo e Erika, com quem compartilhei momentos maravilhosos que nunca esquecerei. Especialmente ao Félix por estar comigo em todos os momentos, por seu amor e compreensão.

À minha família em Arequipa e especialmente aos meus pais Dora e Juan, pelo amor e apoio constante durante toda minha vida.

Aos meus filhos, Sara, Elias e Karel, que com suas palavras ternas e amorosas me deram a força necessária para continuar.

À Deus, meu pai celestial, por estar comigo sempre.

Resumo

Em geral, quando modelamos problemas de planejamento probabilístico do mundo real, usando o arcabouço de Processos de Decisão Markovianos (MDPs), é difícil obter uma estimativa exata das probabilidades de transição. A incerteza surge naturalmente na especificação de um domínio, por exemplo, durante a aquisição das probabilidades de transição a partir de um especialista ou de dados observados através de técnicas de amostragem, ou ainda de distribuições de transição não estacionárias decorrentes do conhecimento insuficiente do domínio. Com o objetivo de se determinar uma política robusta, dada a incerteza nas transições de estado, Processos de Decisão Markovianos com Probabilidades Imprecisas (MDP-IPs) têm sido usados para modelar esses cenários. Infelizmente, apesar de existirem diversos algoritmos de solução para MDP-IPs, muitas vezes eles exigem chamadas externas de rotinas de otimização que podem ser extremamente custosas. Para resolver esta deficiência, nesta tese, introduzimos o MDP-IP *fatorado* e propomos métodos eficientes de programação matemática e programação dinâmica que permitem explorar a estrutura de um domínio de aplicação. O método baseado em programação matemática propõe soluções aproximadas eficientes para MDP-IPs fatorados, estendendo abordagens anteriores de programação linear para MDPs fatorados. Essa proposta, baseada numa formulação multilinear para aproximações robustas da função valor de estados, explora a representação fatorada de um MDP-IP, reduzindo em ordens de magnitude o tempo consumido em relação às abordagens não-fatoradas previamente propostas. O segundo método proposto, baseado em programação dinâmica, resolve o gargalo computacional existente nas soluções de programação dinâmica para MDP-IPs propostas na literatura: a necessidade de resolver múltiplos problemas de otimização não-linear. Assim, mostramos como representar a função valor de maneira compacta usando uma nova estrutura de dados chamada de Diagramas de Decisão Algébrica Parametrizados, e como aplicar técnicas de aproximação para reduzir drasticamente a sobrecarga computacional das chamadas a um otimizador não-linear, produzindo soluções ótimas aproximadas com erro limitado. Nossos resultados mostram uma melhoria de tempo de até duas ordens de magnitude em comparação às abordagens tradicionais enumerativas baseadas em programação dinâmica e uma melhoria de tempo de até uma ordem de magnitude sobre a extensão de técnicas de iteração de valor aproximadas para MDPs fatorados. Além disso, produzimos o menor erro de todos os algoritmos de aproximação avaliados.

Palavras-chave: Processos de Decisão Markovianos, planejamento probabilístico, planejamento sob incerteza, planejamento robusto.

Abstract

When modeling real-world decision-theoretic planning problems with the framework of Markov Decision Processes (MDPs), it is often impossible to obtain a completely accurate estimate of transition probabilities. For example, uncertainty arises in the specification of transitions due to elicitation of MDP transition models from an expert or data, or non-stationary transition distributions arising from insufficient state knowledge. In the interest of obtaining the most robust policy under transition uncertainty, Markov Decision Processes with Imprecise Transition Probabilities (MDP-IPs) have been introduced. Unfortunately, while various solution algorithms exist for MDP-IPs, they often require external calls to optimization routines and thus can be extremely time-consuming in practice. To address this deficiency, we introduce the *factored* MDP-IP and propose efficient dynamic programming and mathematical programming methods to exploit its structure. First, we derive efficient approximate solutions for Factored MDP-IPs based on mathematical programming. To do this, we extend previous linear programming approaches for linear-value approximation in Factored MDPs, resulting in a multilinear formulation for robust “maximin” linear-value approximations in Factored MDP-IPs. By exploiting factored structure in MDP-IPs we are able to demonstrate orders of magnitude reduction in solution time over standard exact non-factored approaches. Second, noting that the key computational bottleneck in the dynamic programming solution of factored MDP-IPs is the need to repeatedly solve nonlinear constrained optimization problems, we show how to target approximation techniques to drastically reduce the computational overhead of the nonlinear solver while producing bounded, approximately optimal solutions. Our results show up to two orders of magnitude speedup in comparison to traditional “flat” dynamic programming approaches and up to an order of magnitude speedup over the extension of factored MDP approximate value iteration techniques to MDP-IPs while producing the lowest error among all approximation algorithms evaluated.

Keywords: Markov Decision Processes, probabilistic planning, planning under uncertainty, robust planning.

Sumário

Lista de Abreviaturas	v
Lista de Símbolos	vii
Lista de Figuras	xi
Lista de Tabelas	xv
Lista de Algoritmos	xvii
1 Introdução	1
1.1 Objetivo	5
1.2 Principais Contribuições dessa Tese	5
1.3 Organização	6
I Fundamentos	7
2 Probabilidades e Planejamento sob Incerteza	9
2.1 Probabilidades: Conceitos Básicos	9
2.1.1 Espaço amostral e eventos	9
2.1.2 Medidas de probabilidade	9
2.1.3 Origem das probabilidades	10
2.1.4 Probabilidade condicional e probabilidade conjunta	10
2.1.5 Teorema de Bayes	10
2.1.6 Variáveis aleatórias	11
2.1.7 Função de densidade de probabilidade	11
2.1.8 Densidade de probabilidade conjunta e marginalização	11
2.1.9 Esperança	12
2.1.10 Conjunto de medidas de probabilidade	12
2.2 Modelos Gráficos para Inferência	13
2.2.1 Redes Bayesianas	13
2.2.2 Redes Credais	15
2.3 Tomada de Decisão Sequencial e Planejamento	16

2.3.1	Planejamento não-determinístico	17
2.3.2	Planejamento probabilístico	17
2.3.3	Planejamento não-determinístico e probabilístico	18
2.4	Resumo	18
3	Processo de Decisão Markoviano	19
3.1	MDP: Definição	19
3.1.1	Avaliação da política	20
3.1.2	Política ótima	21
3.2	A Complexidade de Resolver um MDP	21
3.3	MDP: Soluções Baseadas em Programação Dinâmica	22
3.3.1	Programação dinâmica síncrona	22
3.3.2	Programação dinâmica assíncrona	25
3.4	MDP: Soluções Baseadas em Programação Matemática	27
3.4.1	Solução exata baseada em programação linear	28
3.4.2	Solução aproximada baseada em programação linear	28
3.5	Resumo	29
4	Processo de Decisão Markoviano Fatorado	31
4.1	Modelo Fatorado	31
4.1.1	Recompensa local	31
4.1.2	Modelo de transição fatorado	32
4.2	Diagramas de Decisão Algébrica	34
4.3	MDP Fatorado: Soluções baseadas em Programação Dinâmica	39
4.3.1	SPUDD	39
4.3.2	APRICODD	42
4.4	MDP Fatorado: Soluções baseadas em Programação Matemática	42
4.5	Resumo	44
5	Processo de Decisão Markoviano com Probabilidades Imprecisas	45
5.1	MDP-IP Enumerativo: Definição	45
5.1.1	MDP-IP: Solução baseada em programação dinâmica	47
5.1.2	MDP-IP: Solução baseada em programação matemática	48
5.2	Subclasses de MDP-IP	49
5.3	Resumo	49
II	Soluções Eficientes para MDP-IP	51
6	MDP-IP Fatorado	53
6.1	MDP-IP Fatorado: Definição	54

6.2	Diagramas de Decisão Algébrica Parametrizados	56
6.2.1	PADD: Definição formal e propriedades	56
6.2.2	Algoritmo para reduzir PADDs	57
6.2.3	Operações binárias em PADDs	58
6.2.4	Operações unárias em PADDs	62
6.3	Resumo	62
7	Soluções Baseadas em Programação Dinâmica	65
7.1	Iteração de Valor para MDP-IP Fatorado	65
7.1.1	O algoritmo SPUDD-IP	65
7.1.2	SPUDD-IP: Funções implementadas	68
7.2	Iteração de Valor Aproximada para MDP-IP Fatorado	68
7.2.1	Algoritmo APRICODD-IP	70
7.2.2	Algoritmo OBJECTIVE-IP	71
7.3	Resumo	74
8	Solução Baseada em Programação Matemática	75
8.1	Formulação em Dois Níveis e Formulação Multilinear	75
8.2	Como Resolver um MDP-IP Fatorado Baseado na Formulação Multilinear?	76
8.3	Uma Solução Eficiente para MDP-IPs Fatorados: FACTOREDMPA	77
8.3.1	Passo de simplificação: Explorando a estrutura fatorada de um MDPIP	77
8.3.2	Passo de contração: Gerando um conjunto compacto de restrições	78
8.3.3	O algoritmo FACTOREDMPA	81
8.4	Resumo	83
III	Resultados Empíricos e Considerações Finais	87
9	Domínios de Teste	89
9.1	Domínio FACTORY	89
9.2	O Novo Domínio: TRAFFIC	90
10	Análise dos Algoritmos Propostos	97
10.1	Avaliando Nosso Conjunto de Algoritmos de ITERAÇÃO DE VALOR	98
10.1.1	EXACT FLAT VALUE ITERATION vs. SPUDD-IP	98
10.1.2	APRICODD-IP vs. OBJECTIVE-IP	98
10.2	Avaliando o Algoritmo FACTOREDMPA	101
10.2.1	Número de restrições no programa multilinear: Antes vs. depois de aplicar FACTOREDMPA	101
10.2.2	EXACT FLAT VALUE ITERATION vs. FACTOREDMPA	102
10.2.3	FACTOREDMPA: Funções base pares vs. simples	104

10.3	ITERAÇÃO DE VALOR Aproximada vs. Programação Multilinear Aproximada	104
10.4	Resumo dos Resultados	105
11	Trabalhos Correlatos	111
11.1	BMDP e MDPST	111
11.2	Controle Robusto	113
11.3	Jogos Markovianos	113
11.4	Formulação Bayesiana	114
12	Conclusões e Trabalhos Futuros	117
12.1	Resumo das Contribuições	117
12.2	Publicações Geradas	119
12.3	Trabalhos Futuros	119
A	Algoritmos RTDP e BRTDP	121
	Referências Bibliográficas	125

Lista de Abreviaturas

ADD	Algebraic Decision Diagram.
BDD	Binary Decision Diagram.
BE	Bellman Error.
BMDP	Bounded-parameter Markov Decision Process.
BNF	Backus Naur Form.
CPT	Conditional Probability Table.
CSI	Context-Specific Independence.
DAG	Directed Acyclic Graph.
DBN	Dynamic Bayesian Network.
DCN	Dynamic Credal Network.
DET	Deterministic Planning.
DNF	Did Not Finish.
MDP	Markov Decision Process.
MDPST	Markov Decision Process with Set-valued Transitions.
MDP-IP	Markov Decision Process with Imprecise Transition Probabilities.
NON-DET	Non-Deterministic Planning.
OADD	Ordered Algebraic Decision Diagram.
PADD	Parameterized ADD.
PTAE	Percentage True Approximation Error.
ROBDD	Reduced Ordered Binary Decision Diagram.
TAE	True Approximation Error.

Lista de Símbolos

$\vec{c}$	estado atual dos semáforos do domínio TRAFFIC.
d_i	Constante.
$error$	Erro de aproximação máximo.
g	Valor do termo i do polinômio.
$\hat{g}$	Valor do termo i do polinômio usando L_i e U_i .
g_i^a	Função intermediária.
h	Nó sucessor <i>high</i> do nó de decisão.
k	Número de funções base.
l	Nó sucessor <i>low</i> do nó de decisão.
m_2	Número de restrições relacionadas com p_{ij}
$\max(\cdot)$	Operação unária max em diagramas de decisão.
$\max(\cdot, \cdot)$	Operação binária max entre diagramas de decisão.
$maxIter$	Número máximo de iterações.
max_depth	máxima profundidade.
$\min(\cdot)$	Operação unária min em diagramas de decisão.
$\min(\cdot, \cdot)$	Operação binária min entre diagramas de decisão.
n	Número de variáveis de estado.
$\vec{p}$	Parâmetros de P .
p_a	probabilidade de chegar novos carros no domínio TRAFFIC.
$pa()$	Pais da variável de estado na DBN ou DCN.
p_{ij}^L	Valor mínimo possível para p_{ij} .
p_{ij}^U	Valor máximo possível para p_{ij} .
p_t	probabilidade de virar para direita do domínio TRAFFIC.
r_t	Recompensa obtida no estágio t .
t_{j_i}	veículo na célula x_{j_i} pretende virar para direita no domínio TRAFFIC.
tol	Tolerância.
var	Variável de teste no diagrama de decisão.
w_i	Pesos usados na combinação linear das funções base.
$\vec{x} = (x_1, x_2, \dots, x_n)$	Representação do estado usando variáveis de estado.
x_{j_i}	célula de intersecção do domínio TRAFFIC.
x_{k_i}	célula alimentadora de pista do domínio TRAFFIC.

A	Conjunto finito de ações.
$\mathcal{B}$	Conjunto de estados num MDPST.
C	Função custo.
$\mathfrak{C}$	Conjunto de equações lineares sobre $\vec{p}$.
$\mathcal{C}^a$	Conjunto de funções c_i^a .
$Conn(c_j, c_i)$	Denota a conexão do computador c_j para c_i .
$Const$	Valor constante.
$\mathcal{D}$	Domínio de planeamento.
$\mathfrak{E}$	Espaço de eventos.
E	Esperança.
$\mathcal{F}$	Função de transição de estados.
$\mathbb{F}$	Conjunto novo de funções $u_z^{e_r}$.
F	Diagrama de decisão.
F_h	Ramo verdadeiro da variável de teste no diagrama de decisão.
F_l	Ramo falso da variável de teste no diagrama de decisão.
$F _{x_i}$	Operação unária <i>restringir</i> em diagramas de decisão.
G	Conjunto de estados metas.
$H = \{h_1, \dots, h_k\}$	Conjunto de funções base.
I	Conjunto de estados iniciais.
$\mathbb{I}$	Função indicador.
$\mathbb{J}$	Conjunto de restrições.
K	Conjunto credal de transição.
L_1	Norma 1.
L_2	Norma 2.
L_i	Valor mínimo possível para o termo i usando p_{ij}^L ou p_{ij}^U .
L_∞	Norma infinita.
$\mathcal{M} = \langle S, A, R, C, P, \gamma \rangle$	Tupla que define formalmente um MDP.
$\mathcal{M}_{IP} = \langle S, A, R, K, \gamma \rangle$	Tupla que define formalment um MDP-IP.
O	Critério de Ordenação.
$\mathcal{P}$	Problema de planeamento.
P	Probabilidade de transição.
$Poly$	Polinômio do tipo $d_0 + \sum_i d_i \prod_j p_{ij}$.
P_{DD}	Probabilidade de transição representado por um diagrama de decisão.
Q	Valor do um estado usando uma ação.
Q_{DD}^t	Função Q t-estágios-para-frente representado por um diagrama de decisão.
R	Função recompensa.
$\mathcal{R}^a$	Conjunto de funções recompensa local R_j^a .
R_j	Recompensa local.
R_{DD}	Função recompensa representado por um diagrama de decisão.

S	Conjunto discreto e finito de estados.
T	Número de iterações para alcançar ϵ -otimalidade.
U_i	Valor máximo possível para o termo i usando p_{ij}^L ou p_{ij}^U .
V_{max}	Valor máximo possível em cada passo da ITERAÇÃO DE VALOR.
V^t	Função valor t-estágios-para-frente.
V_{DD}^t	Função valor t-estágios-para-frente representado por um diagrama de decisão.
V_π	Valor da política π .
V^*	Função valor ótima.
$\bar{V}_l$	Valor limite inferior estimado de V^* .
V_{l_0}	Limite inferior <i>admissível</i> da função valor.
$\bar{V}_u$	Valor limite superior estimado de V^* .
V_{u_0}	Limite superior <i>admissível</i> da função valor.
$\hat{V}(s)$	Função valor aproximada.
$\mathbf{X} = \{X_1, X_2, \dots, X_n\}$	Conjunto de variáveis de estado.
α	Parâmetro do critério maximix.
(α_i, β_i)	parâmetros da distribuição Beta.
γ	Fator de desconto.
δ	Fração do valor máximo possível.
ϵ	Perda na função valor.
π	Política.
π_V	Política gulosa com respeito a alguma função valor V .
π^*	Política ótima.
$\rho \in \{0, 1\}^n$	Atribuição de valores das variáveis $\{x_1, \dots, x_n\}$.
τ	constante maior do que 1.
ψ	Número de funções de recompensa local.
ω	Pontos amostrais.
Γ	Escopo da função g_i^a .
$\aleph_i$	Subconjunto de variáveis de estado.
Θ	Escopo da função c_i^a .
Ξ	informação adicional sobre o estado s .
Ω	Espaço amostral.
$\oplus$	Operação adição entre diagramas de decisão.
$\ominus$	Operação subtração entre diagramas de decisão.
$\otimes$	Operação multiplicação entre diagramas de decisão.
$\sum_{x_i \in X_i} P)$	Marginalização em diagramas de decisão.
$\#terms$	Número de termos do polinômio.

Lista de Figuras

1.1	Problema TRAFFIC.	3
1.2	Problema SYSADMIN com 7 computadores conetados na forma de anel unidirecional. O administrador da rede deve decidir qual máquina reinicializar em cada instante, sendo que existem eventos exógenos que fazem com que o computador deixe de funcionar ou passe a funcionar (representado pela fada na figura).	4
2.1	Densidade de probabilidade conjunta $P(X_1, X_2)$ e o resultado de aplicar a marginalização sobre X_2	12
2.2	Exemplo de conjunto credal válido representado pela região cinza. O conjunto credal é definido pelas triplas $\{P(x_1), P(x_2), P(x_3)\}$ que pertencem a essa região.	13
2.3	Rede Bayesiana para o exemplo do alarme. As tabelas indicam as probabilidades a priori e as probabilidades condicionais.	14
2.4	Rede Credal para o exemplo do alarme.	15
3.1	Exemplo de Atualização de Bellman para s_0 : $V^t(s_0) = \max_{a \in A} Q^t(s_0, a)$	23
3.2	O algoritmo de ITERAÇÃO DE VALOR executa Atualizações de Bellman para um MDP com 2 estados, s_1 e s_2 , e 3 iterações, calculando V^t baseado em V^{t-1}	23
3.3	RTDP atualiza $\bar{V}_u$ e escolhe uma ação gulosa a_2 , isto é, aquela com o maior valor Q_u^t	26
3.4	RTDP escolhe o próximo estado a ser visitado s_3 (simulação), baseado na distribuição de probabilidade e um sorteio.	26
3.5	Exemplo de uma simulação de RTDP que terminou ao encontrar o estado objetivo.	26
4.1	a) Uma Rede Bayesiana Dinâmica (DBN) para uma ação $a \in A$. b) Tabela de probabilidade condicional para $X'_2 = 1$. c) Tabela de probabilidade condicional para $X'_2 = 0$. d) ADD para representar a função de transição que chamaremos de $P_{DD}(x'_2 x_1, x_2, x'_1, a)$; a linha contínua indica o ramo verdadeiro ($X_i = 1$) da variável de teste e a linha tracejada indica o ramo falso ($X_i = 0$).	32
4.2	Topologias de rede para o exemplo SYSADMIN: a) anel unidirecional, b) anel bidirecional, (c) anéis bidirecionais independentes de pares de computadores e (d) estrela [Guestrin et al., 2003]	34

4.3	Um exemplo de função recompensa $R(x_1, x_2, x_3) = \sum_{i=1}^3 x_i$ representada como um ADD. A linha contínua indica o ramo verdadeiro ($X_i = 1$) da variável de teste e a linha tracejada indica o ramo falso ($X_i = 0$).	35
4.4	Representação gráfica de um ADD: <i>var</i> é o nome da variável, e F_h e F_l são os identificadores dos nós dos ramos verdadeiro e falso, respectivamente.	36
4.5	Um diagrama de decisão ordenado e a sua representação canônica. A linha contínua indica o ramo verdadeiro ($X_i = 1$) da variável de teste e a linha tracejada indica o ramo falso ($X_i = 0$).	36
4.6	a) Função f e g . b) Representação canônica de f e g	37
4.7	a) Funções f , g e $f + g$. b) ADDs canônicas da função f , g e da soma $f \oplus g$	37
4.8	a) Função f e $\max(f)$. b) ADD canônica de f e o resultado da operação $\max(f)$	38
4.9	a) Funções f , g e $\max(f, g)$. b) ADDs canônicas das funções f , g e do máximo $\max(f, g)$	38
4.10	a) Funções P e $P _{X_2=1}$ (i.e., a operação que considera unicamente as linhas da tabela em que $X_2 = 1$). b) Representação canônica de P e o resultado da operação $P _{X_2=1}$	38
4.11	a) As funções $P(X_1, X_2)$ e marginalização $\sum_{x_2} P$. b) Resultado da operação de marginalização $\sum_{x_2 \in X_2} P = F _{x_2=1} \oplus F _{x_2=0}$ sobre ADDs (como a soma de duas operações de restrição).	39
4.12	a) A função valor V^t representada como um ADD. b) O resultado do passo de redução de ADDs de APRICODD, que substitui os nós internos por um intervalo de valores.	42
5.1	Relações entre MDP-IP e suas subclasses [Trevizan et al., 2007].	49
6.1	a) Rede Credal Dinâmica para ação <i>notreboot</i> para uma topologia de anel unidirecional do domínio SYSADMIN com dois computadores. b) Tabela de probabilidade condicional para as variáveis $X'_1 = 1$ e $X'_2 = 1$ e as restrições relacionadas às probabilidades. c) A representação com ADD Parametrizado da probabilidade de transição chamada de $P_{DD}(x'_1 x_1, x_2, \text{notreboot})$. A linha contínua indica o ramo verdadeiro (1) da variável de teste e a linha tracejada indica o ramo falso (0).	55
6.2	a) Tabela de probabilidade condicional para a variável de estado X'_2 para a ação a_1 . b) A representação ADD Parametrizada para $P(X'_2 = 1 x_1, x_2, x_3, x_4, a_1)$	56
6.3	Um exemplo de aplicação do algoritmo REDUCEPADD (Algoritmo 3). a) Diagrama de entrada para o algoritmo. b) e c) Diagramas intermediários. d) Representação canônica para a entrada.	59
6.4	Notação usada nos algoritmos APPLY e CHOOSEVARBRANCH.	59
6.5	Um exemplo de multiplicação de PADDs.	62
6.6	Um exemplo de aplicação da operação <i>Restrição</i> e <i>Marginalização</i> em um PADD.	62

7.1	a) $V_{DD}^0 = R(x_1, x_2)$ para a topologia anel unidirecional do domínio SYSADMIN com 2 computadores multiplicado por $P_{DD}(x'_1 x_1, x_2, \text{notreboot})$. b) Resultado de $V_{ADD}^0 \otimes P_{DD}(x'_1 pa(X'_1), \text{notreboot}, \vec{p})$ e sua marginalização sobre x'_1	66
7.2	a) O PADD antes da minimização e um programa multilinear para a primeira folha, a solução para essa folha é o valor constante c_1 . b) O ADD resultante depois da minimização em todas as folhas.	67
7.3	a) Função valor V_{DD}^t representada como um ADD. b) Resultado da aplicação de APPROXADD em V_{DD}^t com aproximação $error = 1$. Note que as folhas cuja diferença esteja dentro dos limites de $error = 1$ foram fundidas calculando a média foi calculada e o ADD resultante foi simplificado.	71
9.1	Diagrama mostrando a intersecção de 2 pistas com células (quadros pontilhados) que representam as posições dos veículos e as variáveis de estado usadas na nossa descrição. Note que não modelamos células da pista na intersecção, uma vez que assumimos que os carros devam passar livremente pela intersecção, isto é, sem parar. Nesse modelo simplificado as filas possuem 3 células.	91
9.2	Rede Credal Dinâmica para o Problema TRAFFIC para as ações <i>change</i> e <i>no-change</i>	95
10.1	Comparação do tempo de execução dos algoritmos SPUDD-IP e ITERAÇÃO DE VALOR ENUMERATIVA para os problemas TRAFFIC, SYSADMIN e FACTORY. Note que os nomes dos problemas incluem o número de variáveis. Assim, o número correspondente de estados é $2^{\#variables}$	99
10.2	Tempo de execução de APRICODD-IP e OBJECTIVE-IP para os problemas TRAFFIC, SYSADMIN e FACTORY com $\delta = 0.1$	100
10.3	Comparação detalhada entre as aproximações realizadas por APRICODD-IP e OBJECTIVE-IP para o problema de tráfego com 10 variáveis. Variando o valor de δ , i.e., resolvemos um único problema variando o limite de erro dado por $\delta \cdot Vmax$	101
10.4	O número de restrições para o domínio do Administrador de Sistemas com topologia de anel unidirecional: (1) o número original de restrições; (2) as restrições após aplicar o algoritmo FACTOREDMPA com funções base simples; e (3) com funções base pares.	102
10.5	Tempo de execução do algoritmo FACTOREDMPA usando funções base simples e pares, e a solução exata para o domínio do Administrador de Sistemas com configuração Anel Unidirecional	103
10.6	Tempo de execução do algoritmo FACTOREDMPA usando funções base simples e pares, e a solução exata para o domínio do Administrador de Sistemas com configuração Estrela	103
10.7	Porcentagem do Erro de Aproximação Real de FACTOREDMPA para a configuração Anel Unidirecional do domínio do Administrador de Sistemas usando funções base pares e simples.	104

10.8	Percentagem do Erro de Aproximação Real de FACTOREDMPA para a configuração Estrela do domínio do Administrador de Sistemas usando funções base pares e simples.	105
10.9	Erro de Aproximação Real vs. tempo requerido por APRICODD-IP, OBJECTIVE-IP e FACTOREDMPA com funções base simples (FACTOREDMPA com funções base pares não terminou dado um tempo limite de 10 horas e com funções base simples não terminou para dois problemas) para três problemas FACTORY.	106
10.10	Erro de Aproximação Real vs. tempo requerido por APRICODD-IP, OBJECTIVE-IP e FACTOREDMPA com funções base simples e pares para o problema TRAFFIC. .	107
10.11	Erro de Aproximação Real vs. tempo requerido por APRICODD-IP, OBJECTIVE-IP e FACTOREDMPA com funções base simples e pares para o problema SYSADMIN com configuração Anel Unidirecional.	108
10.12	Erro de Aproximação Real vs. tempo requerido por APRICODD-IP, OBJECTIVE-IP e FACTOREDMPA com funções base simples e pares para o problema SYSADMIN com configuração Anel bidirecional.	109
10.13	Erro de Aproximação Real vs. tempo requerido por APRICODD-IP, OBJECTIVE-IP e FACTOREDMPA com funções base simples e pares para o problema SYSADMIN com configuração de Anéis bidirecionais independentes.	110
11.1	Exemplo de BMDP.	112
11.2	Exemplo de MDPST.	112
11.3	Exemplo de MDP-IP.	113
11.4	Exemplo de um MDP-IP e as suas distribuições <i>a priori</i>	115
11.5	Exemplo de diagrama de transição de hiper-estados em que o estado inicial é s_1 . . .	115

Lista de Tabelas

6.1	Casos de entrada e resultados do método COMPUTERESULT para as operações binárias $\oplus$, $\ominus$ e $\otimes$ para PADDs.	61
-----	---	----

Lista de Algoritmos

1	SOLVEMDP($\text{MDP}, \text{tol}, \text{maxIter}$)	41
2	REGRESS(V_{DD}, a)	41
3	REDUCEPADD(F)	58
4	GETNODE($\langle \text{var}, F_h, F_l \rangle$)	59
5	APPLY(F_1, F_2, op)	60
6	CHOOSEVARBRANCH(F_1, F_2)	61
7	SOLVE(MDP-IP, $\text{tol}, \text{maxIter}, \delta, \text{APP}, \text{Obj}$)	69
8	REGRESS($V_{DD}, a, \text{error}, \text{Obj}$)	70
9	APPROXADD($\text{value}_{DD}^i, \text{error}$)	70
10	APPROXPADDLEAVES(DD, error)	72
11	COMPUTEC $_i^a$ ($MDPIP, H, a$)	79
12	FACTOREDMP($\mathcal{C}^a, \mathcal{R}^a, O$)	82
13	GENERATEEQUALITYCONSFORREWARD($\mathbb{J}, \mathcal{R}^a, \mathbb{F}$)	82
14	GENERATEEQUALITYCONSFORC $_i$ ($\mathbb{J}, \mathcal{C}^a, \mathbb{F}$)	83
15	GENERATEINEQUALITYCONS($\mathbb{J}, \mathbb{F}, O, n$)	84
16	FACTOREDMPA($MDPIP, H, O$)	85
17	RTDP(MDP, $I, G, \text{max_depth}, V_{u_0}$) [Sanner et al., 2009]	122
18	BRTDP(MDP, $I, G, \text{max_depth}, V_{u_0}, V_{l_0}, \tau$) [Sanner et al., 2009]	123
19	CHOOSENEXTSTATEBRTDP(s_0, s, a, τ)	123

Capítulo 1

Introdução

O Processo de Decisão Markoviano (MDP) [Puterman, 1994] tornou-se *de fato* o modelo padrão para tomada de decisões sequencial em problemas de planejamento probabilístico, isto é, problemas de tomada de decisão sequencial. Um MDP modela a interação entre um agente e seu ambiente: em cada fase, o agente decide executar uma ação (com efeitos probabilísticos), que leva o agente a um estado futuro, e ganha uma recompensa. O objetivo do agente é maximizar a recompensa esperada sobre uma sequência de ações.

Tradicionalmente, MDPs assumem uma representação enumerativa de todos os estados. Uma representação mais compacta para MDPs usa uma representação fatorada do estado usando múltiplas variáveis de estado — os chamados MDPs fatorados. Tais representações sofrem da *maldição de dimensionalidade de Bellman* [Bellman, 1957]: o tamanho do espaço de estados cresce exponencialmente com o número de variáveis de estado. Soluções recentes exatas e aproximadas para MDPs fatorados exploram a representação fatorada do domínio [Guestrin, 2003, Hoey et al., 1999, St-aubin et al., 2000], de modo a resolver problemas com ordens de magnitude maiores do que as soluções usando abordagens clássicas enumerativas de programação dinâmica. Por exemplo:

- SPUD [Hoey et al., 1999] estende o algoritmo clássico de programação dinâmica ITERAÇÃO DE VALOR para explorar a estrutura do domínio usando Redes Bayesianas Dinâmicas (*DBN - Dynamic Bayesian Network*) (modelo gráfico para representar independências probabilísticas) e Diagramas de Decisão Algébrica (estrutura de dados para representar funções $\{0, 1\}^n \rightarrow \mathbb{R}$);
- APRICODD [St-aubin et al., 2000] é uma versão aproximada de SPUD com garantias de limite de erro;
- FACTORED LPA [Guestrin, 2003] usa uma combinação linear de funções base para aproximar a função valor de estado e explorar a estrutura aditiva das funções base para resolver o problema de forma eficiente.

Apesar desses recentes avanços para representar problemas do mundo real em que é simplesmente impossível obter uma representação precisa das probabilidades de transição, os Processos de Decisão Markovianos tornam-se insuficientes. Isto pode ocorrer por vários motivos: (a) informação imprecisa ou conflitante de especialistas nas medidas de probabilidades, (b) existência de dados

insuficientes usados para estimar o modelo preciso de transição de estados ou (c) as probabilidades de transição são não-estacionárias devido à falta de informação sobre a dinâmica do processo.

Por exemplo, em um MDP para controle de semáforos, é difícil estimar as probabilidades de um veículo prosseguir na mesma pista ou virar (para direita ou esquerda). Essas probabilidades podem mudar durante o dia ou ao longo do ano em função do tráfego em outras intersecções, e com base nos feriados e eventos especiais. Em geral, é impossível modelar precisamente todas essas dependências. Neste caso, o ideal seria ter uma política de controle de tráfego otimizada ao longo de um intervalo de probabilidades para virar, a fim de permitir um tratamento robusto para as características não-estacionárias nas probabilidades de um MDP.

Exemplo 1.1. TRAFFIC

O domínio TRAFFIC [Delgado et al., 2009c] é motivado por um problema real de controle de tráfego (através do controle de semáforos) em pistas de sentido duplo, como ilustrado na Figura 1.1¹. Suponha que no exemplo da figura queremos controlar somente a pista vertical (com dois sentidos: norte e sul). Existem dois semáforos, um para cada sentido da pista, que assumiremos ter somente dois estados: verde ou vermelho. Cada pista pode ser vista como uma fila: veículos no início da fila podem continuar na mesma pista, virar para direita ou para esquerda; veículos que não estão no início da fila podem avançar para uma posição à sua frente, sempre que esta estiver desocupada. As condições que devem ser satisfeitas para que um veículo (por exemplo, v_1) no início da fila possa avançar, virar para direita ou esquerda, dependem de dois fatores: do estado dos semáforos c_1 e c_2 , bem como da intenção do veículo no início da pista de sentido oposto (v_2). Na Figura 1.1, v_1 pode avançar ou virar para esquerda sempre que o semáforo c_1 estiver verde. No entanto, v_1 só pode virar para direita (manobra que cruza a pista no sentido oposto) nas seguintes situações: (i) se o semáforo c_1 estiver verde e o semáforo c_2 estiver vermelho; (ii) se c_1 e c_2 estiverem verdes e v_2 também pretende virar para a sua direita. Essa última condição é considerada causadora de aumento de tráfego nos cruzamentos. Nesse domínio estamos interessados em resolver o seguinte problema: como controlar as mudanças dos semáforos, levando-se em conta as probabilidades de veículos efetuarem manobras cruzando a pista oposta (no exemplo, virar para direita) e a probabilidade de existirem muitos veículos em cada pista (isto é, de chegarem muitos veículos no final das filas)? Na prática, é difícil estimar probabilidades exatas do veículo v_2 prosseguir na mesma pista ou virar e de novos veículos chegarem no final das filas. Assim, para resolver esse problema de maneira robusta, devemos considerar que essas probabilidades sejam imprecisas. O objetivo é reduzir o congestionamento, o que é equivalente a minimizar a quantidade de veículos nas filas. $\diamond$

Exemplo 1.2. SYSADMIN

O domínio SYSADMIN [Guestrin et al., 2003] é motivado por um problema hipotético de um administrador que gerencia uma rede de computadores, em que existem n computadores, $c_1, \dots, c_n$, conectados através de diferentes topologias, por exemplo, as máquinas podem estar conetadas em

¹Note que na Figura 1.1 os sentidos das pistas duplas seguem o sistema australiano de trânsito, uma vez que esse domínio faz parte de um projeto para cidade de Sydney, Austrália (STAR, NICTA).

Figura 1.1: Problema TRAFFIC.

anel unidirecional (Figura 1.2). Em cada estado um computador pode estar funcionando ou não e em cada instante o administrador pode decidir reinicializar um computador c_i , o que causará que c_i funcione no próximo instante (além de outros computadores conectados a ele, com uma dada probabilidade). Se um computador não for reinicializado, a sua probabilidade de funcionar (ou não) no próximo instante depende do seu estado atual e do número de computadores conectados a ele, que também estão funcionando (ou não) no estado atual. O administrador recebe um pagamento (recompensa) se todos os computadores estiverem funcionando. Portanto, o trabalho do administrador é decidir qual das máquinas reinicializar de modo a obter a maior recompensa. Assim, existem $n+1$ possíveis ações: reinicializar uma das n máquinas ou não reinicializar nenhuma. Uma política ótima neste problema irá reiniciar o computador que tenha o maior impacto na recompensa esperada.

Dado que neste domínio existem eventos exógenos difíceis de serem modelados, que fazem com que o computador deixe de funcionar ou se reinicialize em cada instante e dada a dificuldade de se modelar as probabilidades de transição para determinadas configurações da rede, é mais apropriado considerar probabilidades imprecisas para problemas nesse domínio, a fim de obter soluções mais robustas. $\diamond$

Para acomodar melhor os modelos de decisão sequencial na presença de probabilidades imprecisas, foi proposto o MDP com Probabilidades de Transição Imprecisas (MDP-IP) [Satia and Lave

Jr., 1973, White III and El-Deib, 1994]. Este arcabouço matemático fornece uma estrutura robusta para aplicações no mundo real de planejamento probabilístico, porém, uma solução geral exige o uso de rotinas de otimização computacionalmente caras que são extremamente demoradas na prática.

O MDP-IP foi formalmente proposto na década de 70 por Satia e Lave [Satia and Lave Jr., 1973] com o nome de *MDP com incerteza nas probabilidades de transição*. Satia e Lave propuseram dois algoritmos baseados em programação dinâmica para MDP-IPs: ITERAÇÃO DE VALOR e ITERAÇÃO DE POLÍTICA. Dada a complexidade do problema, os algoritmos usam rotinas de otimização em cada iteração dos algoritmos, e por isso, eles conseguem resolver apenas problemas muito pequenos.

No período entre a década de 70 e 90 não foram propostos trabalhos diretamente relacionados à melhoria de desempenho das soluções para MDP-IP propostos por Satia e Lave. Foi somente na década de 90 que White e El-Deib [White III and El-Deib, 1994] propuseram uma modificação no algoritmo de ITERAÇÃO DE POLÍTICA de Satia e Lave, usando a técnica de aproximações sucessivas. No entanto, o algoritmo ITERAÇÃO DE POLÍTICA modificado por White e El-Deib só consegue resolver problemas muito pequenos.

A primeira solução eficiente para uma subclasse de MDP-IPs, chamada de *Bounded-parameter Markov Decision Process* (BMDP) [Givan et al., 2000], foi proposta no final da década de 90. Num BMDP, as probabilidades e as recompensas são especificadas por intervalos de probabilidade. Um outro modelo foi proposto em 2007 por Trevizan et al [Trevizan et al., 2007], chamada de *Markov Decision Process with Set-valued Transitions* (MDPSTs) [Trevizan et al., 2007], em que são dadas distribuições de probabilidade sobre conjuntos finitos de estados, o que caracteriza uma subclasse de MDP-IP diferente daquela definida pelo BMDP. As soluções para BMDPs e MDPSTs reduzem a complexidade das soluções de Satia para MDP-IP. Porém, elas não podem ser usadas para resolver problemas gerais de MDP-IPs.

Nilim e El Ghaoui [Nilim and El Ghaoui, 2005], em 2005, retomaram o estudo teórico sobre MDP-IPs na área de controle robusto gerando resultados teóricos importantes para MDP-IPs. No entanto, a solução proposta resolve MDP-IPs em que a imprecisão nas probabilidades é modelada

Figura 1.2: Problema SYSADMIN com 7 computadores conectados na forma de anel unidirecional. O administrador da rede deve decidir qual máquina reinicializar em cada instante, sendo que existem eventos exógenos que fazem com que o computador deixe de funcionar ou passe a funcionar (representado pela fada na figura).

por conjuntos de incerteza com características particulares (por exemplo, com incertezas definidas sobre determinadas regiões de vizinhança) o que também caracteriza uma solução particular para uma subclasse de problemas modelados como MDP-IPs.

1.1 Objetivo

Da discussão anterior podemos concluir que existem poucos trabalhos na literatura relacionados a MDPs com probabilidades imprecisas. Uma característica comum a todas essas soluções é que todas trabalham com um espaço de estados enumerativo e, devido à sua complexidade computacional, apenas resolvem problemas muito pequenos. Assim, o objetivo principal desta tese é propor soluções eficientes para MDP-IPs para problemas de planejamento que envolvam milhões de estados. Para isso, exploraremos a representação fatorada de MDP-IPs em duas principais abordagens: (i) baseada em programação dinâmica e (ii) baseada em programação matemática.

1.2 Principais Contribuições dessa Tese

Dado o objetivo de encontrar soluções eficientes para MDP-IPs, esse trabalho propõe um conjunto de contribuições para as áreas de Inteligência Artificial, Pesquisa Operacional e Controle Ótimo, a saber:

1. **Representação fatorada de um MDP-IP:** Em [Delgado et al., 2008] propomos substituir as habituais Redes Bayesianas Dinâmicas (DBNs - Dynamic Bayes Nets) [Dean and Kanazawa, 1990] utilizadas em MDPs fatorados, por Redes Credais Dinâmicas (DCNs - Dynamic Credal Nets) [Cozman, 2000], para representar a imprecisão nas distribuições de MDP-IPs fatorados.
2. **Uma estrutura de dados mais adequada para representar funções num MDP-IP fatorado:** Em [Delgado et al., 2009c] estendemos o Diagrama de Decisão Algébrica (ADD - Algebraic Decision Diagram) [Bahar et al., 1993], que representa de forma compacta funções $\{0, 1\}^n \rightarrow \mathbb{R}$, que é usado pelos algoritmos SPUDD e APRICODD para resolver MDPs complexos [Hoey et al., 1999, St-aubin et al., 2000]. Introduzimos uma nova extensão de ADD, chamado de *ADD Parametrizado* (PADD), que contém expressões polinomiais nas suas folhas e mostramos como estender as operações de ADDs para PADDs. A definição dessa nova estrutura permitiu que fosse possível explorar a estrutura do domínio representado por uma DCN, e assim realizar cálculos de forma mais eficiente.
3. **Soluções baseadas em Programação Dinâmica para MDP-IP Fatorado:** Em [Delgado et al., 2009c] propomos um conjunto de algoritmos originais e eficientes que generalizam SPUDD [Hoey et al., 1999] e APRICODD [St-aubin et al., 2000] para MDP-IPs, usando PADDs. Esta generalização é apenas o primeiro passo na obtenção de soluções eficientes. Com base nas primeiras extensões, observamos que o principal gargalo computacional na solução de MDP-IPs é a necessidade de, repetidamente, resolver problemas de otimização não-linear. Mostramos então como direcionar nossas aproximações para reduzir drasticamente a sobre-

carga computacional do otimizador não-linear, enquanto produzimos soluções aproximadas com erro limitado. Este novo algoritmo é chamado OBJECTIVE-IP.

4. **Programação Multilinear Aproximada para MDP-IP Fatorado:** Em [Delgado et al., 2009a] propomos uma formulação para MDP-IPs fatorados baseada numa formulação multilinear para MDP-IPs [Shirota et al., 2007]. Além disso, propomos um algoritmo original, chamado FACTOREDMPA, que encontra políticas maximin para MDP-IPs fatorados. O algoritmo é uma adaptação do algoritmo FACTOREDLP (Factored Linear Programming-based Approximation) usado para resolver MDPs fatorados [Guestrin et al., 2003]. Nossos experimentos mostram que, explorando a representação fatorada de um problema de decisão sequencial e usando um conjunto de funções base para construir uma aproximação da função valor ótima, podemos resolver problemas grandes com solução aproximada de modo eficiente. Funções base são fornecidas por especialistas do domínio ou podem ser geradas automaticamente. Caso isso não seja possível, podemos utilizar as soluções alternativas baseadas em programação dinâmica descritas acima.

Com base na pesquisa realizada sobre trabalhos correlatos, este é o primeiro trabalho na literatura sobre MDP-IPs que mostra resultados experimentais para problemas que envolvem um grande número de estados. Por exemplo, para $2^{22} \sim 4$ milhões de estados foi possível fornecer uma solução exata e uma solução aproximada com garantias de limite de erro. Assim, acreditamos que esta é uma contribuição significativa para as comunidades de planejamento e pesquisa operacional, bem como para domínios específicos em que políticas robustas são necessárias.

1.3 Organização

O restante desta tese está organizado da seguinte forma:

- Na Parte I fornecemos os fundamentos que são usados ao longo desta tese. Começamos no Capítulo 2 oferecendo uma breve revisão dos conceitos básicos de probabilidade. Em seguida, mostramos os conceitos básicos sobre Processos de Decisão Markovianos e apresentamos uma variedade de soluções para MDPs no Capítulo 3. No Capítulo 4 definimos um MDP fatorado. Mostramos como a estrutura do MDP fatorado pode ser explorada de modo a evitar a enumeração de estados e apresentamos as idéias principais dos algoritmos para MDPs fatorados da literatura, relevantes para esta tese. Finalmente, no Capítulo 5, descrevemos a teoria de MDP-IPs enumerativos e os principais algoritmos existentes na literatura.
- Na Parte II mostramos as principais contribuições dessa tese. No Capítulo 6 introduzimos a representação fatorada para MDP-IPs baseada em *Redes Credais Dinâmicas* (Dynamic Credal Networks - DCNs) [Cozman, 2000, Cozman, 2005a]. Adicionalmente, introduzimos ADDs Parametrizados (PADDs) que permitem cálculos eficientes durante a ITERAÇÃO DE VALOR, como veremos no Capítulo 7. Além disso, no Capítulo 7 mostramos nosso conjunto de algoritmos originais baseados em ITERAÇÃO DE VALOR para resolver MDP-IP fatorados (soluções

exatas e aproximadas). No Capítulo 8 apresentamos um algoritmo original aproximado formulado como um programa multilinear.

- Na Parte III apresentamos os domínios de teste e os resultados experimentais envolvendo o conjunto de algoritmos propostos nessa tese, na resolução de problemas envolvendo tamanhos grandes de espaço de estados (até 2^{22} estados) e finalmente, apresentamos os trabalhos correlatos, as conclusões da tese e nossas propostas de trabalhos futuros.

Parte I

Fundamentos

Capítulo 2

Probabilidades e Planejamento sob Incerteza

Neste capítulo, discutiremos alguns conceitos sobre probabilidades que serão usados na tese entre eles evento e variável aleatória. Além disso, veremos modelos gráficos usados para representar problemas com uma grande quantidade de variáveis aleatórias e as relações de independência entre elas. Definimos a tarefa de planejamento e mostramos os diferentes tipos de planejamento.

2.1 Probabilidades: Conceitos Básicos

Alguns conceitos básicos sobre medidas de probabilidade [Cozman, 2005b] relacionados à ocorrência de eventos e aos valores que uma variável aleatória pode assumir são introduzidos a seguir.

2.1.1 Espaço amostral e eventos

Chamamos de espaço amostral o conjunto não vazio de todos os possíveis resultados de um experimento aleatório, denotado por Ω . Os elementos $\omega \in \Omega$ são os pontos amostrais e apenas um deles ocorre num dado experimento. Neste trabalho, será considerado Ω finito e enumerável.

Outro conceito importante é o evento, que é um subconjunto de Ω . Um evento $A \subseteq \Omega$ ocorre quando um ponto amostral $\omega \in A$ ocorre. Dizemos que dois eventos A e B são mutuamente exclusivos se $A \cap B = \emptyset$. Um espaço de eventos é o conjunto de todos os subconjuntos de Ω , denotado por $\mathfrak{E}$.

2.1.2 Medidas de probabilidade

Uma medida de probabilidade é uma função P que associa um valor, entre 0 e 1, para um evento A . Uma medida de probabilidade deve satisfazer os seguintes axiomas:

- $P(\Omega) = 1.$ (2.1)

- $P(A) \geq 0, \forall A \in \mathfrak{E}.$ (2.2)

- $P(A \cup B) = P(A) + P(B)$ se $A \cap B = \emptyset.$ (2.3)

Algumas consequências destes axiomas são:

- $P(A) = 1 - P(A^c)$, em que A^c é o complemento do evento A .

- $P(\emptyset) = 0.$

- $P(A \cup B) = P(A) + P(B) - P(A \cap B) \forall A, B \in \mathfrak{E}$.

2.1.3 Origem das probabilidades

Existem dois principais pontos de vista sobre a origem fundamental da probabilidade: a visão *frequentista* e a visão *Bayesiana* [Magalhães and de Lima, 2008].

Frequentistas argumentam que as probabilidades vêm de experimentos. A probabilidade de um evento aleatório indica a frequência relativa de ocorrência do resultado de um experimento, realizado diversas vezes. Assim, a visão frequentista não pode lidar com eventos que acontecem apenas uma vez. Para os Bayesianos a probabilidade é uma medida da crença, ou ignorância, da pessoa que as especifica, ou seja, é uma forma de representar a opinião de um especialista. Nesta tese, consideramos a interpretação Bayesiana sobre as probabilidades.

2.1.4 Probabilidade condicional e probabilidade conjunta

Às vezes, é importante calcular a probabilidade de um evento A , dada a ocorrência de algum outro evento B , chamada de probabilidade condicional de A dado B , denotada por $P(A|B)$ e definida por:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}, \quad (2.4)$$

em que o primeiro evento A , o *evento condicionado*, é um subconjunto de Ω enquanto que o segundo B , o *evento condicionante*, deve ser um subconjunto não vazio de Ω .

$P(A \cap B)$ é chamada também de probabilidade conjunta, ou seja, a probabilidade de dois eventos ocorrerem em conjunto, que também pode ser denotada por $P(A, B)$. Da Equação (2.4) obtemos:

$$P(A \cap B) = P(A|B)P(B). \quad (2.5)$$

Generalizando a Equação (2.5) para n eventos, obtemos:

$$P(B_1, B_2, \dots, B_n) = P(B_1)P(B_2|B_1) \dots P(B_n|B_{n-1}, \dots, B_2, B_1). \quad (2.6)$$

Outro conceito importante é eventos independentes. Dois eventos são independentes se $P(A \cap B) = P(A)P(B)$ e se $P(B) > 0$ na Equação (2.4) obtemos:

$$P(A|B) = P(A). \quad (2.7)$$

2.1.5 Teorema de Bayes

Da definição de probabilidade condicional (Equação (2.4)) e da propriedade comutativa de conjuntos $A \cap B = B \cap A$, podemos obter o Teorema de Bayes:

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)}, \quad (2.8)$$

em que $P(A)$ e $P(A|B)$ são conhecidas como probabilidade *a priori* e *posteriori* respectivamente.

2.1.6 Variáveis aleatórias

Uma variável aleatória é uma função $X : \Omega \rightarrow \mathbb{R}$ e pode ser usada para definir eventos interessantes de Ω . Por exemplo, a função X que atribui a idade de uma pessoa ω selecionada de uma população Ω é uma variável aleatória. Podemos definir um evento B como: $\{\omega \mid 60 < X(\omega) < 100\}$, para definir o evento de selecionarmos de um espaço amostral, pessoas entre 60 e 100 anos de idade. Se na população só existem pessoas com menos de 50 anos, o evento B seria um conjunto vazio.

Uma variável aleatória pode ser discreta ou contínua. Se os valores que a variável pode assumir pertencem a um conjunto enumerável, a variável é chamada de discreta, caso contrário, ela é chamada de contínua. Neste trabalho estamos interessados em variáveis discretas.

2.1.7 Função de densidade de probabilidade

Podemos associar uma probabilidade de ocorrência a cada valor que a variável aleatória pode assumir. A função que atribui a cada valor x_i da variável aleatória X uma probabilidade é chamada de *função de densidade* e é denotada por $P(X = x_i) = P(x_i)$. Para variáveis discretas, uma função de densidade de probabilidade satisfaz:

- $0 \leq P(X = x_i) \leq 1$.
- $\sum_i P(X = x_i) = 1$.

Por exemplo, para a variável $X = \{x_1, x_2, x_3\}$, $P(X)$ é definida por $P(X = x_1) = 0.3$, $P(X = x_2) = 0.2$ e $P(X = x_3) = 0.5$ ou simplesmente $P(x_1) = 0.3$, $P(x_2) = 0.2$ e $P(x_3) = 0.5$.

2.1.8 Densidade de probabilidade conjunta e marginalização

A probabilidade conjunta quando X e Y são variáveis aleatórias é chamada de *densidade de probabilidade conjunta* e é denotada por $P(X = x, Y = y)$. Às vezes, estamos interessados na probabilidade de ocorrência de valores de um subconjunto de variáveis aleatórias na probabilidade conjunta. O processo de calcular esta probabilidade é chamado de **marginalização** e envolve fazer a soma das probabilidades sobre os valores das variáveis aleatórias que não são de interesse. Por exemplo, para duas variáveis X e Y , se estamos interessados em determinar $P(X = x)$ e conhecemos $P(X = x, Y = y)$, podemos aplicar a marginalização $\sum_y$ para eliminar a variável y , i.e.:

$$P(X = x) = \sum_y P(X = x, Y = y). \quad (2.9)$$

Na Figura 2.1 mostramos a densidade de probabilidade conjunta $P(X_1, X_2)$ e a marginalização sobre X_2 . Para obter, por exemplo, a probabilidade $P(X_1 = 1)$, somamos as probabilidades $P(X_1 = 1, X_2 = 1)$ e $P(X_1 = 1, X_2 = 0)$, que é igual a 0.6.

A marginalização é usada em cálculos complexos de probabilidades conjuntas, eliminando variáveis da expressão. Por exemplo, sejam $X_1, X_2, \dots, X_n$ variáveis binárias aleatórias e $x_1, x_2, \dots, x_n$ seus valores, respectivamente. Se desejamos calcular a soma (uma soma de 2^n termos):

x_1	x_2	$P(x_1, x_2)$
1	1	0.4
1	0	0.2
0	1	0.3
0	0	0.1

x_1	$\sum_{x_2} P$
1	0.6
0	0.4

Figura 2.1: Densidade de probabilidade conjunta $P(X_1, X_2)$ e o resultado de aplicar a marginalização sobre X_2 .

$$\sum_{x_1, x_2, \dots, x_n} P(X_1, X_2, \dots, X_n) f(X_1, X_2, \dots, X_n) = \sum_{x_1, x_2, \dots, x_n} \prod_{i=1}^n P(X_i | \dots) f(X_1, X_2, \dots, X_n) \quad (2.10)$$

e se sabemos que nenhuma outra variável nessa expressão depende de X_n e se X_n não depende de qualquer outra variável, para eliminar essa variável, podemos deslocar a soma sobre x_n para dentro da produtória e obter:

$$\sum_{x_1, x_2, \dots, x_n} \prod_{i=1}^n P(X_i | \dots) f(X_1, X_2, \dots, X_n) = \sum_{x_1, x_2, \dots, x_{n-1}} \prod_{i=1}^{n-1} P(X_i | \dots) \sum_{x_n} P(X_n) f(X_1, X_2, \dots, X_n), \quad (2.11)$$

fazendo a marginalização $\sum_{x_n}$, podemos reduzir a expressão e aplicar a mesma ideia para outras variáveis independentes, uma a uma, simplificando assim o cálculo de expressões complexas.

2.1.9 Esperança

Dada uma função de probabilidade $P(\cdot)$ e uma variável aleatória discreta X , a esperança de X , denotada por $E[X]$, é a soma ponderada dos valores que a variável pode assumir, em que os pesos são obtidos a partir da função de densidade de probabilidade:

$$E[X] = \sum_{x_i} x_i P(X = x_i). \quad (2.12)$$

A esperança tem as seguintes propriedades:

- Para as constantes α e β , se $\alpha \leq X \leq \beta$, então $\alpha \leq E[X] \leq \beta$.
- Para qualquer constante λ , $E[\lambda X] = \lambda E[X]$.
- $E[X + Y] = E[X] + E[Y]$.

2.1.10 Conjunto de medidas de probabilidade

Especificar um valor preciso de probabilidade para cada evento ou para cada ocorrência dos valores de uma variável aleatória parece ser fácil, no entanto, em alguns casos, essa é uma tarefa

Figura 2.2: Exemplo de conjunto credal válido representado pela região cinza. O conjunto credal é definido pelas triplas $\{P(x_1), P(x_2), P(x_3)\}$ que pertencem a essa região.

difícil ou mesmo impossível. Uma forma de contornar essa dificuldade é especificar a probabilidade, usando um conjunto de restrições (também chamadas de *assessments*). O conjunto de restrições define um conjunto de medidas de probabilidade que é chamado de **conjunto credal** [Cozman, 2000], e denotado por K .

Por exemplo, sejam x_1, x_2 e x_3 os valores da variável X e a função de densidade de probabilidade $P(X)$ dada pelas probabilidades $P(x_1)$, $P(x_2)$ e $P(x_3)$. Seja o conjunto de medidas de probabilidade válidas definido pelo conjunto de restrições a seguir: $P(x_1) \leq 2/3$, $P(x_3) \leq 2/3$, $2 * P(x_1) \geq P(x_2)$ e $P(x_1) + P(x_2) + P(x_3) = 1$. Na Figura 2.2, mostramos a região de todas as medidas de probabilidade que satisfazem as restrições (região cinza). Este conjunto de medidas de probabilidade é um *conjunto credal*.

2.2 Modelos Gráficos para Inferência

No mundo real precisamos representar problemas com uma grande quantidade de variáveis aleatórias com relações de dependência. Para representar explicitamente tais relacionamentos, foram propostos modelos baseados em grafos. A seguir mostramos dois modelos gráficos: Redes Bayesianas e Redes Credais.

2.2.1 Redes Bayesianas

Uma Rede Bayesiana [Pearl, 1988] representa uma única distribuição de probabilidade conjunta sobre um conjunto de variáveis aleatórias. É um grafo dirigido acíclico que representa a dependência condicional entre as variáveis. Um nó representa uma variável aleatória e as arestas representam dependências condicionais. Uma Rede Bayesiana satisfaz a propriedade de Markov: cada variável é independente de seus não-descendentes não-pais, dados seus pais. Essa propriedade faz com que os cálculos envolvendo as probabilidades sejam simplificados. Assim, a probabilidade conjunta dada pela Equação (2.6), com $pa(X_i)$ sendo os pais de X_i no grafo, pode ser fatorada como:

Figura 2.3: Rede Bayesiana para o exemplo do alarme. As tabelas indicam as probabilidades a priori e as probabilidades condicionais.

$$P(X_1, \dots, X_n) = \prod_i P(X_i | pa(X_i)). \quad (2.13)$$

Na Figura 2.3 mostramos a Rede Bayesiana para o exemplo do alarme [Pearl, 1988, Russell and Norvig, 2002]: um novo alarme contra roubo foi instalado na casa de David que também é acionado por pequenos tremores de terra. Há dois vizinhos, John e Mary que prometeram ligar para David no trabalho, caso eles ouçam o alarme. Note que cada nó está associado a uma *tabela de probabilidade condicional* (CPT - *Conditional Probability Table*) e cada linha em uma CPT contém a probabilidade condicional do valor de cada nó para uma combinação de valores dos nós pais. Assim, especificamos as probabilidades sobre os valores da variável, dada uma combinação de valores dos pais da variável. As variáveis aleatórias usadas são variáveis booleanas, dadas por A (Alarme), B (Roubo), E (Tremor), J (Ligação de John) e M (Ligação de Mary).

Um conjunto de observações de uma Rede Bayesiana é chamado de *evidência* da rede. Por exemplo, a Rede Bayesiana da Figura 2.3, que representa as dependências entre as variáveis, pode ser usada para obter as combinações de valores das variáveis que maximizam algum valor ou realizar inferência probabilística, isto é, calcular a probabilidade *a posteriori* dadas as evidências. No exemplo, poderíamos desejar calcular a probabilidade de ter ocorrido um roubo, dadas as evidências de que houve uma ligação de John e uma ligação de Mary, i.e., $P(B = 1 | J = 1, M = 1)$. A inferência probabilística numa Rede Bayesiana é computacionalmente intratável no pior caso, mas pode ser realizada de maneira eficiente em muitas situações práticas [Russell and Norvig, 2002]. Existem algoritmos de inferência exata e aproximada para Redes Bayesianas, entre eles, o algoritmo de *Eliminação de Variáveis* que elimina cálculos repetidos na soma de produtos de probabilidades condicionais através da operação de marginalização [Russell and Norvig, 2002].

Uma classe especial de Redes Bayesianas que possui apenas duas camadas é chamada de *Rede*

Figura 2.4: Rede Credal para o exemplo do alarme.

Bayesiana Dinâmica, e é usada para representar MDPs, como veremos no Capítulo 4.

2.2.2 Redes Credais

Redes Credais [Cozman, 2005a] são grafos dirigidos acíclicos, em que cada nó está associado a conjuntos de medidas de probabilidade. Uma Rede Credal pode ser vista como uma coleção de Redes Bayesianas sobre um conjunto fixo de variáveis aleatórias. Para cada variável e para cada combinação de valores dos pais, podemos especificar um **conjunto de medidas de probabilidade**. Assumimos que a Rede Credal representa um conjunto de distribuições de probabilidade conjunta sobre as variáveis aleatórias, em que **cada** distribuição satisfaz a seguinte fatoração:

$$P(X_1, \dots, X_n) = \prod_i P(X_i | pa(X_i)). \quad (2.14)$$

A vantagem das Redes Credais é que elas permitem a representação de problemas mais próximos do mundo real [Cozman, 2005a]. Na Figura 2.4 mostramos a Rede Credal para o exemplo do alarme com CPTs parametrizados e com um conjunto de restrições sobre os parâmetros dos CPTs, definindo os valores válidos para as probabilidades de transição. Observe que cada linha na CPT pode conter um conjunto de medidas de probabilidade. Por exemplo, as restrições que definem

os valores válidos para o parâmetro p_{41} são: a) a probabilidade de Mary ligar para David quando o alarme tocar é menor do que a probabilidade de John ligar para David quando o alarme tocar ($p_{41} \leq p_{31}$) e b) a probabilidade de John ligar para David quando o alarme tocar está entre 0,7 e 0,9 ($0.7 \leq p_{31} \leq 0.9$), uma vez que p_{41} depende do valor de p_{31} .

2.3 Tomada de Decisão Sequencial e Planejamento

A teoria de probabilidade está relacionada com a teoria de decisão [Berger, 1985]. Um tomador de decisões tem um conjunto de ações a serem consideradas, cada uma produz recompensas diferentes em diferentes estados.

Preferências completas sobre as ações implica que um valor preciso de probabilidade seja associado com cada estado, essa situação é chamada de risco [Knight, 1921]. No entanto, às vezes as preferências sobre as ações são apenas parcialmente ordenadas e então não é possível definir um valor preciso de probabilidade. Nesse caso um conjunto de medidas de probabilidade (conjunto credal) é uma representação adequada da incerteza. Essa situação é chamada de incerteza Knightiana, ambigüidade ou simplesmente incerteza [Knight, 1921].

A habilidade de planejar tarefas é um aspecto fundamental do comportamento inteligente sendo que, há mais de quarenta anos [McCarthy, 1968, Newell and Simon, 1963], sua automação tem sido um dos principais objetivos da pesquisa realizada na área de *Inteligência Artificial* [Russell and Norvig, 2002]. Em planejamento, a forma com que um sistema evolui a cada ação executada, chamada de *dinâmica das ações*, pode ser: determinística, não-determinística ou probabilística. Enquanto os modelos determinísticos não modelam incerteza no efeito das ações (executar uma ação em um determinado estado sempre leva o agente a um único estado) os modelos não-determinísticos e probabilísticos modelam diferentes formas de incerteza, isto é, a incerteza Knightiana e incerteza de risco [Trevizan et al., 2007], respectivamente.

Definição 2.1. *Um domínio de planejamento clássico é definido por uma tupla $\mathcal{D} = \langle S, A, \mathcal{F} \rangle$, sendo:*

- $S \neq \emptyset$, um conjunto finito de estados possíveis do ambiente (também chamado de estados do mundo);
- $A \neq \emptyset$, um conjunto finito de ações executáveis pelo agente e
- $\mathcal{F} : S \times A \mapsto S$, uma função de transição de estados (isto é, ações com efeitos determinísticos).

Definição 2.2. *Um problema de planejamento clássico é definido por uma tupla $\mathcal{P} = \langle \mathcal{D}, I, G \rangle$, sendo:*

- $\mathcal{D}$, um domínio de planejamento (Definição 2.1);
- $I \subseteq S$, um conjunto de estados iniciais do ambiente e
- $G \subseteq S$, um conjunto de estados metas.

A solução para um problema de planejamento clássico $\mathcal{P}$ é uma sequência de ações que, ao ser executada, leva o agente do estado inicial para o estado meta. Porém, em muitos domínios do mundo real, a suposição do planejamento clássico de que as ações possuem efeitos determinísticos se torna inadequada, uma vez que:

- A especificação do ambiente é incompleta devido à complexidade intrínseca do domínio de aplicação.
- Em geral, existem eventos exógenos sobre os quais o agente não tem controle e que podem interferir nos efeitos das ações executadas pelo agente.
- A aplicação de uma ação pode resultar em falha.
- Os efeitos das ações são intrínsecamente incertos.

Em domínios em que existe incerteza nos efeitos das ações, a solução de um problema de planejamento é uma política que associa uma ação para cada estado em o agente se encontra. As principais abordagens para resolver problemas com incerteza nos efeitos das ações são: (i) planejamento não-determinístico e (ii) planejamento probabilístico, conforme descrito a seguir.

2.3.1 Planejamento não-determinístico

No planejamento não-determinístico, a execução de uma ação pode levar o sistema para estados diferentes, sendo que o agente não tem nenhuma informação da preferência da Natureza sobre os estados sucessores, i.e., não são conhecidas as probabilidades de transição entre estados. Planejamento não-determinístico é um caso extremo de incerteza Knightiana em que nenhuma probabilidade é especificada.

Existem três tipos de soluções para planejamento não-determinístico:

- **Soluções fracas** são políticas que podem eventualmente atingir a meta, mas não garantem alcançá-las.
- **Soluções fortes** são políticas que garantem alcançar a meta, independentemente da incerteza, supondo que os caminhos para atingir a meta são finitos.
- **Soluções fortes cíclicas** são políticas que garantem atingir a meta porém num número indeterminado de passos.

A maioria das soluções para estes problemas se baseiam em técnicas de Verificação de Modelos (*Model Checking*) [Cimatti et al., 1997, Cimatti et al., 1998, Daniele et al., 1999, Pereira and de Barros, 2008], sendo que as fórmulas em lógica temporal expressam metas temporais com possíveis condições sobre os caminhos de execução.

2.3.2 Planejamento probabilístico

No planejamento probabilístico, assim como no planejamento não-determinístico, a execução de uma ação pode levar o agente para diferentes estados. No entanto, são conhecidas as probabilidades de transição e a solução é uma política que otimiza algum critério de custo e/ou recompensa esperada. Planejamento probabilístico é um exemplo de tomada de decisão sequencial sob risco “puro” [Trevizan et al., 2007].

A principal abordagem para resolver problemas de planejamento probabilístico é através da sua modelagem como um Processo de Decisão Markoviano (MDPs) [Howard, 1960, Bonet and Geffner, 2005], em que são feitas as suposições que a seleção da melhor ação pode ser tomada conhecendo-se apenas o estado atual do agente e o ambiente é completamente observável. No Capítulo 3 apresentamos este modelo em detalhes. As principais soluções para MDPs são baseadas em programação matemática [Manne, 1960] e programação dinâmica como *Iteração de Valor* [Bellman, 1957], *Iteração de Política* [Howard, 1960]), *Programação Dinâmica em Tempo Real (RTDP - Real Time Dynamic Programming)* [Barto et al., 1995] e algumas extensões.

2.3.3 Planejamento não-determinístico e probabilístico

Esforços para unificar planejamento não-determinístico e probabilístico são: *MDP delimitado* (BMDP - *Bounded-parameter Markov Decision Process*) [Givan et al., 2000], *MDP com transições para conjuntos valorados* (MDPSTs - *Markov Decision Process with Set-valued Transitions*) [Trevizan et al., 2007] e *MDP com probabilidades de transição imprecisas* (MDP-IP, *MDP with imprecise transition probabilities*) [Satia and Lave Jr., 1973, White III and El-Deib, 1994]. MDP-IPs, descritos no Capítulo 5, são os modelos estudados nessa tese. A relação entre MDP-IPs e os demais modelos de planejamento não-determinístico e probabilístico é discutida na Seção 5.2.

2.4 Resumo

Neste capítulo, introduzimos alguns conceitos básicos sobre probabilidades que serão usados nessa tese. Além disso, mostramos dois modelos gráficos, Redes Bayesianas e Redes Credais, que usaremos para representar problemas com uma grande quantidade de variáveis aleatórias e para representar as relações de independência entre elas.

Finalmente, definimos a tarefa de planejamento e mostramos os diferentes modelos de planejamento: determinístico (clássico), não determinístico e probabilístico. Enquanto os modelos determinísticos não modelam incerteza no efeito das ações, os modelos não-determinísticos e probabilísticos modelam diferentes formas de incerteza. A principal abordagem para modelar problemas de planejamento probabilístico usa Processos de Decisão Markovianos, que serão apresentados no Capítulo 3.

Capítulo 3

Processo de Decisão Markoviano

A principal abordagem para resolver um problema de planejamento probabilístico consiste em modelá-lo como um Processo de Decisão Markoviano (*Markov Decision Process - MDP*) [Howard, 1960, Bonet and Geffner, 2005]. Originalmente proposto pela comunidade de Pesquisa Operacional, um MDP [Puterman, 1994] fornece um arcabouço matemático elegante para representar e resolver problemas de decisão sequencial sob incerteza em ambientes completamente observáveis.

Um MDP modela a interação entre um agente e o seu ambiente: em cada estágio, o agente decide executar uma ação (com efeitos probabilísticos) que irá produzir um estado futuro, uma recompensa e um custo. O objetivo do agente MDP é maximizar a recompensa esperada adquirida ao longo de uma sequência de escolhas de ação.¹

A seguir, definimos formalmente um MDP e descrevemos vários algoritmos para resolvê-los.

3.1 MDP: Definição

Formalmente, um *Processo de Decisão Markoviano* (*Markov Decision Process - MDP*) é definido pela tupla $\mathcal{M} = \langle S, A, R, C, P, \gamma \rangle$ em que:

- S é um conjunto discreto e finito de estados completamente observáveis que modelam o mundo;
- A é um conjunto finito de ações; a execução de uma ação permite que o sistema mude do estado atual para o próximo estado. Assim, a tarefa de planejamento é escolher a melhor ação para ser executada num determinado estado;
- $R : S \times A \rightarrow \mathbb{R}$ é a função recompensa associada a cada par $\langle \text{estado}, \text{ação} \rangle$ e que representa as preferências do agente.
- $C : A \rightarrow \mathbb{R}$ é a função custo associada a cada ação e que representa o custo de aplicar a ação.
- P define a probabilidade de transição, sendo que $P(s'|s, a)$ é a probabilidade condicional de ir para o estado $s' \in S$ quando o agente está no estado $s \in S$ e executa a ação $a \in A$. Note que a função transição satisfaz a suposição de Markov, ou seja, a probabilidade de ir para o estado s' a partir de s depende unicamente de s , i.e., independe dos estados anteriores;

¹MDPs variam quanto aos seus objetivos; por exemplo, o objetivo do agente pode ser maximizar a recompensa menos o custo ou apenas minimizar o custo.

- γ é o fator de desconto, sendo $0 \leq \gamma < 1$; assim, depois de t estágios a recompensa (e/ou custo) é descontada por γ^t .

Um problema de planejamento pode ser modelado com um MDP. As principais diferenças entre um problema de planejamento e de MDP são:

- ao invés de recompensa define-se apenas o custo e
- são dados os conjuntos de estados iniciais $I \subseteq S$ e metas $G \subseteq S$.

Na literatura de MDPs, quando são dados I , G e $\gamma = 1$ o MDP passa a ser chamado de SSP (Stochastic Shortest Path), sendo um modelo mais adequado para modelar problemas de planejamento (como veremos na Seção 3.3.2). Porém, a solução de um MDP é mais geral, isto é, a solução ótima de um MDP contém a solução do SSP correspondente.

É importante notar que a solução para um problema MDP não é uma sequência fixa de ações como no planejamento clássico, uma vez que há incerteza sobre os efeitos das ações. Assim, o objetivo do agente é determinar a melhor ação para cada estado em que ele se encontra; esta solução é chamada de *política* e é denotada por π .

3.1.1 Avaliação da política

Para avaliar uma política precisamos especificar quanto tempo ela vai ser seguida pelo agente ao longo do horizonte, ou seja, o número de passos de decisão até o seu término. Assim, MDPs podem ser classificados por seu horizonte. Um MDP com *horizonte finito* pode ser visto como um problema em que o agente tem algum recurso finito, que limita a quantidade de ações que podem ser executadas. Em um MDP com *horizonte infinito* é impossível (ou não queremos) restringir o número de passos de decisão até o seu término. Ou seja, não há um prazo para concluir o processo.

Uma política pode ser *estacionária*, ou seja, depende do estado atual e independe do número de passos de decisão (que também chamamos de estágios) restantes. Assim, se temos uma política estacionária, a melhor ação a ser executada no estado s é sempre a mesma. Caso contrário, se uma política depende do número de passos de decisão restantes, a política é chamada de *não-estacionária*. Neste trabalho, estamos interessados em:

- políticas estacionárias,
- MDPs de horizonte infinito e
- MDPs com recompensa (isto é, sem considerar custo).

Uma política estacionária $\pi : S \rightarrow A$ especifica a ação $a = \pi(s)$ a ser escolhida em cada estado s . O objetivo do agente MDP é encontrar a política que maximiza a soma esperada da recompensa descontada num horizonte infinito, formalmente descrito pela expressão:

$$E_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 \right] \quad (3.1)$$

em que r_t é a recompensa obtida no estágio t e s_0 é o estado inicial. O valor da política π começando no estado s e executando π , denotado por $V_\pi(s)$, é definido como a soma esperada das recompensas descontadas sob um horizonte infinito, em que $s_0 = s$, isto é:

$$V_\pi(s) = E_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s \right] \quad (3.2)$$

Definimos uma política gulosa π_V com respeito a alguma função valor $V : S \rightarrow \mathbb{R}$, como a política que escolhe uma ação em cada estado que maximiza o valor esperado de um estado s com respeito a V , conforme definida a seguir:

$$\pi_V(s) = \text{GREEDYACTION}(V, s) \quad (3.3)$$

$$= \arg \max_{a \in A} (Q(s, a)) \quad (3.4)$$

$$= \arg \max_{a \in A} \left(R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V(s') \right) \quad (3.5)$$

onde $Q(s, a)$ é o valor do estado s aplicando a ação a . Isso significa que, conhecendo uma estimativa qualquer da função valor V para os próximos estados $s' \in S$, as probabilidades de transição $P(s'|s, a)$ e a recompensa imediata $R(s, a)$, podemos determinar a melhor ação com respeito a V escolhendo a ação que maximiza $Q(s, a)$ (i.e, podemos obter a política gulosa).

3.1.2 Política ótima

Nosso objetivo é encontrar uma política ótima π^* que produz o valor máximo em cada estado, i.e., $\forall s, \pi' V_{\pi^*}(s) \geq V_{\pi'}(s)$. Para o caso de horizonte infinito existe sempre uma política ótima estacionária [Howard, 1960].

A *função valor ótima*, representada por V^* , é a função valor associada com qualquer política ótima. Assim, para um agente que deseja maximizar sua recompensa esperada, V^* satisfaz a seguinte igualdade de ponto fixo [Bellman, 1957]:

$$V^*(s) = \max_{a \in A} \{ R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^*(s') \}. \quad (3.6)$$

A Equação (3.6) é conhecida como o *Princípio de Otimalidade de Bellman para MDPs* ou *Equação de Bellman*. Os principais algoritmos para encontrar V^* são baseados em Programação Dinâmica e Programação Matemática, conforme mostraremos a seguir.

3.2 A Complexidade de Resolver um MDP

O problema de encontrar uma política ótima para um MDP, com representação explícita dos estados (também chamado de MDP enumerativo), é P-Completo [Papadimitriou and Tsitsiklis, 1987], ou seja, é um problema que pode ser resolvido em tempo polinomial no pior caso e não

é provável que a sua complexidade possa ser reduzida através de computação paralela. A classe P-Completa contém os problemas mais difíceis em P. Apesar de MDPs poderem ser resolvidos em tempo polinomial no tamanho do espaço de estados, precisamos investigar algoritmos práticos para resolver problemas grandes envolvendo milhões de estados [Littman et al., 1995].

Note que uma representação mais natural de um MDP usa uma representação de estado fatorada em termos de variáveis de estado. Esta representação sofre da “maldição da dimensionalidade de Bellman” [Bellman, 1957] — o tamanho do espaço de estados cresce exponencialmente com o número de variáveis de estado. Para contornar a “maldição da dimensionalidade” é fundamental explorar o espaço de estados de uma forma eficiente, como é a proposta dessa tese. Antes disso, apresentamos nas próximas seções desse capítulo os algoritmos mais conhecidos para MDPs com representação explícita de estados.

3.3 MDP: Soluções Baseadas em Programação Dinâmica

Existem algoritmos clássicos para gerar políticas ótimas em MDPs baseados em programação dinâmica com garantias de convergência. Os algoritmos podem ser executados utilizando: atualizações *síncronas* do valor de estado estimado, isto é, em que todos os estados são atualizados em cada iteração; ou *assíncronas*, em que o valor de um estado é atualizado para alguns estados mais do que para outros e, ainda assim, mantendo-se as propriedades de convergência. A atualização da estimativa de $V(s)$ é chamada de *Atualização de Bellman* (*Bellman Backup*).

A seguir, discutimos dois algoritmos síncronos para resolver MDPs de horizonte infinito, ITERAÇÃO DE VALOR [Bellman, 1957] e ITERAÇÃO DE POLÍTICA [Howard, 1960]) e também apresentamos um método de programação dinâmica assíncrona, o algoritmo *Programação Dinâmica em Tempo Real* (RTDP) [Barto et al., 1995] e suas extensões.

3.3.1 Programação dinâmica síncrona

Um algoritmo bem conhecido para resolver um MDP é o algoritmo de ITERAÇÃO DE VALOR (IV) [Puterman, 1994]. Ele faz uma série de atualizações da função valor V^t (V^t pode ser interpretada como o valor que o agente obterá em t -estágios-para-frente (*t-stages-to-go*)). Começando com um V^0 arbitrário, IV executa atualizações de todos os estados s , calculando V^t baseado em V^{t-1} . Para compreendermos o algoritmo IV usamos a função $Q(s, a)$ (Equação (3.4)) que representa o valor do estado s quando a ação a é usada, dada por:

$$Q^t(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^{t-1}(s') \quad (3.7)$$

e o melhor valor possível no estágio de decisão t e estado s é:

$$V^t(s) = \max_{a \in A} Q^t(s, a). \quad (3.8)$$

O cálculo da função valor $V^t(s) = \text{BACKUP}(V^{t-1}, s)$ é a Atualização de Bellman. Na Figura 3.1 mostramos um exemplo de Atualização de Bellman no estágio t para o estado s_0 , isto é, $V^t(s_0)$.

Figura 3.1: Exemplo de Atualização de Bellman para s_0 : $V^t(s_0) = \max_{a \in A} Q^t(s_0, a)$.

Figura 3.2: O algoritmo de ITERAÇÃO DE VALOR executa Atualizações de Bellman para um MDP com 2 estados, s_1 e s_2 , e 3 iterações, calculando V^t baseado em V^{t-1} .

Dadas as estimativas V^{t-1} para os estados sucessores s_1 , s_2 , s_3 e s_4 (após aplicar todas as ações possíveis em s_0 , no exemplo da figura, a_1 e a_2), calculamos $Q^t(s_0, a_1)$ e $Q^t(s_0, a_2)$ usando a Equação (3.7). O maior valor é usado para atualizar $V^t(s_0)$, que no exemplo é definido por $Q^t(s_0, a_2)$. A Figura 3.2, ilustra a execução do algoritmo IV para t variando de 0 a 3, num MDP com apenas 2 estados, s_1 e s_2 e 2 ações a_1 e a_2 . A Figura 3.2 não indica os valores selecionados como no exemplo da Figura 3.1. Porém, mostra como o algoritmo de IV atualiza os valores de todos os estados a cada iteração usando o mesmo cálculo ilustrado na Figura 3.1.

No horizonte infinito a função valor converge para V^* , sendo que $V^t(s) \cong V^{t-1}(s)$. Isso pode ser formalmente definido por:

$$\lim_{t \rightarrow \infty} \max_s |V^t(s) - V^{t-1}(s)| = 0, \quad (3.9)$$

resultando na política ótima, determinística e estacionária $\pi^* = \pi_{V^\infty}$ [Puterman, 1994]. Para soluções práticas de MDPs, muitas vezes estamos apenas preocupados com ϵ -otimalidade. Se terminamos o algoritmo IV quando alcançamos a condição:

$$\max_s |V^t(s) - V^{t-1}(s)| < \frac{\epsilon(1-\gamma)}{2\gamma}, \quad (3.10)$$

então garantimos que a política gulosa π_{V^t} não perde mais do que ϵ na função valor sob um horizonte infinito, quando comparado com π^* [Puterman, 1994]. A expressão $\max_s |V^t(s) - V^{t-1}(s)|$ na Equação (3.10) é chamada de *Erro de Bellman* (BE - *Bellman Error*) e a expressão $\frac{\epsilon(1-\gamma)}{2\gamma}$ será chamada de *tolerância* (*tol*) [Puterman, 1994]. Uma vez que o algoritmo de IV precisa atualizar o espaço de estados inteiro a cada iteração, o tempo de execução para cada iteração deste algoritmo de programação dinâmica é $O(|S|^2 * |A|)$.

Outro algoritmo clássico para resolver um MDP é o algoritmo de ITERAÇÃO DE POLÍTICA (IP). Enquanto o algoritmo de ITERAÇÃO DE VALOR melhora iterativamente o valor esperado estimado, o algoritmo de ITERAÇÃO DE POLÍTICA seleciona uma política arbitrária inicial π_0 e em cada iteração modifica a política de forma que o valor esperado seja aumentado. Neste algoritmo, cada iteração i é dividida em duas fases:

- *Determinação do Valor*, em que a política atual é avaliada, i.e., para cada estado $s \in S$, $V_{\pi_i}(s)$ é calculado baseado na política π_i . Para fazer isso podemos resolver um sistema de equações lineares.
- *Aperfeiçoamento da Política*, em que a política atual é melhorada: obtemos π_{i+1} que é a política gulosa com respeito a V_{π_i} , i.e., para cada estado $s \in S$ escolhemos uma ação $a \in A$ que maximiza $Q(s, a)$, isto é:

$$\pi_{i+1}(s) = \max_{a \in A} (Q(s, a)) \quad (3.11)$$

$$\pi_{i+1}(s) = \max_{a \in A} \left(R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V_{\pi_i}(s') \right). \quad (3.12)$$

O algoritmo termina quando $\pi_{i+1}(s) = \pi_i(s)$ para todo $s \in S$. Para avaliarmos a complexidade dos algoritmos IV e IP, analisamos primeiro a complexidade de cada iteração. A fase de Determinação do Valor pode ser resolvida através do sistema de equações lineares em tempo $O(|S|^3)$ ou através de aproximações sucessivas em $O(|S|^2 * T)$ (em que T é o número de iterações para alcançar ϵ -otimalidade, no caso descontado $T \ll |S|$). A fase de Aperfeiçoamento de Política pode ser executada em $O(|S|^2 * |A|)$. Portanto, o tempo de cada iteração para os algoritmos ITERAÇÃO DE VALOR e ITERAÇÃO DE POLÍTICA é polinomial e o tempo total também é polinomial se e somente se o número de iterações necessárias para encontrar uma política ótima é polinomial [Littman et al.,

1995].

A principal desvantagem dos métodos de programação dinâmica síncrona é que eles precisam atualizar o espaço de estados inteiro a cada iteração. Isso proíbe a sua aplicação quando o espaço de estados é muito grande.

3.3.2 Programação dinâmica assíncrona

Os métodos assíncronos são mais flexíveis na seleção dos estados a serem atualizados, neste caso, o agente pode atualizar alguns estados mais do que outros. Um exemplo deste método é chamado de *Programação Dinâmica em Tempo Real* (RTDP - Real Time Dynamic Programming) [Barto et al., 1995]. Este método combina a busca heurística com atualizações de programação dinâmica da função valor dos estados encontrados durante sessões de simulações de ações, sendo que cada sessão (*trial*) é limitada em T estágios, que corresponde a um limite de profundidade de uma sessão de simulação [Barto et al., 1995]. RTDP resolve MDPs com as seguintes especificações adicionais:

- $I \subseteq S$ é um conjunto de estados iniciais e
- $G \subset S$ é um conjunto, possivelmente vazio, de estados metas, modelados como estados absorventes com auto-transições, recompensa 0 e com probabilidade 1 para todas as ações.

Em muitos problemas de planejamento, apenas uma pequena fração do espaço de estados é relevante. Assim, a programação dinâmica só precisa se concentrar na atualização de um conjunto de estados alcançáveis (i.e., estados relevantes). Nestes problemas, é suficiente calcular uma política parcial, ou seja, uma política definida só para os estados relevantes. O algoritmo RTDP apresenta as seguintes vantagens:

- não avalia o espaço inteiro de estados, mas só os estados alcançáveis a partir do conjunto de estados iniciais e
- oferece um desempenho bom em qualquer instante, i.e., o algoritmo pode ser interrompido a qualquer momento, uma vez que as soluções fornecidas melhoram monotonicamente com o tempo.

RTDP primeiro inicializa $\bar{V}_u$ (valor estimado de V^*) com um limite superior *admissível* V_{u_0} , i.e., $\bar{V}_u(s) \geq V^*(s) \forall s \in S$ e a seguir, executa várias sessões de simulações. Cada sessão começa com um estado aleatório do conjunto de estados iniciais I . Para cada estado encontrado durante uma simulação, o valor do limite superior $\bar{V}_u(s)$ é atualizado e uma ação gulosa é escolhida (Figura 3.3). Durante uma sessão de simulação, para obter o próximo estado a ser visitado, RTDP sorteia um estado a partir da função de transição $P(\cdot|s, a)$ (Figura 3.4), i.e.:

$$\text{CHOOSENEXTSTATE}(s, a) = s' \sim P(\cdot|s, a). \quad (3.13)$$

RTDP termina uma simulação se encontrar um estado meta (ou objetivo) ou quando uma profundidade limitada é alcançada (Figura 3.5). O algoritmo RTDP (Algoritmo 17) é detalhado no Apêndice A.

Figura 3.3: RTDP atualiza $\bar{V}_u$ e escolhe uma ação gulosa a_2 , isto é, aquela com o maior valor Q_u^t .

Figura 3.4: RTDP escolhe o próximo estado a ser visitado s_3 (simulação), baseado na distribuição de probabilidade e um sorteio.

Comprovou-se que, dado um limite superior inicial admissível para a função valor $\bar{V}_u$, a execução de repetidas sessões de simulações de RTDP encontra o valor ótimo para todos os estados *relevantes*, ou seja, os estados alcançáveis a partir dos estados iniciais segundo uma política ótima [Barto et al., 1995].

Note que este algoritmo ainda mantém as propriedades de convergência sob certas condições. Para o caso descontado, em que estamos interessados, RTDP converge para um valor ϵ -ótimo sobre todos os estados relevantes, dada uma heurística admissível (limite superior admissível) e fazendo as sessões de simulações com um limite de profundidade suficiente. Barto [Bonet and Geffner, 2003] descreve as condições necessárias para a convergência de RTDP, no caso sem desconto.

Figura 3.5: Exemplo de uma simulação de RTDP que terminou ao encontrar o estado objetivo.

Uma fraqueza do RTDP é que caminhos pouco prováveis tendem a ser ignorados e, consequentemente, a convergência do RTDP é lenta [Bonet and Geffner, 2003]. Assim, algumas extensões do RTDP foram propostas a fim de melhorar a convergência: *Labeled RTDP (LRTDP)* [Bonet and Geffner, 2003], *Bounded RTDP (BRTDP)* [McMahan et al., 2005], *Focused RTDP (FRTDP)* [Smith and Simmons, 2006] e *Bayesian RTDP* [Sanner et al., 2009].

LRTDP [Bonet and Geffner, 2003] rotula os estados que já convergiram para evitar visitá-los novamente e assim priorizar a visita aos demais. BRTDP [McMahan et al., 2005] mantém limites superiores e inferiores da função valor ótima, $\bar{V}_u(s)$ e $\bar{V}_l(s)$, respectivamente, e focaliza a busca em áreas onde o valor estimado está mais distante do valor ótimo.

McMahan [McMahan et al., 2005] provou que a atualização dos limites superior e inferior ($\bar{V}_u(s) \geq V^*(s) \forall s \in S$ e $\bar{V}_l(s) \leq V^*(s) \forall s \in S$) durante o BRTDP preserva a admissibilidade da heurística e faz com que o BRTDP monotonicamente convirja para $V^*(s)$ [McMahan et al., 2005], isto é:

$$\lim_{t \rightarrow \infty} (\bar{V}_l^t(s) = V^*(s) = \bar{V}_u^t(s)).$$

A diferença entre os limites superior e inferior ($\bar{V}_u^t(s) - \bar{V}_l^t(s)$) fornece uma *medida da incerteza* do valor para o estado s . BRTDP primeiro inicializa $\bar{V}_u$ e $\bar{V}_l$ com valores superiores e inferiores admissíveis e então executa várias simulações. Cada simulação começa escolhendo um estado inicial. Para cada estado visitado, os valores superior e inferior são atualizados e uma ação gulosa é escolhida. BRTDP prioriza a escolha do próximo estado de acordo com essa diferença. BRTDP termina uma simulação se: encontrar um estado meta, se uma profundidade limite for atingida ou se o valor da incerteza for pequeno. O algoritmo BRTDP (Algoritmo 18) e o algoritmo que escolhe o próximo estado a ser visitado por BRTDP (Algoritmo 19) são mostrados no Apêndice A.

BRTDP converge pela simples razão de que ele ainda atualiza todos os estados relevantes que RTDP atualizaria com probabilidade não-zero, mas sorteia com base numa distribuição diferente, isto é, uma distribuição que predispõe as atualizações de estados mais incertos, a fim de reduzir essa incerteza mais rapidamente e assim convergir mais rapidamente [McMahan et al., 2005].

As soluções de programação dinâmica assíncrona, no pior caso, possuem a mesma complexidade das soluções de programação dinâmica síncrona (IV e IP). Porém, dependendo do tamanho do conjunto de estados relevantes do problema em questão, as soluções do tipo RTDP podem resolver MDPs envolvendo milhares de estados.

3.4 MDP: Soluções Baseadas em Programação Matemática

É possível resolver um MDP usando-se programação matemática. Na sequência, mostramos uma solução de programação linear exata e uma solução aproximada que usa funções de valor linear.

3.4.1 Solução exata baseada em programação linear

A Equação de Bellman (3.6) também pode ser formulada como um programa de otimização linear [Manne, 1960]:

$$\begin{aligned} \min_{V^*} & : \sum_s V^*(s) \\ s.a & : V^*(s) \geq R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^*(s'), \forall s \in S, a \in \mathcal{A}, \end{aligned} \quad (3.14)$$

sendo $V^*(s)$ as variáveis que se deseja otimizar.

Lema 3.1. *A solução do problema linear (3.14) é a função valor ótima, V^* .*

No programa linear (3.14) podemos encontrar a solução de ponto-fixo única da Equação de Bellman (3.6) que é $V^*(s)$, ao usar as restrições para forçar que $V^*(s)$ para $s \in S$ seja maior ou igual que:

$$R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^*(s'), \forall a \in \mathcal{A}$$

o que corresponde a forçar que: $V^*(s) \geq \max_a \{R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^*(s')\}$, $\forall s \in S$ e então minimizamos $\sum_s V^*(s)$ para forçar que a igualdade seja verdadeira.

Embora o uso de programação linear tenha sido associado com ineficiência computacional no passado, muito progresso nessa área tem acontecido. Os métodos de programação linear para MDPs produzem uma solução exata, sem necessidade de especificar nenhum critério de parada (critério necessário nos algoritmos IV e IP). Porém, o número de estados em aplicações práticas é muito grande para usar essa abordagem de otimização exata (o que resulta em $|S| \times |A|$ restrições no programa (3.14)). Algoritmos de Programação Linear Aproximados foram propostos para resolver essa limitação.

3.4.2 Solução aproximada baseada em programação linear

A idéia básica da *Programação Linear Aproximada* (ALP - *Approximate Linear Programming*) é resolver um MDP formulado como o Problema (3.14), aproximando a função valor ótima através de funções base. Funções base são fornecidas por especialistas do domínio ou geradas automaticamente [Patrascu, 2004, Mahadevan, 2005]. A função valor aproximada é denotada por $\hat{V}(s)$. Dado $s \in S$ e um conjunto de funções base $H = \{h_1, \dots, h_k\}$, $V^*(s)$ pode ser aproximada por uma combinação linear de H :

$$\hat{V}(s) = \sum_{j=1}^k w_j h_j(s). \quad (3.15)$$

Uma vez que o conjunto de funções base tem sido especificado, o problema de encontrar uma função valor aproximada se reduz ao problema de encontrar um bom conjunto de pesos w_j . A qualidade *a posteriori* da função valor aproximada $\hat{V}(s)$ pode ser medida com base na distância entre o vetor de valores aproximados $\hat{V}$ e o vetor de valores ótimos V^* . Para isso, podemos escolher

uma entre as diferentes normas ² [Patrascu, 2004]:

$$L_1 = \|V^* - \hat{V}\|_1 = \sum_s |V^*(s) - \hat{V}(s)| \quad (3.16)$$

$$L_2 = \|V^* - \hat{V}\|_2 = \sqrt{\sum_s (V^*(s) - \hat{V}(s))^2} \quad (3.17)$$

$$L_\infty = \|V^* - \hat{V}\|_\infty = \max_s |V^*(s) - \hat{V}(s)| \quad (3.18)$$

Análises de erro e convergência de algoritmos para MDP geralmente usam a norma infinita (max) (Equação (3.18)). No entanto, a qualidade da aproximação depende mais do conjunto de funções base selecionado do que da escolha da melhor norma. Em resumo, a formulação ALP de um MDP, considerando-se o Problema (3.14) e a Expressão (3.15), é dada pelo programa linear:

$$\begin{aligned} \min_w \quad & \sum_s \sum_{i=1}^k w_i h_i(s) \\ s.a \quad & \sum_{i=1}^k w_i h_i(s) \geq R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) \sum_{i=1}^k w_i h_i(s'), \\ & \forall s \in S, a \in A. \end{aligned} \quad (3.19)$$

A formulação ALP não otimiza diretamente o erro usando a norma max, no entanto se as funções base selecionadas admitem uma solução ótima, então a função objetivo garante encontrar essa solução ótima, o que realmente minimiza o erro usando a norma max (que neste caso é zero).

Note que o Problema (3.19) continua sendo linear nos pesos w_i a serem otimizados e o número de variáveis (w_i) deve ser menor que $|S|$, se selecionarmos um número relativamente pequeno de funções base. No entanto, o número de restrições é o mesmo que no Problema (3.14). No Capítulo 4 veremos como é possível reduzir o número de restrições num MDP formulado como um problema ALP.

3.5 Resumo

Neste capítulo foram apresentadas: a definição de MDPs e as técnicas principais para solucioná-los. Dentre essas soluções, existem três dimensões de escolhas: a primeira é de se utilizar métodos de programação dinâmica versus programação linear; a segunda escolha é de se determinar soluções exatas ou aproximadas; e a terceira está relacionada à escolha de algoritmos que fornecem políticas parciais ou totais. Soluções parciais são indicadas se uma pequena fração do espaço de estados for relevante para um conjunto restrito de estados iniciais. Para isso, é possível usar algoritmos

²Norma é uma função que atribui um número real não-negativo para um vetor, denotada por $\|\cdot\|$, e está relacionada com o comprimento do vetor.

baseados em programação dinâmica assíncrona, os quais oferecem ganhos computacionais em termos de tempo de convergência para determinar a política ótima (família de algoritmos RTDP).

É importante notar que existem muitos problemas de planejamento probabilístico que exigem a enumeração explícita de estados, a fim de representar ou de resolver o problema, como foi adotado nas soluções desse capítulo. No Capítulo 4 apresentamos uma representação mais natural para MDPs e discutimos como obter ganhos computacionais com essa representação de modo a superar as limitações das soluções para MDPs discutidas nesse capítulo.

Capítulo 4

Processo de Decisão Markoviano Fatorado

No capítulo anterior vimos MDPs com uma representação de estados enumerados (que chamamos de MDPs enumerativos). Neste capítulo, nos concentramos numa representação mais natural¹ para MDPs, que usa uma representação de estados fatorada em termos de variáveis de estado, chamada de **MDP Fatorado**. Versões fatoradas de Processos de Decisão Markovianos têm sido bastante estudadas [Boutilier et al., 1999], particularmente nas soluções de problemas grandes de planejamento, tratados em Inteligência Artificial.

4.1 Modelo Fatorado

Num *MDP Fatorado*, estados são descritos por um conjunto $\mathbf{X} = \{X_1, X_2, \dots, X_n\}$ de *variáveis de estado*. Dessa forma, um estado $s \in \mathcal{S}$ é representado como um vetor $\vec{x} = (x_1, x_2, \dots, x_n)$, em que x_i é o valor da variável de estado X_i . Com essa representação, todo $s \in \mathcal{S}$ é expresso por $\vec{x}$, isto é, $s = \vec{x}$ (usaremos a notação $\vec{x} \in \mathcal{S}$), sendo a cardinalidade de $\mathcal{S}$ exponencial no número n de variáveis de estado. Assim, enquanto encontrar uma política ótima é P-Completo para MDPs enumerativos [Papadimitriou and Tsitsiklis, 1987, Patrascu, 2004], MDPs fatorados introduzem um aumento exponencial no tamanho do espaço de estados (isto é, presupondo variáveis de estado binárias, $|\mathcal{S}| = 2^n$).

4.1.1 Recompensa local

Num MDP Fatorado, a função recompensa $R(\vec{x}, a)$ é usualmente definida pela soma de ψ funções de recompensa local $R_j(\vec{x}, a)$:

$$R(\vec{x}, a) = \sum_{j=1}^{\psi} R_j(\vec{x}, a). \quad (4.1)$$

O escopo de cada função recompensa local $R_j(\cdot, a)$ é restrita, tipicamente, a um subconjunto de variáveis $\mathbf{X}$. Por exemplo, podemos definir uma recompensa local sobre variáveis de estado individuais como: $R_j(X_j = 1, a) = 1$ e $R_j(X_j = 0, a) = 0$ para $j = 1, 2, \dots, n$. Assim, para o estado $\vec{x}_0 = (1, 1, \dots, 1)$, teríamos $R(\vec{x}_0, a) = n$.

¹Problemas grandes de tomada de decisão sequencial envolvem um número muito grande de estados. Assim, uma descrição compacta do problema, em termos de variáveis de estado, é a maneira mais natural de se especificar um MDP.

Figura 4.1: a) Uma Rede Bayesiana Dinâmica (DBN) para uma ação $a \in A$. b) Tabela de probabilidade condicional para $X'_2 = 1$. c) Tabela de probabilidade condicional para $X'_2 = 0$. d) ADD para representar a função de transição que chamaremos de $P_{DD}(x'_2 | x_1, x_2, x'_1, a)$; a linha contínua indica o ramo verdadeiro ($X_i = 1$) da variável de teste e a linha tracejada indica o ramo falso ($X_i = 0$).

4.1.2 Modelo de transição fatorado

Num MDP fatorado as probabilidades de transição de estado são representadas usando **Redes Bayesianas Dinâmicas (DBNs)** [Dean and Kanazawa, 1990]. Ou seja, para cada ação $a \in A$ descrevemos um grafo dirigido acíclico com duas camadas: uma camada representa as variáveis de estado atual e a outra camada representa as variáveis do próximo estado (Figura 4.1.a). Os nós são denotados por X_i e X'_i para as variáveis do estado atual e do próximo estado, respectivamente. Arestas são permitidas de nós da primeira camada para a segunda camada, e também entre nós da segunda camada (aresta síncrona). Denotamos por $\text{pa}(X'_i)$ os pais de X'_i no grafo, isto é, os nós que possuem uma aresta incidente em X'_i . Por exemplo, na Figura 4.1.a $\text{pa}(X'_2) = \{X_1, X_2, X'_1\}$. É feita a suposição que o grafo obedece a seguinte condição de Markov: uma variável X'_i é condicionalmente independente de seus não-descendentes não-pais dados seus pais. Isso implica na seguinte fatoração das probabilidades de transição:

$$P(\vec{x}' | \vec{x}, a) = \prod_{i=1}^n P(x'_i | \text{pa}(X'_i), a), \quad (4.2)$$

isto é, a probabilidade conjunta de ir para $\vec{x}' \in \mathcal{S}$, dado que o agente está no estado $\vec{x} \in \mathcal{S}$ e executa a ação $a \in \mathcal{A}$, é o produto das probabilidades condicionais para $\{X'_i = x'_i\}$ dados os valores dos pais de X'_i e a ação $a \in \mathcal{A}$. Por exemplo, para a DBN da Figura 4.1.a, a distribuição de probabilidade conjunta pode ser fatorada como:

$$P(\vec{x}' | \vec{x}, a) = P(x'_1 | x_1, x_4, a) \cdot P(x'_2 | x_1, x_2, x'_1, a) \cdot P(x'_3 | x_2, x_3, a) \cdot P(x'_4 | x_3, x_4, a).$$

Tradicionalmente, a função de transição em uma Rede Bayesiana Dinâmica é dada por tabelas de probabilidade condicional (CPTs), como visto na Seção 2.2, uma para cada tripla $\langle X_i, x_i, a \rangle$.

A Figura 4.1.b mostra a tabela de probabilidade condicional para $\langle X'_2, 1, a \rangle$, isto é, $X'_2 = 1$ e ação a . A Figura 4.1.c mostra a CPT para $X'_2 = 0$ e a mesma ação a . As tabelas incluem somente as combinações dos valores das variáveis em $\text{pa}(X'_2)$, isto é, as tabelas envolvem apenas os pais da variável X'_2 . Por isso, dizemos que a DBN permite explorar a estrutura do domínio, uma vez que a DBN permite identificar as independências entre variáveis para representar a função de transição. A última coluna da tabela indica as probabilidades de transição.² Por exemplo, a primeira linha da tabela (Figura 4.1.b) indica que $P(X'_2 = 1 | X_1 = 1, X_2 = 1, X'_1 = 1, a) = 0.4$. Note que a soma de cada linha na Figura 4.1.b e Figura 4.1.c deve ser igual a 1, o que define uma função de densidade de probabilidades sobre os valores que a variável X'_2 pode assumir.

A seguir descrevemos o modelo fatorado do Problema SYSADMIN mostrado na Seção 1.

Exemplo 4.1. SysAdmin No domínio SYSADMIN [Guestrin et al., 2003] temos n computadores $c_1, \dots, c_n$ conectados através de diferentes topologias de grafos dirigidos: (a) *anel unidirecional*, (b) *anel bidirecional* e (c) *anéis bidirecionais independentes de pares de computadores* e (d) *estrela* (Figure 4.2).

A variável x_i indica se um computador c_i está ligado e funcionando ($X_i = 1$) ou não ($X_i = 0$). $\text{Conn}(c_j, c_i)$ denota a conexão de c_j para c_i . Formalmente, as CPTs na DBN de transição para este domínio tem a seguinte forma:

$$P(X'_i = 1 | \vec{x}_i, a) = \begin{cases} \text{se } a = \text{reboot}(c_i) & \text{então } 1. \\ \text{se } a \neq \text{reboot}(c_i) \wedge x_i = 1 & \text{então} \\ & 0.95 \cdot \frac{|\{x_j | j \neq i \wedge x_j = 1 \wedge \text{Conn}(c_j, c_i)\}| + 1}{|\{x_j | j \neq i \wedge \text{Conn}(c_j, c_i)\}| + 1}. \\ \text{se } a \neq \text{reboot}(c_i) \wedge x_i = 0 & \text{então} \\ & 0.05 \cdot \frac{|\{x_j | j \neq i \wedge x_j = 1 \wedge \text{Conn}(c_j, c_i)\}| + 1}{|\{x_j | j \neq i \wedge \text{Conn}(c_j, c_i)\}| + 1}. \end{cases} \quad (4.3)$$

Temos $n + 1$ ações: $\text{reboot}(c_1), \dots, \text{reboot}(c_n)$ e $\text{notreboot}()$. A ação $\text{notreboot}()$ indica que nenhuma máquina foi reiniciada. Se um computador é reiniciado, a sua probabilidade de funcionar no instante seguinte é 1, caso contrário a probabilidade depende do seu estado atual e do número de computadores conectados a ele que também estão funcionando no estado atual.

Além disso, a recompensa é 1 se todos os computadores estão funcionando em qualquer instante do tempo, caso contrário, a recompensa é 0, i.e., $R(\vec{x}) = \prod_{i=1}^n \mathbb{I}[X_i = 1]$ ³.

Uma política ótima neste exemplo irá reiniciar o computador que tenha o maior impacto na recompensa esperada descontada, dada a configuração da rede. Por exemplo, na configuração estrela da Figura 4.2, se o computador c_0 e c_2^2 não estão funcionando, o administrador deve reiniciar c_0 , uma vez que essa é a ação que terá maior impacto em toda a rede. $\diamond$

²Para simplificar a notação nas CPTs indicamos o valor da variável, ao invés da notação de probabilidade condicional. Por exemplo, na Figura 4.1.b usamos X'_2 ao invés de $P(X'_2 = 1 | \text{pa}(X'_2), a)$ como na Seção 2.2.

³ $\mathbb{I}$ é a função indicador.

Figura 4.2: Topologias de rede para o exemplo SYSADMIN: a) anel unidirecional, b) anel bidirecional, (c) anéis bidirecionais independentes de pares de computadores e (d) estrela [Guestrin et al., 2003]

4.2 Diagramas de Decisão Algébrica

Podemos representar a função de transição numa DBN de forma ainda mais compacta do que enumerar todos os possíveis valores que as variáveis pais podem assumir através de CPTs. Por exemplo, na Figura 4.1.b, para $X'_1 = 1$ a probabilidade de transição é 0.4, independentemente dos valores das variáveis X_1 e X_2 . Ou seja, numa CPT, podemos identificar que alguns valores de probabilidade de transição independem dos valores de algumas variáveis pais. Essa independência é conhecida como *independência específica de contexto* (CSI - *Context-Specific Independence*) [Boutilier et al., 1996]. Para poder representar de forma compacta esse tipo de independência, usamos os *Diagramas de Decisão Algébrica* (ADDs - *Algebraic Decision Diagrams*) [Bahar et al., 1993]. Dessa forma, será possível explorar ainda mais a estrutura de um domínio de aplicação para a construção de soluções eficientes.

ADDs permitem cálculos eficientes durante a ITERAÇÃO DE VALOR, como veremos na Seção 4.3. A seguir, definiremos os Diagramas de Decisão Algébrica, suas operações básicas e mostraremos como eles podem ser usados na construção de algoritmos eficientes para MDPs [Hoey et al., 1999, St-aubin et al., 2000].

Figura 4.3: Um exemplo de função recompensa $R(x_1, x_2, x_3) = \sum_{i=1}^3 x_i$ representada como um ADD. A linha contínua indica o ramo verdadeiro ($X_i = 1$) da variável de teste e a linha tracejada indica o ramo falso ($X_i = 0$).

O *Diagrama de Decisão Algébrica* (ADD) [Bahar et al., 1993] é uma generalização de um *Diagrama de Decisão Binária* (BDD) que representa funções booleanas do tipo $f : \{0, 1\}^n \rightarrow \{0, 1\}$ [Bryant, 1992]. Um BDD é uma estrutura de dados que contém nós de decisão, cada um rotulado com uma variável booleana (variável de teste) com dois nós sucessores, chamados: sucessor l (low) e sucessor h (high). A aresta de um nó ao seu sucessor l (h) representa uma atribuição 0 (1) à variável. O termo BDD em geral refere-se a *Reduced Ordered Binary Decision Diagram* (ROBDD), usado quando aspectos de redução e ordenação precisam ser enfatizados. Uma das vantagens de ROBDD é que existe um único ROBDD que representa uma função.

Soluções eficientes para resolver MDPs fatorados usam ADDs para representar funções reais. A única diferença entre um ADD e um BDD é que os nós terminais de um ADD são valores reais, isto é, ADDs representam funções do tipo $f : \{0, 1\}^n \rightarrow \mathbb{R}$, como *grafos dirigidos acíclicos* (DAGs - Directed Acyclic Graphs), cujas variáveis de teste em qualquer caminho da raiz até uma folha seguem uma ordenação previamente fixada. Um ADD é usado para gerar o valor de uma função real da seguinte forma: dados os valores booleanos de teste, seguimos os ramos l ou h , respectivamente até chegarmos numa folha, que é o valor real devolvido pela função (Figura 4.3).

Assim como nos BDDs, ADDs podem ser reduzidos quando a mesma função é compartilhada por diferentes nós (subgrafos idênticos). Por exemplo, a função recompensa $R(x_1, x_2, x_3) = \sum_{i=1}^3 x_i$, representada na Figura 4.3 como um ADD, explora a estrutura redundante dos sub-diagramas através de sua representação DAG. Além disso, ADDs fornecem uma representação compacta de funções com independência do tipo CSI [Boutilier et al., 1996]. Na Figura 4.1.d mostramos o ADD para $P(x'_2 | x_1, x_2, x'_1, a)$. Note que no ADD não é necessário testar os valores das variáveis x_1 e x_2 , quando sabemos que $X'_1 = 1$.

Formalmente, podemos definir um ADD F através de uma gramática BNF (Backus Naur Form). Os nós de decisão internos do ADD são rotulados com uma variável individual de teste var , e têm grau 2; os dois arcos de saída são o *ramo verdadeiro* da variável de teste (F_h) e o *ramo falso* (F_l). Os nós terminais que têm grau 0 são rotulados com um valor constante $Const \in \mathbb{R}$. Assim, a

Figura 4.4: Representação gráfica de um ADD: *var* é o nome da variável, e F_h e F_l são os identificadores dos nós dos ramos verdadeiro e falso, respectivamente.

Figura 4.5: Um diagrama de decisão ordenado e a sua representação canônica. A linha contínua indica o ramo verdadeiro ($X_i = 1$) da variável de teste e a linha tracejada indica o ramo falso ($X_i = 0$).

gramática BNF é dada por:

$$F ::= \text{Const} \mid \text{if}(F^{var}) \text{ then } F_h \text{ else } F_l. \quad (4.4)$$

Isso significa que, para avaliar uma expressão, se *var* é igual a *verdadeiro* então o ramo F_h deve ser seguido e quando *var* é igual a *falso* F_l deve ser seguido (daí o nome “diagrama de decisão”). Uma representação gráfica da Expressão (4.4) é mostrada na Figura 4.4.

O valor de um ADD F contendo um conjunto de variáveis $\{x_1, \dots, x_n\}$ na atribuição de valores das variáveis $\rho \in \{0, 1\}^n$ pode ser definido recursivamente por:

$$\text{Val}(F, \rho) = \begin{cases} \text{se } F = \text{Const} & \text{então } \text{Const}. \\ \text{se } F \neq \text{Const} \wedge \rho(F^{var}) = \text{true} & \text{então } \text{Val}(F_h, \rho). \\ \text{se } F \neq \text{Const} \wedge \rho(F^{var}) = \text{false} & \text{então } \text{Val}(F_l, \rho). \end{cases}$$

Para qualquer função $f(x_1, \dots, x_n)$ e uma ordenação de variáveis fixa sobre $x_1, \dots, x_n$, um ADD reduzido é definido como a representação de *diagrama de decisão ordenado de tamanho mínimo* que representa a função f . Existe um único ADD reduzido que representa a função f , chamado de representação canônica. Isso foi provado para BDDs em [Bryant, 1986] e pode ser generalizado trivialmente para ADDs.

Na Figura 4.5 mostramos um Diagrama de Decisão Algébrico Ordenado (OADD que chamaremos simplesmente de ADD) e a sua representação canônica.

Operações unárias tais como $\min(\cdot)$, $\max(\cdot)$, bem como operações binárias como adição ($\oplus$),

Figura 4.6: a) Função f e g . b) Representação canônica de f e g . Figura 4.7: a) Funções f , g e $f + g$. b) ADDs canônicas da função f , g e da soma $f \oplus g$.

subtração ($\ominus$), multiplicação ($\otimes$), $\min(\cdot, \cdot)$ e $\max(\cdot, \cdot)$, podem ser executadas de forma eficiente em ADDs. Na Figura 4.6 mostramos as funções f e g com suas respectivas representações ADD canônicas. A Figura 4.6.a mostra as funções f e g representadas em tabelas. Note que a função g somente depende da variável X_2 , o que é representado de forma compacta com o ADD da Figura 4.6.b.

O resultado da operação $f + g$ é mostrado na última coluna da tabela na Figura 4.7.a e a sua representação canônica $f \oplus g$ é mostrada na Figura 4.7.b.

Na Figura 4.8 é mostrado o resultado de $\max(f)$ e na Figura 4.9 é mostrado o resultado da operação $\max(f, g)$. A diferença entre $\min(\cdot)$ e $\min(\cdot, \cdot)$ ($\max(\cdot)$ e $\max(\cdot, \cdot)$) é que o resultado da operação unária é um número real, i.e., o valor mínimo (máximo) das folhas, respectivamente, enquanto o resultado da operação binária é um ADD, i.e., o mínimo (máximo) entre as duas funções ponto a ponto.

A operação auxiliar unária *restringir* uma variável X_i para *verdadeiro* ou *falso* consiste em considerar somente as linhas na tabela em que X_i é *verdadeira* (1) ou *falsa* (0), operação que representamos por $F|_{X_i=1}$ ou $F|_{X_i=0}$, respectivamente. Na Figura 4.10 mostramos a função P e o resultado da operação $P|_{X_2=1}$. Note que no ADD canônico resultante, Figura 4.10.b, somente foram representadas as linhas da tabela da Figura 4.10.a em que $X_2 = 1$.

Figura 4.8: a) Função f e $\max(f)$. b) ADD canônica de f e o resultado da operação $\max(f)$.

Figura 4.9: a) Funções f , g e $\max(f,g)$. b) ADDs canônicas das funções f , g e do máximo $\max(f,g)$.

Figura 4.10: a) Funções P e $P|_{x_2=1}$ (i.e., a operação que considera unicamente as linhas da tabela em que $x_2 = 1$). b) Representação canônica de P e o resultado da operação $P|_{x_2=1}$.

Uma outra operação importante que pode ser executada em ADDs e usada para resolver MDPs é a marginalização (também chamada de *sum-out*) ou marginalização de probabilidade conjunta como vista na Seção 2.1.4. Por exemplo, a marginalização de $P(x_1, x_2, x_i, \dots)$ com relação a uma

Figura 4.11: a) As funções $P(X_1, X_2)$ e marginalização $\sum_{x_2} P$. b) Resultado da operação de marginalização $\sum_{x_2 \in X_2} P = F|_{x_2=1} \oplus F|_{x_2=0}$ sobre ADDs (como a soma de duas operações de restrição).

variável é denotada por $\sum_{x_i \in X_i} P(X_1, X_2, X_i, \dots)$ (numa notação simplificada usamos $\sum_{x_i \in X_i} P$). Essa operação pode ser calculada usando a função auxiliar *restringir* como definida acima. Na Figura 4.11.a temos uma distribuição de probabilidade conjunta $P(X_1, X_2)$ e a marginalização sobre X_2 . Note que a marginalização $\sum_{x_2} P$ elimina a variável X_2 , resultando numa tabela com dois valores, uma para cada valor possível da variável X_1 . A operação $\sum_{x_2} P$, usando ADDs, pode ser calculada restringindo a variável X_2 para ser verdadeira ($F|_{X_2=1}$), e restringindo a variável X_2 para ser falsa ($F|_{X_2=0}$), e finalmente somando ambos os resultados.

4.3 MDP Fatorado: Soluções baseadas em Programação Dinâmica

Para resolvermos um MDP Fatorado, são conhecidos algoritmos exatos e aproximados baseados em programação dinâmica, tais como: SPI [Boutilier et al., 1995], SPUDD [Hoey et al., 1999] e APRICODD [St-aubin et al., 2000]. Esses algoritmos criam um modelo abstrato em que estados similares são agrupados resultando em uma redução do tamanho do espaço de estados e dessa maneira, tentam tratar a “maldição da dimensionalidade de Bellman”.

O algoritmo SPI (*Structured Policy Iteration*) [Boutilier et al., 1995] usa árvores para agregar estados similares. Esse algoritmo implementa ITERAÇÃO DE POLÍTICA modificada em que CPTs, recompensa, políticas e funções valor são representadas por árvores de decisão. Um inconveniente deste tipo de representação é que não pode representar de forma compacta funções com sub-estruturas compartilhadas, o que implica na replicação dos valores [Hoey et al., 1999]. Para lidar com este problema, foi proposto o uso de ADDs, no lugar de árvores de decisão.

4.3.1 SPUDD

SPUDD (*Stochastic Planning using Decision Diagrams*) [Hoey et al., 1999] estende o algoritmo de ITERAÇÃO DE VALOR para explorar a estrutura do domínio de aplicação representada por uma DBN e ADD e encontrar uma solução de maneira eficiente. Na Seção 4.2 vimos que ADDs podem ser usados para representar de forma compacta tanto a função de transição como CPTs quanto a função recompensa de um MDP. Adicionalmente, ADDs podem ser usados para representar a função valor, geralmente de forma extremamente compacta e agrupando estados de mesmo valor em cada iteração do algoritmo. Para isso, expressamos a atualização de Bellman na ITERAÇÃO DE VALOR (Equações (3.7) e (3.8)) usando operações em ADDs da seguinte forma fatorada :

$$V_{DD}^t(\vec{x}) = \max_{a \in A} \left\{ R_{DD}(\vec{x}, a) \oplus \gamma \sum_{\vec{x}'} \bigotimes_{i=1}^n P_{DD}(x'_i | pa_a(x'_i), a) V_{DD}^{t-1}(\vec{x}') \right\}. \quad (4.5)$$

em que as variáveis com índice DD indicam funções representadas por ADDs. Note que na Equação (4.5) usamos a operação unária de *marginalização* e as operações binárias: $\oplus$, $\otimes$ e $\max(\cdot, \cdot)$. Assim, o $V^t(\vec{x})$ resultante pode ser expresso como um ADD (uma vez que essas operações são fechadas para ADDs).

SPUDD usa o algoritmo de ELIMINAÇÃO DE VARIÁVEIS [Zhang and Poole, 1994] fazendo a marginalização sobre cada variável de estado $\sum_{x'_i}$ no lugar de fazer diretamente a marginalização sobre todos os estados $\sum_{\vec{x}}$. As principais funções do SPUDD são:

- SOLVEMDP (Algoritmo 1) constrói uma serie de funções valor t-estágios-para-frente V_{DD}^t , representadas como ADDs. Primeiro, cria-se a representação ADD de todas as CPTs das DBNs no MDP e inicializa-se a primeira função valor com 0. O laço é repetido até atingir um número máximo de iterações (*maxIter*) ou até que a condição de término ($BE < tol$) é satisfeita, em que o Erro de Bellman é definido como $BE = \max_{\vec{x}} |V^t(\vec{x}) - V^{t-1}(\vec{x})|$ e *tol* é a tolerância. Em cada iteração o algoritmo REGRESS é chamado e V_{DD}^t é atualizado com o máximo sobre todos os Q_{DD}^t (sendo que existe um Q_{DD}^t para cada ação *a*). Em seguida, *BE* é computado e testado para terminar.
- REGRESS (Algoritmo 2) atualiza V_{DD}^{t-1} (Atualização de Bellman), i.e., fornece o valor Q_{DD}^t que seria obtido se executamos a ação *a*. Durante a atualização convertemos cada X_i para X'_i (usando a função CONVERTTOPRIMES) e as CPTs para a ação *a* são multiplicadas e marginalizadas (note que a marginalização é sobre cada variável de estado). Finalmente, o valor futuro é descontado de um fator γ e a recompensa R_{DD} é adicionada para completar a Atualização de Bellman.

SPUDD mostrou ser um método promissor para a resolução de MDPs fatorados de maneira eficiente cuja estrutura pode ser explorada pelos ADDs. No pior caso, quando todos os estados têm valores diferentes, ou seja, não existe uma estrutura compacta que pode ser explorada pelos ADDs, a sobrecarga com a criação de ADDs não é muito grave (SPUDD leva apenas uma constante vezes

Algoritmo 1: SOLVEMDP(MDP, tol , $maxIter$)

```

input : MDP (given by  $\langle S, A, R, P, \gamma \rangle$ ),
         $tol$  (tolerance that guarantees  $\epsilon$ -optimality),
         $maxIter$  (maximum number of iterations)
output:  $V_{DD}^t$  (t-state-to-go value function)

1 begin
2   Create ADD CPTs  $CPT_a^{x'_i}$  for MDP;
3    $V_{DD}^0 = 0$ ;
4    $t = 0$ ;
5   //construct t-stage-to-go value functions  $V_{DD}^t$  until termination condition is met
6   while  $t < maxIter$  do
7      $t = t + 1$ ;
8      $V_{DD}^t = -\infty$ ;
9     //update  $V_{DD}^t$  with the max over all  $Q_{DD}^t$ 
10    foreach  $a \in A$  do
11       $Q_{DD}^t = \text{REGRESS}(V_{DD}^{t-1}, a)$ ;
12       $V_{DD}^t = \max(V_{DD}^t, Q_{DD}^t)$ ;
13    //compute Bellman Error (BE) and check for termination
14     $Diff_{DD} = V_{DD}^t \ominus V_{DD}^{t-1}$ ;
15     $BE = \max(\max(Diff_{DD}), -\min(Diff_{DD}))$ ;
16    if  $BE < tol$  then
17      break;
18    end
19  return  $V_{DD}^t$  ;
20 end

```

Algoritmo 2: REGRESS(V_{DD}, a)

```

input :  $V_{DD}$  (value function),
         $a$  (action)
output:  $Q_{DD}$  (the value function obtained if executing  $a$  and acting so as obtain  $V_{DD}$ 
        thereafter)

1 begin
2    $Q_{DD} = \text{CONVERTTOPRIMES}(V_{DD})$ ; //convert variables  $X_i$  to  $X'_i$ 
3   //CPTs are multiplied in and summed out
4   for all  $X'_i$  in  $Q_{DD}$  do
5      $Q_{DD} = Q_{DD} \otimes CPT_a^{x'_i}$ ;
6      $Q_{DD} = \sum_{x'_i \in X'_i} Q_{DD}$ ;
7    $Q_{DD} = R_{DD} \oplus (\gamma \otimes Q_{DD})$  ;
8   return  $Q_{DD}$  ;
9 end

```

o tempo da ITERAÇÃO DE VALOR com espaço enumerativo) [Hoey et al., 1999]. Portanto, no pior caso, SPUDD mantém a mesma complexidade dos algoritmos IV e IP.

Figura 4.12: a) A função valor V^t representada como um ADD. b) O resultado do passo de redução de ADDs de APRICODD, que substitui os nós internos por um intervalo de valores.

4.3.2 APRICODD

Às vezes, é impossível obter soluções exatas devido às limitações de tempo e espaço. Assim, a fim de resolver MDPs grandes, muitas vezes temos que recorrer a métodos aproximados.

APRICODD (*Approximate Policy Construction using Decision Diagrams*) [St-aubin et al., 2000] é uma versão aproximada de SPUDD com um passo adicional após a linha 17 do algoritmo SOLVEMDP (Algoritmo 1), que reduz o tamanho das funções valor geradas durante a ITERAÇÃO DE VALOR. Para isso, APRICODD poda nós internos na representação ADD da função valor (V_{DD}^t) e os substitui com o valor mínimo e máximo das folhas, criando um novo ADD reduzido em cada iteração, cujas folhas são representadas por intervalos de valores reais. Um exemplo é mostrado na Figura 4.12. É importante notar que este método ainda produz garantias sobre o erro de aproximação [St-aubin et al., 2000].

4.4 MDP Fatorado: Soluções baseadas em Programação Matemática

Como na Seção 3.4.2, a função valor para MDPs fatorados pode ser aproximada por uma combinação linear de funções base:

$$\hat{V}(\vec{x}) = \sum_{j=1}^k w_j h_j(\vec{x}). \quad (4.6)$$

Por exemplo, podemos aproximar a função valor ótima para o Exemplo 4.1 usando a combinação linear de funções base sobre variáveis simples $h_i(X_i = 1) = 1$ e $h_i(X_i = 0) = 0$.

A formulação ALP de um MDP fatorado, considerando-se as Expressões (3.14), (4.1) e (4.6), é dada pelo programa linear:

$$\begin{aligned}
\min_w \quad & \sum_{\vec{x}} \sum_{i=1}^k w_i h_i(\vec{x}) \\
\text{s.a} \quad & \sum_{i=1}^k w_i h_i(\vec{x}) \geq \sum_{j=1}^{\psi} R_j(\vec{x}, a) + \gamma \sum_{\vec{x}' \in \mathcal{S}} P(\vec{x}' | \vec{x}, a) \sum_{i=1}^k w_i h_i(\vec{x}'), \\
& \forall \vec{x} \in \mathcal{S}, a \in \mathcal{A}.
\end{aligned} \tag{4.7}$$

Koller e Parr [Koller and Parr, 1999] mostraram que, se estamos trabalhando com um MDP fatorado (Problema (4.7)), uma condição necessária para aplicar de forma eficiente a técnica ALP é restringir o escopo de cada função base para um subconjunto pequeno de variáveis de estado $\mathbb{N}_i \subset \mathbf{X} = \{X_1, \dots, X_n\}$ e também assumir pouca dependência na DBN⁴. Guestrin [Guestrin, 2003] explorou essas condições e desenvolveu algoritmos eficientes para MDPs fatorados. O sucesso do seu algoritmo FACTOREDLP é devido a: (i) exploração da estrutura da representação fatorada para evitar a geração de restrições complexas na especificação do problema ALP [Koller and Parr, 1999]; (ii) o algoritmo FACTOREDLP que cria um novo conjunto menor de restrições equivalentes para o problema de programação linear (4.7) e (iii) o uso de funções base capazes de gerar uma boa aproximação (o que nem sempre é uma tarefa fácil para determinados domínios).

Do problema (4.7), dado $\vec{x} \in \mathcal{S}$ e $a \in \mathcal{A}$, temos restrições do tipo:

$$\sum_i w_i h_i(\vec{x}) \geq \sum_{j=1}^{\psi} R_j(\vec{x}, a) + \gamma \sum_{\vec{x}' \in \mathcal{S}} P(\vec{x}' | \vec{x}, a) \sum_i w_i h_i(\vec{x}').$$

Podemos reordenar a soma e obter:

$$0 \geq \sum_{j=1}^{\psi} R_j(\vec{x}, a) + \gamma \sum_i w_i \sum_{\vec{x}' \in \mathcal{S}} P(\vec{x}' | \vec{x}, a) h_i(\vec{x}') - \sum_i w_i h_i(\vec{x}).$$

Finalmente, as restrições podem ser reescritas como:

$$0 \geq \sum_{j=1}^{\psi} R_j(\vec{x}, a) + \sum_i w_i \left(\gamma \sum_{\vec{x}' \in \mathcal{S}} P(\vec{x}' | \vec{x}, a) h_i(\vec{x}') - h_i(\vec{x}) \right). \tag{4.8}$$

A idéia básica é substituir o conjunto de restrições em (4.8) (uma para cada par estado-ação) por um conjunto equivalente de restrições não-lineares (não-linear pelo operador $\max_{\vec{x}}$) para $\forall a \in \mathcal{A}$, dado por:

⁴Embora essa hipótese pareça ser muito restritiva, na prática existe um conjunto grande de aplicações em que ela pode ser feita [Guestrin, 2003].

$$0 \geq \max_{\vec{x}} \left\{ \sum_{j=1}^{\psi} R_j(\vec{x}, a) + \sum_i w_i \left(\gamma \sum_{\vec{x}' \in \mathcal{S}} P(\vec{x}' | \vec{x}, a) h_i(\vec{x}') - h_i(\vec{x}) \right) \right\}. \quad (4.9)$$

Assim, para uma ação a , temos que calcular a maximização a seguir:

$$0 \geq \max_{\vec{x}} \left\{ \sum_{j=1}^{\psi} R_j(\vec{x}) + \sum_i w_i \left(\gamma \sum_{\vec{x}' \in \mathcal{S}} P(\vec{x}' | \vec{x}) h_i(\vec{x}') - h_i(\vec{x}) \right) \right\}. \quad (4.10)$$

Guestrin também emprega as idéias da técnica de ELIMINAÇÃO DE VARIÁVEIS e faz a maximização sobre cada variável de estado, uma por uma, criando um novo conjunto de restrições e variáveis, em vez de somar todos os termos e fazer a maximização sobre todos os estados. Assim, a restrição não-linear (Equação (4.10)) é transformada novamente em restrições lineares ao custo de introduzir variáveis extra, mas com a vantagem de que um conjunto menor de restrições é criado para cada ação.

Existem outros algoritmos eficientes que usam técnicas gerais para resolver problemas lineares com um número grande de restrições [Patrascu, 2004, de Farias and Roy, 2004, Dolgov and Durfee, 2006] (e.g., Geração de Restrições) que são extensões do trabalho original de Guestrin [Guestrin, 2003].

4.5 Resumo

Neste capítulo, apresentamos uma representação fatorada de MDPs e algoritmos que evitam a enumeração explícita dos estados. Esses algoritmos exploram diferentes formas de estruturar o problema, tais como: 1) utilização de estruturas de dados como ADDs para representar independências do tipo CSI e estruturas compartilhadas nas funções e 2) combinações lineares de funções base para aproximar a função valor e explorar essa estrutura aditiva na solução, ou seja a estrutura da aproximação por soma de funções base.

Os algoritmos apresentados neste capítulo foram a inspiração para os algoritmos propostos nessa tese. Como veremos no Capítulo 7, SPUDD e APRICODD podem ser generalizados para resolver MDP-IPs fatorados. Adicionalmente, no Capítulo 8 é apresentado um novo algoritmo FACTOREDMP que generaliza o algoritmo FACTOREDLP, proposto por Guestrin [Guestrin, 2003], para MDP-IPs.

Capítulo 5

Processo de Decisão Markoviano com Probabilidades Imprecisas

Conforme descrito no exemplo introdutório sobre tráfego [Delgado et al., 2009c], muitas vezes é necessário trabalhar com probabilidades imprecisas a fim de representar crenças incompletas, ambíguas ou contraditórias do especialista sobre as probabilidades de transição. Um *MDP com Probabilidades de Transição Imprecisas* (MDP-IP) é projetado especificamente para essa situação¹. Neste capítulo, definimos formalmente um MDP-IP e mostramos também as soluções existentes na literatura para esse tipo de problema.

5.1 MDP-IP Enumerativo: Definição

Os MDP-IPs [Satia and Lave Jr., 1973, White III and El-Deib, 1994] enumerativos representam uma extensão dos MDPs enumerativos (Capítulo 3), nos quais existe imprecisão nas probabilidades de transição. Para especificar um MDP-IP enumerativo, devemos especificar todos os elementos de um MDP, exceto as probabilidades de transição. Em seguida, deve-se especificar um conjunto de probabilidades para cada transição entre estados, isto é, para cada tripla $\langle s, s', a \rangle$. Referimo-nos a esses conjuntos como *conjuntos credais de transição*. Formalmente, um MDP-IP é definido por uma tupla da forma $\mathcal{M}_{IP} = \langle S, A, R, K, \gamma \rangle$ em que:

- S é um conjunto discreto e finito de estados completamente observáveis que modelam o mundo;
- A é um conjunto finito de ações; a execução de uma ação permite que o sistema mude do estado atual para o próximo estado. Assim, a tarefa de planejamento é escolher a melhor ação para ser executada num determinado estado;
- $R : S \times A \rightarrow \mathbb{R}$ é a função recompensa associada a cada par $\langle \text{estado}, \text{ação} \rangle$ e que representa as preferências do agente.
- $K(s'|s, a)$ define um conjunto de medidas de probabilidade de transição válidas, i.e., um conjunto credal de transição (*transition credal sets*). $K(s'|s, a)$ pode ser implicitamente representado por um conjunto de probabilidades de transição consistente com um conjunto de

¹O termo MDP-IP, foi proposto por White III e Eldeib [White III and El-Deib, 1994], enquanto Satia e Lave Jr. [Satia and Lave Jr., 1973] adotaram o termo *MDP com incerteza nas probabilidades de transição*.

restrições lineares, i.e, assumimos que P é parametrizado por valores reais $\vec{p}$ e que temos um conjunto $\mathfrak{C}$ de equações e inequações lineares sobre $\vec{p}$ que definem o conjunto de probabilidades de transição lícitas, isto é, $K(s'|s, a) = \{P(s'|s, a, \vec{p}) \mid \vec{p} \text{ satisfaz } \mathfrak{C}\}$. Por exemplo, sejam:

$$\begin{aligned} P(s'_1|s_0, a_1) &= p_1 \\ P(s'_2|s_0, a_1) &= p_2 \\ &\dots \\ P(s'_n|s_0, a_1) &= p_n \end{aligned}$$

o conjunto de probabilidades de transição lícitas satisfaz o conjunto $\mathfrak{C}$ de equações e inequações a seguir:

$$\begin{aligned} p_1 + p_2 + \dots + p_n &= 1 \\ p_2 &> p_5 \\ &\dots \end{aligned}$$

- γ é o fator de desconto.

Note que a definição de MDP-IP é a mesma dada para MDP (Seção 3.1), exceto que a distribuição de transição P é substituída pelo *conjunto* de distribuições K .

Existem duas possíveis formulações para MDP-IP: (1) baseada em Teoria de Jogos, em que o MDP é formulado como um jogo entre duas entidades: o agente e a Natureza e (2) Bayesiana em que as distribuições de probabilidade *a priori* sobre P são atualizadas com novas observações usando a regra de Bayes. Neste trabalho, estamos interessados na formulação baseada em Teoria de Jogos, uma vez que esta é a formulação mais utilizada na literatura. A formulação Bayesiana será discutida no Capítulo 11 sobre Trabalhos Correlatos.

Na formulação baseada em Teoria de Jogos, dadas as características de um MDP-IP, é possível definir vários critérios de otimização para avaliar uma política, entre eles:

- Critério maximin [Satia and Lave Jr., 1973]: a política ótima é aquela que produz a maior recompensa acumulada esperada (com fator de desconto), considerando-se as escolhas menos favoráveis de distribuições de probabilidades (melhor escolha no pior caso).
- Critério maximax [Satia and Lave Jr., 1973]: a política ótima é aquela que produz a maior recompensa acumulada esperada (com fator de desconto), considerando-se as escolhas mais favoráveis de distribuições de probabilidades (melhor escolha no melhor caso).
- Critério maximix [Kikuti et al., 2005]: a política que produz o máximo de $\alpha(\max_P V_\pi) + (1 - \alpha)(\min_P V_\pi)$, para algum $\alpha \in (0, 1)$.
- Critério admissibilidade: qualquer política que maximize a recompensa acumulada esperada com fator de desconto, para pelo menos uma escolha das probabilidades de transição.

Nesse trabalho, estamos interessados em políticas que levem o agente para o maior valor de $V(s)$, assumindo que as “escolhas da Natureza” pretendem minimizar a recompensa esperada do agente, isto é, o critério maximin. O critério maximin é o critério mais considerado na área de Inteligência Artificial e que oferece uma abordagem razoável quando procuramos por políticas robustas (maximizar a recompensa no pior caso). Satia e Lave [Satia and Lave Jr., 1973] mostraram vários resultados importantes para MDP-IPs de horizonte infinito que adotam esse critério, a saber:

- Existe sempre uma política estacionária determinística que é ótima segundo esse critério [Satia and Lave Jr., 1973].
- Dada a suposição de que A é finito e o conjunto credal K é fechado², essa política ótima induz a uma função valor ótima V^* , que é a única solução do ponto fixo de:

$$V^*(s) = \max_{a \in A} \min_{P \in K} \left\{ R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a, \vec{p}) V^*(s') \right\}. \quad (5.1)$$

Na Equação (5.1) o agente escolhe a ação a que maximiza o valor de $V(s)$ e a Natureza escolhe a probabilidade $P \in K$ que minimiza $V(s)$. A idéia da solução mostrada na Equação (5.1) para MDP-IP é ser a mais robusta possível frente à incerteza na transição dada por K . A política da Natureza não é necessariamente estacionária, i.e, ela pode mudar ao longo do tempo. No exemplo do tráfego [Delgado et al., 2009c], observamos que os padrões de tráfego podem ser diferentes nos feriados, quando comparados com os dias de semana normal e, apesar do sistema não modelar explicitamente essa informação na sua descrição do estado, o sistema deve ainda ser robusto nesse cenário .

MDP-IP: Modelo Conceitual para o agente maximin *Um MDP-IP modela a interação entre um agente e seu ambiente. A cada instante o agente faz uma escolha de ações (com efeitos probabilísticos) e decide executar uma ação que produzirá um estado futuro e uma recompensa que depende também das escolhas da Natureza. O objetivo do agente é maximizar a recompensa ganha ao longo de uma sequência de escolhas de ações assumindo que a escolha da Natureza minimiza sua recompensa.*

5.1.1 MDP-IP: Solução baseada em programação dinâmica

Existem vários algoritmos para resolver MDP-IPs enumerativos baseados em programação dinâmica [Satia and Lave Jr., 1973, White III and El-Deib, 1994]. A solução de ITERAÇÃO DE VALOR dada por Satia e Lave [Satia and Lave Jr., 1973] é definida por:

$$V^t(s) = \max_{a \in A} \min_{P \in K} \left\{ R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a, \vec{p}) V^{t-1}(s') \right\}, \quad (5.2)$$

que corresponde àquela dada nas Equações (3.7) e (3.8) para MDPs, exceto que, para cada estado s , otimizamos nossa escolha de ação $a \in A$ com respeito ao pior caso da distribuição $P \in K$, i.e.,

²Um conjunto é fechado se e só se contém a sua fronteira.

que minimiza o valor esperado futuro. Assim, poderemos garantir que a política é robusta para o pior resultado que a Natureza pode escolher em função do valor futuro $V^{t-1}(s')$ que esperamos atingir.

Notamos que a função da Natureza P pode ser não-estacionária; a Natureza pode escolher um $P \in K$ diferente para cada ação a e cada estado s , a cada estágio t . Propriedades de convergência da ITERAÇÃO DE VALOR em (5.2) ainda se mantêm quando as transições da Natureza são não estacionárias mas limitadas por $P \in K$ [Nilim and El Ghaoui, 2005].

5.1.2 MDP-IP: Solução baseada em programação matemática

A princípio, não parece ser possível reduzir a solução da Equação (5.1) para programação linear (como no Problema (3.14)). No entanto, em Shirota et al [Shirota et al., 2007] foi mostrado que é possível gerar soluções usando problemas de programação bem conhecidos. Primeiro, a Equação (5.1) pode ser reduzida para *programação de dois níveis* [Colson et al., 2007], simplesmente considerando P como variável e introduzindo no Problema (3.14) mais um nível de otimização para fazer a minimização sobre a variável P .

$$\begin{aligned} \min_{V^*} & : \sum_s V^*(s) \\ \text{s.a} & : V^*(s) \geq R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a, \vec{p}) V^*(s'), \forall s \in S, a \in A; \\ & P \in \arg \min \sum_{s' \in S} P(s'|s, a, \vec{p}) V^*(s'), \\ & \text{s.a} : P(s'|s, a, \vec{p}) \in K(s'|s, a). \end{aligned} \quad (5.3)$$

O programa de dois níveis (5.3) pode ser ainda transformado em um *programa multilinear* [Drenick, 1992] equivalente:

$$\begin{aligned} \min_{V^*, P} & : \sum_s V^*(s) \\ \text{s.a} & : V^*(s) \geq R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a, \vec{p}) V^*(s'), \forall s \in S, a \in A \\ & P(s'|s, a, \vec{p}) \in K(s'|s, a). \end{aligned} \quad (5.4)$$

Lema 5.1. *O Problema (5.3) e o Problema (5.4) produzem a função valor ótima, $V^*(s)$.*

Prova: Para verificar que o Problema (5.3) produz $V^*(s)$, isto é, que o Problema (5.3) encontra a solução única de ponto fixo da Equação (5.1), $V^*(s)$, usamos as restrições para forçar que $V^*(s)$ seja maior ou igual que:

$$\max_a \{ R(s, a) + \gamma \min_P \sum_{s' \in S} P(s'|s, a, \vec{p}) V^*(s') \},$$

Figura 5.1: Relações entre MDP-IP e suas subclasses [Trevizan et al., 2007].

e em seguida minimizamos $\sum_s V^*(s)$ para obrigar que o mínimo $V^*(s)$ (na igualdade) seja obtido. Uma vez que os Problemas (5.3) e (5.4) são equivalentes [Shirota et al., 2007], também podemos afirmar que resolvendo o programa multilinear (5.4) encontramos $V^*(s)$. ■

Note que a solução de programas multilineares está longe de ser trivial para problemas grandes. Assim, apenas MDP-IPs relativamente pequenos podem ser tratados diretamente desta maneira [Shirota et al., 2007].

5.2 Subclasses de MDP-IP

Há duas subclasses especiais de MDP-IP que são importantes de se mencionar: *Bounded-parameter Markov Decision Process (BMDP)* [Givan et al., 2000] e *Markov Decision Process with Set-valued Transitions (MDPSTs)* [Trevizan et al., 2007]. A primeira é um caso especial de MDP-IP³, em que as probabilidades são especificadas por intervalos, por exemplo, $P(s_2|s_1, a_1) \in [1/3, 1/2]$. Num MDPST, as distribuições de probabilidade são dadas sobre conjuntos finitos de estados, por exemplo, $P(\{s_2, s_3, s_4\}|s_1, a_1) = 0.8$. Em [Trevizan et al., 2007] foi provado que um MDPST pode ser transformado em um MDP-IP, construindo-se um conjunto de restrições ($\mathcal{C}$) sobre as probabilidades de transição para cada estado do conjunto $\mathcal{B}$ em $P(\mathcal{B}|s, a)$. Algoritmos especializados baseados em programação dinâmica foram propostos para resolver essas subclasses de MDP-IPs [Givan et al., 2000, Trevizan et al., 2007].

Na Figura 5.1 [Trevizan et al., 2007] mostramos as relações entre os modelos MDP, MDP-IP, BMDP e MDPST. Note que o modelo mais geral é MDP-IP, em que estamos interessados neste trabalho. Note ainda que ambas subclasses unificam as abordagens de planejamento determinístico (DET), probabilístico (MDP) e não determinístico (NOT-DET).

5.3 Resumo

Neste capítulo apresentamos MDP-IPs enumerativos e a formulação baseada em Teoria de Jogos. Nessa formulação é possível definir vários critérios de otimização para avaliar uma política, entre eles o critério maximin, que é um critério bastante usado uma vez que produz uma política robusta.

³Um BMDP é um MDP em que as probabilidades de transição e as recompensas são especificadas por intervalos. Uma vez que as recompensas em BMDPs são imprecisas, BMDPs e MDP-IPs são incomparáveis. Neste trabalho consideramos BMDPs com valores de recompensa precisa, que podem ser considerados como um caso especial de um MDP-IP.

Além disso, mostramos as principais técnicas de solução baseadas em programação dinâmica e programação matemática para resolver MDP-IPs. Notamos que ambas as soluções não são triviais e apenas pequenos problemas podem ser resolvidos. Para remediar esta situação, na Parte II é introduzida uma definição de MDP-IP fatorado e são apresentadas as soluções eficientes propostas nessa tese.

Parte II

Soluções Eficientes para MDP-IP

Capítulo 6

MDP-IP Fatorado

A primeira contribuição desse trabalho estende a representação de um MDP fatorado [Boutilier et al., 1999] (Seção 4.1) para representar MDP-IPs de forma compacta. No Capítulo 5 vimos que um MDP-IP pode ser definido através de modelos enumerativos. Porém, como visto no Capítulo 4, é mais natural pensar no estado como uma atribuição de múltiplas variáveis de estado que descrevem um domínio de aplicação e uma função de transição que especifique a dependência probabilística, do que enumerar o conjunto total de estados. Assim, como um MDP fatorado, um *MDP-IP fatorado* é definido em termos de estados $\vec{x} \in \{0, 1\}^n$, especificados como uma atribuição conjunta para um vetor de n variáveis binárias de estado $(X_1, \dots, X_n)$.¹

Antes de descrever a função de transição de estados em um MDP-IP fatorado, modelamos um exemplo de problema no domínio SYSADMIN, com base no Exemplo 4.1 (Capítulo 4). Para isso modificamos o MDP fatorado anterior permitindo probabilidades de transição imprecisas.

Exemplo 6.1. SysAdmin Como no Exemplo 4.1 (Seção 4.1) no domínio SYSADMIN [Guestrin et al., 2003] temos n computadores $c_1, \dots, c_n$ conectados através de diferentes topologias de grafos dirigidos: (a) anel unidirecional, (b) anel bidirecional e (c) anéis bidirecionais independentes de pares de computadores e (d) estrela (Figure 4.2).

A variável x_i indica se um computador c_i está ligado e funcionando ($X_i = 1$) ou não ($X_i = 0$). $Conn(c_j, c_i)$ denota a conexão de c_j para c_i . Formalmente, as CPTs na DCN de transição para este domínio tem a seguinte forma:

$$P(X'_i = 1 | \vec{x}_i, a) = \begin{cases} \text{se } a = \text{reboot}(c_i) & \text{então } 1. \\ \text{se } a \neq \text{reboot}(c_i) \wedge x_i = 1 & \text{então} \\ & p_{i1} \cdot \frac{|\{x_j | j \neq i \wedge x_j = 1 \wedge Conn(c_j, c_i)\}| + 1}{|\{x_j | j \neq i \wedge Conn(c_j, c_i)\}| + 1}. \\ \text{se } a \neq \text{reboot}(c_i) \wedge x_i = 0 & \text{então} \\ & p_{i2} \cdot \frac{|\{x_j | j \neq i \wedge x_j = 1 \wedge Conn(c_j, c_i)\}| + 1}{|\{x_j | j \neq i \wedge Conn(c_j, c_i)\}| + 1}. \end{cases} \quad (6.1)$$

¹Enquanto nossas extensões não são necessariamente restritas a variáveis de estado binárias, fazemos esta restrição aqui para simplificar a notação.

e as restrições nas variáveis de probabilidade são:

$$0.85 + p_{i2} \leq p_{i1} \leq 0.95 \quad (6.2)$$

Temos $n + 1$ ações: $\text{reboot}(c_1), \dots, \text{reboot}(c_n)$ e $\text{notreboot}()$. A ação $\text{notreboot}()$ indica que nenhuma máquina foi reiniciada. Se um computador é reiniciado, a sua probabilidade de funcionar no instante seguinte é 1, caso contrário a probabilidade depende do seu estado atual e do número de computadores conectados a ele que também estão funcionando no estado atual. As variáveis de probabilidade p_{i1} , p_{i2} , e as restrições sobre elas, definem os conjuntos credais. p_{i1} é um número entre 0.85 e 0.95 para representar que a probabilidade de um computador continuar funcionando é alta e p_{i2} é um número entre 0 e 0.1 para representar que a probabilidade de um computador passar a funcionar é baixa.

Além disso, a recompensa é 1 se todos os computadores estão funcionando em qualquer instante do tempo, caso contrário, a recompensa é 0, i.e., $R(\vec{x}) = \prod_{i=1}^n \mathbb{I}[X_i = 1]$. $\diamond$

Note que as diferenças entre o Exemplo 4.1 e o Exemplo 6.1 são as variáveis de probabilidade p_{ij} e o conjunto de restrições sobre elas.

6.1 MDP-IP Fatorado: Definição

Como na definição anterior de MDP-IP com estados enumerativos, o conjunto de todas as distribuições de transição válidas para um MDP-IP fatorado está definido por um *conjunto credal* K . O desafio, então, é especificar tais conjuntos credais de transição de forma fatorada e compacta. Para isso, usamos *Redes Credais Dinâmicas* (DCNs - *Dynamic Credal Network*) [Cozman, 2000, Cozman, 2005a], como visto no Capítulo 2.

Um exemplo de DCN para um MDP-IP fatorado é mostrado na Figura 6.1.a. Uma DCN tem a mesma estrutura de duas camadas como a Rede Bayesiana Dinâmica (DBN) [Dean and Kanazawa, 1990] apresentada na Seção 4.1, a diferença é que para cada variável X'_i na DCN, temos uma *Tabela de Probabilidade Condicional (CPT)* com probabilidades imprecisas. Se examinarmos as CPTs na Figura 6.1.b, notamos que as probabilidades na última coluna são especificadas por parâmetros p_{ij} (i para variável X'_i , j para o parâmetro j -ésimo na CPT para X'_i). Além disso, temos um conjunto de restrições lineares nestes p_{ij} (mostrados na parte inferior da Figura 6.1.b). Chamemos esse conjunto de $\mathfrak{C}$. Assim, dado $\vec{p} = (\dots, p_{ij}, \dots)$ consistente com as restrições lineares $\mathfrak{C}$, obtemos uma distribuição de transição válida de K , em que $pa_a(X'_i)$ são os pais de X'_i na DCN para a ação $a \in A$:

$$P(\vec{x}' | \vec{x}, a, \vec{p}) = \prod_{i=1}^n P(x'_i | pa_a(X'_i), a, \vec{p}). \quad (6.3)$$

Note que a probabilidade de transição conjunta pode ser não-linear nos parâmetros $\vec{p}$. No entanto, uma vez que restringimos p_{ij} para aparecer somente na CPT de X'_i (por definição), evitamos a multiplicação de p_{ij} por si próprio. Esse subconjunto de expressões não-lineares, em que o expoente de cada p_{ij} é 0 ou 1, é definido como uma expressão *multilinear*. Essa é uma

Figura 6.1: a) Rede Credal Dinâmica para ação *notreboot* para uma topologia de anel unidirecional do domínio SYSADMIN com dois computadores. b) Tabela de probabilidade condicional para as variáveis $X'_1 = 1$ e $X'_2 = 1$ e as restrições relacionadas às probabilidades. c) A representação com ADD Parametrizado da probabilidade de transição chamada de $P_{DD}(x'_1|x_1, x_2, notreboot)$. A linha contínua indica o ramo verdadeiro (1) da variável de teste e a linha tracejada indica o ramo falso (0).

característica importante, uma vez que existem técnicas que resolvem o programa multilinear de maneira exata (em contraposição com a programação não-linear geral) [de Campos, 2005]. Um exemplo da multilinearidade da probabilidade de transição pode ser visto na Figura 6.1, em que $P(X'_1 = 1, X'_2 = 1|X_1 = 1, X_2 = 1, notreboot, \vec{p}) = p_{11}p_{21}$.

Dada uma DCN, podemos agora definir o conjunto credal de transição fatorado

$$K(x'_i|pa(X'_i), a) = \{P(x'_i|pa(X'_i), a, \vec{p}) | \vec{p} \text{ satisfaz } \mathfrak{C}\}.$$

Como nas Redes Bayesianas Dinâmicas da Seção 4.1, ainda que possamos representar qualitativamente as propriedades de independência condicional de uma distribuição usando DCNs, há certas independências que não podemos representar com a estrutura de uma Rede Credal. A fim de representar de forma compacta funções com independência do tipo CSI e estruturas compartilhadas para um MDP-IP, propomos uma nova extensão de Diagramas de Decisão Algébrica (ADDs) [Bahar et al., 1993] chamados *ADDs Parametrizados* (PADDs), uma vez que as folhas são expressões parametrizadas, como mostramos na Figura 6.1.c. PADDs não só permitem representar de forma compacta as CPTs de uma DCN para MDP-IPs fatorados, mas também permitem cálculos eficientes durante a ITERAÇÃO DE VALOR em um MDP-IP fatorado, como mostraremos a seguir.

Além de definir a função de transição para um MDP-IP fatorado, podemos definir a função recompensa fatorada como definida na Seção 4.1.1:

$$R(\vec{x}, a) = \sum_{j=1}^{\psi} R_j(\vec{x}, a). \quad (6.4)$$

Figura 6.2: a) Tabela de probabilidade condicional para a variável de estado X'_2 para a ação a_1 . b) A representação ADD Parametrizada para $P(X'_2 = 1 | x_1, x_2, x_3, x_4, a_1)$.

6.2 Diagramas de Decisão Algébrica Parametrizados

Nessa tese, propomos a estrutura *Parameterized Algebraic Decision Diagrams* que chamaremos nesse texto de ADDs Parametrizados, ou simplesmente de PADDs, que estendem os ADDs vistos na Seção 4.2 e que permitem uma representação compacta de funções de $\{0, 1\}^n$ para o espaço das expressões parametrizadas por $\vec{p}$ (no nosso caso, ainda restringimos isso para o espaço das expressões multilineares de $\vec{p}$). Por exemplo, a CPT para X'_2 representado na Figure 6.2 como um PADD, contém folhas envolvendo apenas um parâmetro. A seguir, definimos formalmente ADDs Parametrizados e suas operações básicas.

6.2.1 PADD: Definição formal e propriedades

PADDs generalizam as folhas de valores reais constantes de ADDs para polinômios (*Poly*) expressos como uma forma canônica de soma de produtos. A gramática BNF para *Poly*, em que d_i representa constantes e p_{ij} parâmetros, envolve mais um elemento, dado por:

$$Poly ::= d_0 + \sum_i d_i \prod_j p_{ij}. \quad (6.5)$$

ou seja, a expressão nas folhas de um PADD é uma expressão multilinear sobre os parâmetros p_{ij} . Formalmente, podemos definir um PADD F com a gramática BNF a seguir:

$$F ::= Poly | if(F^{var}) \text{ then } F_h \text{ else } F_l. \quad (6.6)$$

ou seja, um nó do diagrama de decisão pode ser simplesmente uma expressão do tipo *Poly* ou uma variável que se divide em dois ramos, F_h e F_l , onde F_h vale para $var = 1$ e F_l vale para $var = 0$.

O valor de um PADD F contendo um conjunto de variáveis $\{x_1, \dots, x_n\}$ na atribuição de valores

das variáveis $\rho \in \{0, 1\}^n$ podem ser definidas recursivamente por:

$$Val(F, \rho) = \begin{cases} \text{se } F = Poly & \text{então } Poly. \\ \text{se } F \neq Poly \wedge \rho(F^{var}) = true & \text{então } Val(F_h, \rho). \\ \text{se } F \neq Poly \wedge \rho(F^{var}) = false & \text{então } Val(F_l, \rho). \end{cases}$$

Como visto na Seção 4.2 para ADDs, dada uma função $f(x_1, \dots, x_n)$ e uma ordenação fixa de variáveis sob $x_1, \dots, x_n$, um PADD reduzido é definido como o menor diagrama de decisão ordenado que representa a função f .

Lema 6.1. *Existe um único PADD reduzido F (representação canônica) satisfazendo uma dada ordenação das variáveis, de tal forma que para todo $\rho \in \{0, 1\}^n$ temos $f(\rho) = Val(F, \rho)$.*

Prova: A prova deste lema para BDDs (Binary Decision Diagrams) foi fornecida por [Bryant, 1986] e pode ser trivialmente generalizada para ADDs e PADDs. Intuitivamente, uma vez que PADDs permitem representar folhas com polinômios, a única mudança para demonstrar este lema é que temos que garantir que existe uma maneira de identificar quando duas expressões nas folhas são idênticas, o que pode ser feito facilmente pela (a) ordenação dos parâmetros em cada termo multilinear, (b) fatoração e soma das constantes nos termos multilineares idênticos, e (c) ordenação da lista de termos de acordo com o índice mais baixo da variável e o número de parâmetros. Com esse método de identificação de folhas únicas, a prova de [Bryant, 1986] generaliza-se para PADDs e mostra que existe uma única representação PADD canônica para cada função de $\{0, 1\}^n$ para polinômios em p_{ij} , na forma de (6.5). ■

Nos algoritmos mostrados a seguir usamos três tabelas de *hash*: *ReduceCache*, *NodeCache* e *ApplyCache*. Usaremos a notação *chave* $\rightarrow$ *value* para representar a *chave* e a informação associada, *value*, nas tabelas de hash. A tabela de hash *ReduceCache* armazena nós canônicos, a tabela *NodeCache* armazena identificadores únicos para cada nó existente e a tabela *ApplyCache* armazena os resultados de operações anteriores APPLY.

6.2.2 Algoritmo para reduzir PADDs

Dado que o Lema 6.1 garante que existe uma única forma canônica para qualquer função expressa como um PADD, o algoritmo REDUCEPADD realmente permite a construção eficiente da representação canônica de um PADD, tendo como entrada um diagrama de decisão ordenado arbitrário com polinômios nas folhas e como saída o PADD reduzido.

O Algoritmo 3 recursivamente constrói um PADD reduzido num processo *bottom-up*. Neste algoritmo, um nó interno é representado por $\langle var, F_h, F_l \rangle$, em que *var* é o nome da variável, e F_h e F_l são os identificadores dos nós dos ramos verdadeiro e falso, respectivamente. Adicionalmente, F refere-se a um nó arbitrário (interno ou terminal), enquanto F_r refere-se a um identificador do nó canônico. Após serem reduzidos, nós canônicos são armazenados na tabela de hash *ReduceCache* e a função auxiliar GETNODE (Algoritmo 4) garante que testes de decisão redundantes em nós internos sejam removidos. A tabela *NodeCache* usada na função GETNODE armazena identificadores únicos para cada nó existente.

Algoritmo 3: REDUCEPADD(F)

```

input :  $F$  (root node id for an arbitrary ordered decision diagram)
output:  $F_r$  (root node id for reduced PADD)

1 begin
2   //if terminal node, return canonical terminal node
3   if  $F$  is terminal node then
4     return canonical terminal node for polynomial of  $F$ ;
5   //use recursion to reduce sub diagrams
6   if  $F \rightarrow F_r$  is not in ReduceCache then
7      $F_h = \text{REDUCEPADD}(F_h)$ ;
8      $F_l = \text{REDUCEPADD}(F_l)$ ;
9     //get a canonical internal node id
10     $F_r = \text{GETNODE}(F^{var}, F_h, F_l)$ ;
11    insert  $F \rightarrow F_r$  in ReduceCache;
12  return  $F_r$ ;
13 end

```

Um exemplo de aplicação do algoritmo REDUCEPADD (Algoritmo 3) é mostrado na Figura 6.3. A seta vazia aponta o nó interno F que está sendo avaliado por REDUCEPADD após as duas chamadas recursivas (linhas 7 e 8) e antes da linha 10: $F_r = \text{GETNODE}(F^{var}, F_h, F_l)$. A Figura 6.3.a mostra o diagrama de entrada para o algoritmo em que a variável x_3 está sendo avaliada por REDUCEPADD. A Figura 6.3.b mostra o diagrama resultante depois das chamadas recursivas: REDUCEPADD(F_h) e REDUCEPADD(F_l); note que REDUCEPADD(F_h) devolveu o mesmo identificador para o nó terminal $0.3 + 5 * p_{12}$ que existia anteriormente. A Figura 6.3.c mostra o diagrama resultante após as duas chamadas recursivas; note que F_h e F_l para x_2 são iguais e assim, depois que GETNODE é chamado, F_l é devolvido e x_2 desaparece. A Figura 6.3.d mostra a representação canônica final para a entrada.

O tempo de execução e o espaço usado por REDUCEPADD são lineares no tamanho do diagrama de entrada, uma vez que o uso da tabela de hash *ReduceCache* garante que cada nó seja visitado apenas uma vez e, um único nó reduzido é gerado no diagrama canônico para cada nó visitado.

6.2.3 Operações binárias em PADDs

É simples estender as operações unárias e binárias de ADDs para serem aplicadas para PADDs. Isto exige apenas que as operações sobre as folhas sejam modificadas para aceitar e produzir polinômios resultantes na forma definida em (6.5). Notamos que as operações unárias, tais como a marginalização ($\sum_{x_i \in X_i} F$), são realmente implementadas usando operações binárias. Assim, faremos uma discussão sobre operadores binários a seguir.

Para *todas* as operações binárias, usamos a função genérica APPLY (Algoritmo 5) e a função auxiliar COMPUTERESULT (implementada na Tabela 6.1) que realiza operações entre nós de PADD arbitrários com polinômios nas folhas. A notação usada nestes algoritmos é mostrada na Figura 6.4. Note que $F_{i,h}$ e $F_{i,l}$ representam sub-diagramas.

Algoritmo 4: GETNODE($\langle var, F_h, F_l \rangle$)

```

input :  $\langle var, F_h, F_l \rangle$  (variable and true and false branches node ids for internal node)
output:  $F_r$  (canonical internal node id)

1 begin
2   //redundant branches
3   if  $F_l = F_h$  then
4     | return  $F_l$ ;
5   //check if the node exists previously
6   if  $\langle var, F_h, F_l \rangle \rightarrow id$  is not in NodeCache then
7     | id = new unlocated id;
8     | insert  $\langle var, F_h, F_l \rangle \rightarrow id$  in NodeCache;
9   return  $id$ ;
10 end

```

Figura 6.3: Um exemplo de aplicação do algoritmo REDUCEPADD (Algoritmo 3). a) Diagrama de entrada para o algoritmo. b) e c) Diagramas intermediários. d) Representação canônica para a entrada.

Figura 6.4: Notação usada nos algoritmos APPLY e CHOOSEVARBRANCH.

O algoritmo APPLY tem como entrada dois operandos, representados como PADDs (F_1, F_2), e um operador binário $op \in \{\oplus, \ominus, \otimes\}$, definidos como o operador soma, diferença e multiplicação entre PADDs; a saída é o resultado da aplicação da função, também representada como um PADD canônico (F_r). APPLY verifica primeiro se o resultado pode ser calculado imediatamente chamando a função *ComputeResult*. Se o resultado é *null*, então verifica se o resultado foi anteriormente calculado examinando a tabela de hash *ApplyCache*, que armazena os resultados de operações

Algoritmo 5: APPLY(F_1, F_2, op)

```

input :  $F_1$  (root node id for operand 1),
         $F_2$  (root node id for operand 2),
         $op$  (binary operator,  $op \in \{\oplus, \ominus, \otimes\}$ )
output:  $F_r$  (root node id for resulting PADD)
1 begin
2   //check if the result can be immediately computed
3   if COMPUTERESULT( $F_1, F_2, op$ )  $\rightarrow F_r \neq null$  then
4     | return  $F_r$ ;
5   //check if we previously computed the same operation
6   if  $\langle F_1, F_2, op \rangle \rightarrow F_r$  is not in ApplyCache then
7     | //choose variable to branch
8     |  $var = \text{CHOOSEVARBRANCH}(F_1, F_2)$ ;
9     | //set up nodes for recursion
10    | if  $F_1$  is non-terminal  $\wedge var = F_1^{var}$  then
11    | |  $F_l^{v1} = F_{1,l}$ ;
12    | |  $F_h^{v1} = F_{1,h}$ ;
13    | else
14    | |  $F_{l,h}^{v1} = F_1$ ;
15    | if  $F_2$  is non-terminal  $\wedge var = F_2^{var}$  then
16    | |  $F_l^{v2} = F_{2,l}$ ;
17    | |  $F_h^{v2} = F_{2,h}$ ;
18    | else
19    | |  $F_{l,h}^{v2} = F_2$ ;
20    | //use recursion to compute true and false branches for resulting PADD
21    |  $F_l = \text{Apply}(F_l^{v1}, F_l^{v2}, op)$ ;
22    |  $F_h = \text{Apply}(F_h^{v1}, F_h^{v2}, op)$ ;
23    |  $F_r = \text{GETNODE}(var, F_h, F_l)$ ;
24    | //save the result to reuse in the future
25    | insert  $\langle F_1, F_2, op \rangle \rightarrow F_r$  into ApplyCache;
26  return  $F_r$ ;
27 end

```

anteriores APPLY. Se não houver um elemento no *ApplyCache* que coincida, APPLY escolhe as primeiras variáveis na ordenação para bifurcar chamando CHOOSEVARBRANCH (Algoritmo 6) e então ramifica nesta variável com duas chamadas recursivas para APPLY, uma para calcular F_l e outra para calcular F_h .

Note que, enquanto o algoritmo REDUCEPADD que constrói o ADD canônico usa a estratégia *bottom-up*, o algoritmo APPLY que realiza a operação entre dois ADDs, usa uma estratégia *top-down* aplicando um operador op para cada variável comum entre dois PADDs.

Um exemplo de multiplicação de PADDs através de *Apply* é mostrado na Figura 6.5. É importante notar que as operações binárias $\oplus$, $\ominus$, e $\otimes$ são *fechadas* para PADDs com folhas na forma definida em (6.5), ou seja, essas operações em PADDs produzem novos PADDs com folhas que

Algoritmo 6: CHOOSEVARBRANCH(F_1, F_2)

```

input  :  $F_1$  (root node id for operand 1),
           $F_2$  (root node id for operand 2)
output:  $var$  (selected variable to branch)
1 begin
2   //select the variable to branch based on the order criterion
3   if  $F_1$  is a non-terminal node then
4       if  $F_2$  is a non-terminal node then
5           if  $F_1^{var}$  comes before  $F_2^{var}$  then
6                $var = F_1^{var}$ ;
7           else
8                $var = F_2^{var}$ ;
9       else
10           $var = F_1^{var}$ ;
11      else
12           $var = F_2^{var}$ ;
13      return  $var$ ;
14  end

```

Case	Return
$F_1 \text{ op } F_2; F_1 = Poly_1; F_2 = Poly_2$	$Poly_1 \text{ op } Poly_2$
$F_1 \oplus F_2; F_2 = 0$	F_1
$F_1 \oplus F_2; F_1 = 0$	F_2
$F_1 \ominus F_2; F_2 = 0$	F_1
$F_1 \otimes F_2; F_2 = 1$	F_1
$F_1 \otimes F_2; F_1 = 1$	F_2
$F_1 \otimes F_2; F_2 = 0$	0
$F_1 \otimes F_2; F_1 = 0$	0
other	<i>null</i>

Tabela 6.1: Casos de entrada e resultados do método COMPUTERESULT para as operações binárias $\oplus$, $\ominus$ e $\otimes$ para PADDs.

podem ser também expressas como na forma BNF (Equação 6.5). PADDs não são fechados para divisão binária, $\min(\cdot, \cdot)$, ou $\max(\cdot, \cdot)$. Felizmente, essas operações não serão necessárias na nossa proposta de solução para MDP-IPs fatorados.

6.2.4 Operações unárias em PADDs

Agora mostramos como as duas operações unárias importantes, *restrição* ($F|_{X_i}$) e *marginalização* ($\sum_{x_i \in X_i}$) podem ser implementadas. A *restrição* de uma variável x_i para *verdadeiro* ($F|_{x_i=1}$) ou *falso* ($F|_{x_i=0}$) pode ser implementada substituindo todos os nós de decisão para a variável x_i com o ramo *verdadeiro* (1) ou *falso* (0), respectivamente. Essa operação pode ser usada

Figura 6.5: Um exemplo de multiplicação de PADDs.

Figura 6.6: Um exemplo de aplicação da operação *Restrição* e *Marginalização* em um PADD.

para fazer marginalização como mostraremos a seguir. A operação de *marginalização* (*sum_out*) representada por, $\sum_{x_i \in X_i}$ ou $\sum_{x_i}$, elimina a variável x_i do PADD. É calculada como a soma das operações *restrição* para *verdadeiro* e *falso*, i.e., $(F|_{x_i=1} \oplus F|_{x_i=0})$. Um exemplo é mostrado na Figura 6.6.

Note que as operações unárias *min*, *max* geralmente não podem ser calculadas para PADDs a menos que as folhas possuam valores constantes, isto é, que seja conhecida a atribuição para os parâmetros $\vec{p}$.

6.3 Resumo

Neste capítulo, introduzimos uma das principais contribuições dessa tese: a definição de MDP-IP fatorado e uma especificação fatorada de probabilidades de transição imprecisas, com base nas Redes Credais Dinâmicas (DCN) [Cozman, 2000, Cozman, 2005a]). Além disso, introduzimos uma nova extensão de ADDs, chamada de ADDs Parametrizados, ou simplesmente de PADDs, que permitem a representação de funções de $\{0, 1\}^n$ para o espaço das expressões parametrizadas por $\vec{p}$. Também mostramos as suas propriedades e como realizar as principais operações necessárias para resolver um MDP-IP fatorado com programação dinâmica. No Capítulo 7 é mostrado como DCNs e PADDs permitem fazer cálculos eficientes para resolver MDP-IPs fatorados em programação dinâmica e no Capítulo 8 é apresentado um novo algoritmo aproximado baseado em programação matemática.

Capítulo 7

Soluções Baseadas em Programação Dinâmica

No capítulo anterior, definimos uma representação fatorada para MDP-IPs, baseada em Redes Credais Dinâmicas (DCNs) e definimos uma nova estrutura de dados, PADDs, para representar a função de transição de estados e recompensa de maneira compacta. A seguir, apresentamos um conjunto de algoritmos de ITERAÇÃO DE VALOR originais e eficientes que exploram estas representações, sendo essa, mais uma importante contribuição dessa tese.

7.1 Iteração de Valor para MDP-IP Fatorado

Nesta seção, iremos apresentar a nossa primeira proposta de solução baseada em ITERAÇÃO DE VALOR exata, como uma extensão do algoritmo SPUDD [Hoey et al., 1999] descrito na Seção 4.3, que chamaremos de SPUDD-IP. Primeiro, faremos uma descrição matemática da solução proposta e depois especificaremos formalmente o algoritmo.

7.1.1 O algoritmo SPUDD-IP

Como o algoritmo SPUDD [Hoey et al., 1999] que explora a estrutura dos DBN e ADD na solução de MDPs, o algoritmo proposto SPUDD-IP [Delgado et al., 2009c, Delgado et al., 2009b], explora a estrutura dos DCNs e PADDs na solução de MDP-IPs. Começamos por expressar ITERAÇÃO DE VALOR para MDP-IP na seguinte forma fatorada, usando operações entre diagramas de decisão¹:

$$V_{DD}^t(\vec{x}) = \max_{a \in A} \left\{ R_{DD}(\vec{x}, a) \oplus \gamma \min_{\vec{p}} \left[\sum_{\vec{x}'} \bigotimes_{i=1}^n P_{DD}(x'_i | pa(X'_i), a, \vec{p}) V_{DD}^{t-1}(\vec{x}') \right] \right\}. \quad (7.1)$$

Uma vez que as CPTs de transição contêm parâmetros $\vec{p}$, essas CPTs devem ser representadas por PADDs ($P_{DD}(x'_i | pa(X'_i), a, \vec{p})$). Por outro lado, a recompensa $R(\vec{x}, a)$ pode ser representada como um ADD, uma vez que contém somente constantes (para efeito de operações, note que ADDs são casos especiais de PADDs). Embora possa parecer que a forma de $V_{DD}^t(\vec{x})$ poderia ser um PADD, notamos que os parâmetros $\vec{p}$ são “minimizados” com respeito às restrições sobre $\vec{p}$ durante a operação $\min_{\vec{p}}$ ². Isto é crucial, uma vez que $\max_{a \in A}$ pode ser calculado somente em ADDs

¹Usamos o índice DD para as funções representadas tanto como ADDs como PADDs, uma vez que o primeiro é um caso particular do segundo.

²O símbolo $\square$ representa o conteúdo envolvido pelo quadro na Equação (7.1).

Figura 7.1: a) $V_{DD}^0 = R(x_1, x_2)$ para a topologia anel unidirecional do domínio SYSADMIN com 2 computadores multiplicado por $P_{DD}(x'_1 | x_1, x_2, \text{notreboot})$. b) Resultado de $V_{ADD}^0 \otimes P_{DD}(x'_1 | pa(X'_1), \text{notreboot}, \vec{p})$ e sua marginalização sobre x'_1 .

(lembre que max não é uma operação fechada em PADDs). Assim, o $V_{DD}^t(\vec{x})$ resultante calculado a partir de $\max_{a \in A}$ tem folhas constantes e pode ser expresso com um ADD, um caso especial de PADDs.

Como no SPUDD, para realizar o cálculo da Equação (7.1) de maneira eficiente usamos o algoritmo de Eliminação de Variáveis [Zhang and Poole, 1994] na marginalização sobre todos os estados sucessores, isto é, $\sum_{x'} \cdot$. Por exemplo, se x'_1 não é dependente de qualquer outro x'_i para $i \neq 1$, podemos deslocar na Equação (7.1) a soma sobre x'_1 para dentro para obter (como foi mostrado na Equação (2.11)):

$$\sum_{x'_i (i \neq 1)} \bigotimes_{i=1 (i \neq 1)}^n P_{DD}(x'_i | pa(X'_i), a, \vec{p}) \sum_{x'_1} P_{DD}(x'_1 | pa(X'_1), a, \vec{p}) V_{DD}^{t-1}(\vec{x}').$$

Mostramos isso graficamente no exemplo na Figura 7.1, em que temos a representação ADD para a primeira função valor $V_{DD}^0 = R_{DD}$ (i.e., na iteração 0, neste exemplo, inicializamos a função valor com a função recompensa). V_{DD}^0 é multiplicada por $P_{DD}(x'_1 | pa(X'_1), a, \vec{p})$ (Figura 7.1.a), produzindo um resultado que é marginalizado sobre x'_1 para obter o resultado final (Figura 7.1.b). Em seguida, multiplicamos esse resultado por $P_{DD}(x'_2 | pa(X'_2), a, \vec{p})$, marginalizando sobre x'_2 e assim por diante. Esse processo se repete para todo $x'_i \in \vec{x}$ até completar o cálculo de $\square$ na Equação (7.1).

Representando o $\square$ da Equação (7.1) por $f(\vec{x}, a, \vec{p})$, obtemos:

$$V^t(\vec{x}) = \max_{a \in A} \left\{ R_{DD}(\vec{x}, a) \oplus \gamma \min_{\vec{p}} \boxed{f(\vec{x}, a, \vec{p})} \right\}. \quad (7.2)$$

Note que $\min_{\vec{p}} f(\vec{x}, a, \vec{p})$ leva a uma minimização da expressão multilinear nas folhas do ADD $f_{DD}(\vec{x}, a, \vec{p})$, sujeito ao conjunto de restrições sobre $\vec{p}$. Esse problema de otimização pode ser representado como um simples *programa multilinear*, uma vez que fazemos a suposição de que p_{ij} aparece somente na CPT da DCN para X'_i , o que impede a multiplicação de p_{ij} por si mesmo, evitando assim expoentes maiores do que 1 (conforme definimos nas Redes Credais Dinâmicas do Capítulo 6). Além dessa suposição ser importante para garantir a existência de soluções exatas, o mais importante é que existem implementações eficientes que podem ser utilizadas na prática para resolver programas multilineares. É importante ressaltar que, ao se fazer essa suposição, não há perda de generalidade, uma vez que essa é apenas uma questão de modelagem.

Para demonstrar como $\min_{\vec{p}} f(\vec{x}, a, \vec{p})$ é executado com PADDs, nos referenciamos à Figura 7.2. Cada expressão folha em $f(\vec{x}, a, \vec{p})$ (Figura 7.2.a) corresponde a uma função que a natureza deve minimizar em cada estado. É crucial notar que PADD agrega estados com o mesmo objetivo de minimização, poupando assim chamadas ao otimizador não-linear (observamos essa economia de tempo em nossos experimentos mostrados no Capítulo 10).

A chamada ao otimizador não-linear para cada folha é feita, passando-se a expressão da folha como o objetivo a minimizar que satisfaz as restrições da nossa DCN. Após a minimização, podemos substituir esta folha por uma constante, isto é, o valor objetivo ótimo devolvido pelo otimizador não-linear (Figura 7.2.b). Podemos ver que após a operação $\min_{\vec{p}}$, todos os PADDs são simplificados para o caso especial de ADDs, isto é, com nós folhas iguais a constantes.

Para completar um passo de ITERAÇÃO DE VALOR no MDP-IP fatorado, pegamos o ADD resultante da operação $\min$, multiplicamos pelo escalar γ , adicionamos a recompensa $R(\vec{x}, a)$ e finalmente executamos uma sequência de operações binárias $\max(\cdot, \cdot)$ de ADDs para calcular $\max_a$, obtendo assim o ADD $V^t(\vec{x})$ a partir do ADD $V^{t-1}(\vec{x})$ e completando um passo do algoritmo ITERAÇÃO DE VALOR (Solução (7.1)).

7.1.2 SPUDD-IP: Funções implementadas

Para implementar o algoritmo de IV proposto, SPUDD-IP, foram implementadas duas funções:

- **SOLVE** (Algoritmo 7) constrói uma série de funções valor t-estágios-para-frente V_{DD}^t que são representadas como ADDs. Primeiro, criamos a representação PADD para todas as CPTs da DCN e inicializamos a primeira função valor com 0. O ciclo é repetido até que um número máximo de iterações, $maxIter$, seja alcançado ou até que a condição de terminação do erro de Bellman BE , $BE < tol$, seja satisfeito. Notamos que, definindo a tolerância tol de acordo com a Expressão (3.10), garante-se ϵ -otimalidade para MDP-IPs, uma vez que a mesma condição de término usada para MDPs pode ser generalizada para MDP-IPs no caso descontado ($\gamma < 1$). Em cada iteração o algoritmo **REGRESS** é chamado e V_{DD}^t é atualizado com o $\max$ sobre

Figura 7.2: a) O PADD antes da minimização e um programa multilinear para a primeira folha, a solução para essa folha é o valor constante c_1 . b) O ADD resultante depois da minimização em todas as folhas.

todos os Q_{DD}^t calculados (um para cada ação). Depois disso, $BE = \max_{\vec{x}} |V^t(\vec{x}) - V^{t-1}(\vec{x})|$ é calculado e testado. Observamos que no Algoritmo 7 os parâmetros δ , APP , Obj e $Vmax$ não são usados; eles só serão usados para os algoritmos aproximados da Seção 7.2.

- REGRESS (Algoritmo 8) calcula Q_{DD}^t , i.e, atualiza V_{DD}^{t-1} através da ação a que fornece o valor Q_{DD}^t máximo. Durante a atualização dos valores V_{DD}^{t-1} , convertamos cada variável X_i para X'_i (uma vez que V_{DD}^i é agora parte do próximo estado, isso é feito usando a função CONVERTTOPRIMES) e os $P_{DD}(x'_i | pa(X'_i), a, \vec{p})$ da ação a são multiplicados e marginalizados.³ Depois disso, o otimizador não-linear é chamado para encontrar o $\vec{p}$ que minimiza cada folha no PADD com respeito às restrições lineares da DCN, resultando num ADD. Observamos que se uma folha é uma constante, então a chamada ao otimizador não-linear pode ser evitada; esta observação será importante mais tarde, quando introduzimos a versão aproximada desse algoritmo descrito na Seção 7.2.2. Finalmente, o valor futuro é descontado e o ADD recompensa é adicionado para completar a atualização. Novamente, os parâmetros Obj e $error$ serão usados somente para ITERAÇÃO DE VALOR aproximada da Seção 7.2.

7.2 Iteração de Valor Aproximada para MDP-IP Fatorado

A solução anterior de ITERAÇÃO DE VALOR exata SPUDD-IP para MDP-IPs fatorados frequentemente produz uma melhoria com relação a ITERAÇÃO DE VALOR enumerativa, como demonstraremos nos nossos experimentos. Porém, para problemas em que o número de variáveis de estado é grande, muitas vezes torna-se impossível obter uma solução exata devido a limitações de tempo e espaço.

ITERAÇÃO DE VALOR APROXIMADA (AVI) é uma forma de conciliar tempo e espaço com um limite de erro, através da aproximação da função valor, após cada iteração. Nesta seção, propomos

³Para simplificar a explicação, assumimos que não há arestas síncronas entre as variáveis X'_i , X'_j para $i \neq j$ na DCN (isto é, ligações entre variáveis do nível X'). Se arcos síncronos estão presentes, o algoritmo pode ser modificado para multiplicar todas as CPTs relevantes.

Algoritmo 7: SOLVE(MDP-IP, tol , $maxIter$, δ , APP , Obj)

```

input : MDP-IP (given by  $\langle S, A, R, K, \gamma \rangle$ ),
         $tol$  (tolerance that guarantees  $\epsilon$ -optimality),
         $maxIter$  (maximum number of iterations),
        //variables used for approximate value iteration
         $\delta$  (fraction of the maximum possible value, with  $0 < \delta \leq 1$ ),
         $APP$  ( $APP = true$  to execute APRICODD-IP),
         $Obj$  ( $Obj = true$  to execute OBJECTIVE-IP)
output:  $V_{DD}^t$  (t-state-to-go value function)

1 begin
2   Create PADD:  $P_{DD}(x'_i|pa(X'_i), a, \vec{p})$  for MDP-IP;
3    $V_{ADD}^0 = 0$ ;
4   //  $Vmax$  is the maximum possible value at each iteration
5    $Vmax = \max(R_{DD})$ ;
6    $t = 0$ ;
7   //construct t-stage-to-go value functions  $V_{DD}^t$  until termination condition is met
8   while  $i < maxIter$  do
9      $t = t + 1$ ;
10     $V_{DD}^t = -\infty$ ;
11    //update  $V_{DD}^t$  with the max over all  $Q_{DD}^t$ 
12    foreach  $a \in A$  do
13       $Q_{DD}^t = \text{REGRESS}(V_{DD}^{t-1}, a, \delta \cdot Vmax, Obj)$ ;
14       $V_{DD}^t = \max(V_{DD}^t, Q_{DD}^t)$ ;
15    //compute Bellman Error (BE) and check for termination
16     $Diff_{DD} = V_{DD}^t \ominus V_{DD}^{t-1}$ ;
17     $BE = \max(\max(Diff_{DD}), -\min(Diff_{DD}))$ ;
18    if  $BE < tol$  then
19      break;
20    //approximate value iteration: APRICODD-IP
21    if  $APP$  pruning then
22       $V_{DD}^t = \text{APPROXADD}(V_{DD}^t, \delta \cdot Vmax)$ ;
23       $Vmax = \max(R_{DD}) + \gamma Vmax$ ;
24  return  $V_{DD}^t$  ;
25 end

```

duas extensões de SPUDD-IP para AVI que garantem um limite no erro da aproximação. Cada método usa uma forma diferente de aproximar a função valor, mas ambos os métodos incorrem em um erro máximo de $\delta \cdot Vmax$ por iteração, em que $Vmax$, como calculado na função SOLVE, representa o valor máximo possível em cada passo da ITERAÇÃO DE VALOR e $0 < \delta \leq 1$. Ao fazer o erro de aproximação sensível a $Vmax$ evitamos aproximações do valor excessivamente agressivas nos estágios iniciais de AVI em que os valores são relativamente pequenos, como discutido em [Staubin et al., 2000]. Mesmo com essa aproximação da função valor a cada iteração, satisfazendo a condição de finalização $BE < tol$ para algum tol , ainda é possível manter garantias rigorosas no erro de aproximação global dado pela Expressão (3.10), como discutido anteriormente para o

Algoritmo 8: REGRESS($V_{DD}, a, error, Obj$)

input : V_{DD} (value function),
 a (action),
 $error$ (maximum error),
 Obj ($Obj = true$ to execute OBJECTIVE-IP)

output: Q_{DD} (the value function obtained if executing a and acting so as obtain V_{DD} thereafter)

```

1 begin
2    $Q_{DD} = \text{CONVERTTOPRIMES}(V_{DD});$  //convert variables  $X_i$  to  $X'_i$ 
3   //CPTs are multiplied in and summed out
4   for all  $X'_i$  in  $Q_{DD}$  do
5      $Q_{DD} = Q_{DD} \otimes P_{DD}(x'_i | pa(X'_i), a, \vec{p});$ 
6      $Q_{DD} = \sum_{x'_i \in X'_i} Q_{DD};$ 
7   //approximate value iteration: OBJECTIVE-IP
8   if  $Obj$  pruning then
9      $Q_{DD} = \text{APPROXPADDLEAVES}(Q_{DD}, error);$ 
10  //call the non-linear solver for each PADD leaf
11   $Q_{DD} = \text{CALLSOLVERFOREACHLEAFIN}(Q_{DD});$ 
12   $Q_{DD} = R_{DD} \oplus (\gamma \otimes Q_{DD});$ 
13  return  $Q_{DD};$ 
14 end

```

Algoritmo 9: APPROXADD($value_{DD}^i, error$)

input : $value_{DD}^i$ (an ADD),
 $error$ (maximum error)

output: a new ADD

```

1 begin
2   //collect all leaves of the ADD
3    $leaves_{old} = \text{COLLECTLEAVESADD}(value_{DD}^i);$ 
4   //group the leaves that can be merged within maximum error
5    $\{leaves_{old} \rightarrow leaves_{new}\} = \text{MERGELEAVES}(leaves_{old}, error);$ 
6   //return a simplified ADD
7   return  $\text{CREATENEWDD}(value_{DD}^i, \{leaves_{old} \rightarrow leaves_{new}\});$ 
8 end

```

algoritmo SPUDD-IP.

7.2.1 Algoritmo APRICODD-IP

O algoritmo APRICODD [St-aubin et al., 2000] fornece uma forma eficiente de aproximar a representação ADD da função valor para um MDP fatorado, reduzindo seu tamanho e, portanto, reduzindo o tempo de processamento por iteração. Esta abordagem pode ser generalizada para MDP-IPs, uma vez que V_{DD}^t é também um ADD. Para executar APRICODD-IP para MDP-IPs, simplesmente chamamos SOLVE (Algoritmo 7) com $APP = true$ e definimos δ entre $0 < \delta \leq 1$

Figura 7.3: a) Função valor V_{DD}^t representada como um ADD. b) Resultado da aplicação de APPROXADD em V_{DD}^t com aproximação $error = 1$. Note que as folhas cuja diferença esteja dentro dos limites de $error = 1$ foram fundidas calculando a média foi calculada e o ADD resultante foi simplificado.

como uma fração do valor máximo possível $Vmax$ para realizar a aproximação usando a função APPROXADD. As principais funções do Algoritmo APRICODD-IP são:

- APPROXADD (Algoritmo 9) coleta todas as folhas do ADD e determina quais podem ser fundidas para formar novos valores sem aproximar mais do que $error = \delta \cdot Vmax$. Os valores antigos são então substituídos com os novos valores, criando um ADD menor, que representa a função valor aproximada. Um exemplo ilustrativo é mostrado na Figura 7.3.
- COLLECTLEAVESADD compila as folhas do ADD e as coloca num conjunto ($leaves_{old}$).
- MERGELEAVES agrupa as folhas que podem ser fundidas dentro do erro. Isso é feito selecionando as folhas da ramificação F_h (F_l) cuja diferença seja menor do que $error = \delta \cdot Vmax$. Em seguida, calcula a média para cada grupo criando um novo conjunto de folhas ($leaves_{new}$).
- CREATENEWDD cria um ADD simplificado substituindo as folhas antigas pelas novas.

7.2.2 Algoritmo OBJECTIVE-IP

APRICODD é uma extensão eficaz de SPUDD para MDPs fatorados, mas não podemos dizer o mesmo para MDP-IPs. Enquanto para MDPs é possível reduzir o tamanho do ADD que representa a função valor, que nesse caso determina a complexidade no tempo do algoritmo, na resolução de MDP-IPs fatorados, o tempo é ditado menos pelo tamanho do ADD da função valor e mais pelo número de chamadas ao otimizador não-linear para calcular $\min_{\vec{p}}$. SPUDD-IP pode diminuir essa fonte de complexidade no tempo agregando estados com o mesmo objetivo para $\min_{\vec{p}}$ (isto é, mesma função objetivo do programa multilinear de cada folha do PADD). Assim, nossa proposta de um novo algoritmo de aproximação, chamado de OBJECTIVE-IP, é implementar uma versão AVI de SPUDD-IP *aproximando os objetivos*. Fazemos isso, deixando de realizar a aproximação da linha 22 do Algoritmo 7 e permitindo o passo de aproximação dentro do algoritmo REGRESS (linha 9)

Algoritmo 10: APPROXPADDLEAVES($DD, error$)

```

input :  $DD$  (parameterized ADD),
         $error$  (maximum error)
output:  $DD$  (simplified parameterized ADD)

1 begin
2   //approximate each leaf independently
3   foreach  $leaf : d_0 + \sum_{i=1}^{\#terms} d_i \prod_j p_{ij} \in DD$  do
4      $i = 1, curError = 0;$ 
5     //for all terms of the leaf, prune term if possible
6     while  $curError < error \wedge i \leq \#terms$  do
7       //compute the average of maximum and minimum values for term  $i$ 
8        $newValue = \frac{d_i}{2} \left( \prod_j p_{ij}^U + \prod_j p_{ij}^L \right);$ 
9       //compute the error of using maximum and minimum values for term  $i$ 
10       $termError_i = \left| \frac{d_i}{2} \left( \prod_j p_{ij}^U - \prod_j p_{ij}^L \right) \right|;$ 
11      //if within error, prune term  $i$  from leaf
12      if  $curError + termError_i < error$  then
13        remove term  $d_i \prod_j p_{ij}$  from  $leaf$ ;
14         $d_0 = d_0 + newValue;$ 
15         $curError = curError + termError_i;$ 
16         $i = i + 1;$ 
17
18   return  $DD$ ;
19 end

```

em seguida o otimizador não-linear é chamado. Note que cada folha do PADD em REGRESS é um objetivo multilinear, que é simplificada chamando APPROXPADDLEAVES:

APPROXPADDLEAVES (Algoritmo 10) é chamado para um PADD (por REGRESS se $Obj = true$) antes de realizar a otimização não-linear em suas folhas. O laço principal do algoritmo tenta aproximar cada folha do PADD. A aproximação é feita no laço interior podando os termos multilineares $d_i \prod_j p_{ij}$ da expressão da folha e substituindo-os pela média de seus valores máximo e mínimo. Isso requer conhecer os limites absolutos máximo p_{ij}^U e mínimo p_{ij}^L para cada p_{ij} , que podem ser facilmente pré-calculados, uma única vez para todo o MDP-IP, chamando-se o otimizador não-linear para calcular $p_{ij}^U = \max p_{ij}$ e $p_{ij}^L = \min p_{ij}$. Em alguns casos, a complexidade da expressão da folha pode ser reduzida a uma expressão mais simples, em outros, pode realmente ser reduzida a uma constante. Note que as folhas são aproximadas de forma independente, isto pode ser feito uma vez que cada folha corresponde a um estado diferente (ou conjunto de estados). Além disso, podemos garantir que nenhum objetivo podado nas folhas do PADD possui um erro de aproximação maior do que $error = \delta \cdot Vmax$, após a otimização não-linear ser realizada.

Teorema 7.1 (APPROXPADDLEAVES Limite de Erro). *Dado um MDP-IP, suas constantes pré-calculadas p_{ij}^L e p_{ij}^U , para todo p_{ij} , e o erro de aproximação máximo (“error”), toda vez que AP-*

PROXPADDLEAVES (Algoritmo 10) reduz uma folha $d_0 + \sum_{i=1}^{\#terms} d_i \Pi_j p_{ij}$ para uma expressão mais simples, o erro de aproximação no objetivo de minimização ($\min_{\vec{p}}$) dessa folha é limitado por “error”.

Prova: Começamos mostrando que o erro de aproximação induzido pela remoção de um termo individual i do objetivo é delimitada por $termError_i$. Para fazer isto, primeiro encontramos os limites superior e inferior do termo i ($d_i \Pi_j p_{ij}$) baseados nos valores válidos de $\vec{p}$. Conhecemos que o máximo (mínimo) valor possível para cada p_{ij} é p_{ij}^U (p_{ij}^L), chamando-se um otimizador. Assim, para quaisquer valores válidos possíveis de $\vec{p}$, o termo i deve ser delimitado ao intervalo $[L_i, U_i]$ com L_i e U_i definidos a seguir:

$$L_i = \begin{cases} \text{se } d_i > 0 & \text{então } d_i \Pi_j p_{ij}^L \\ \text{se } d_i < 0 & \text{então } d_i \Pi_j p_{ij}^U \end{cases}, \quad U_i = \begin{cases} \text{se } d_i > 0 & \text{então } d_i \Pi_j p_{ij}^U \\ \text{se } d_i < 0 & \text{então } d_i \Pi_j p_{ij}^L \end{cases}$$

Seja g o valor do termo i e $\hat{g} = \frac{L_i + U_i}{2}$, então $\max_g |g - \hat{g}|$ ocorre quando $g = L_i$ ou $g = U_i$. Assim, o máximo $termError_i = \max(|L_i - \hat{g}|, |U_i - \hat{g}|) = \max\left(\left|\frac{L_i - U_i}{2}\right|, \left|\frac{U_i - L_i}{2}\right|\right) = \left|\frac{L_i - U_i}{2}\right|$ (como calculado na linha 10 do Algoritmo 10).

Agora, seja $OBJ1 = d_0 + \sum_{i=1}^{\#terms} d_i \Pi_j p_{ij}$ a expressão do objetivo original (i.e. não aproximada) a ser minimizada e v_1 o valor do objetivo ótimo usando $\vec{p} = \vec{p}_1$. Seja $OBJ2 = d_0 + \hat{g} + \sum_{i=2}^{\#terms} d_i \Pi_j p_{ij}$ a expressão do objetivo aproximado a ser minimizada depois de substituir o termo 1 com $\frac{L_1 + U_1}{2}$ e v_2 o objetivo ótimo usando $\vec{p} = \vec{p}_2$.

Desejamos provar que $-\left|\frac{L_1 - U_1}{2}\right| < v_1 - v_2 < \left|\frac{L_1 - U_1}{2}\right|$. Primeiro provamos a segunda parte dessa desigualdade. Usando $\vec{p}_2$ em $OBJ1$ e a expressão do objetivo aproximado obtemos $v'_1 = d_0 + eval(d_1 \Pi_j p_{1j}, \vec{p}_2) + v_2 - d_0 - \hat{g}$ (sendo $eval$ uma função para avaliar o termo com os valores atribuídos). Uma vez que v_1 é ótima $v_1 \leq v'_1$ então:

$$v_1 - v_2 \leq eval(d_1 \Pi_j p_{1j}, \vec{p}_2) - \hat{g}. \quad (7.3)$$

Adicionalmente para qualquer valor válido possível de $\vec{p}$ e $\vec{p}_2$, $|eval(d_1 \Pi_j p_{1j}, \vec{p}_2) - \hat{g}| < \left|\frac{L_1 - U_1}{2}\right|$, i.e., $-\left|\frac{L_1 - U_1}{2}\right| < eval(d_1 \Pi_j p_{1j}, \vec{p}_2) - \hat{g} < \left|\frac{L_1 - U_1}{2}\right|$. A partir desta equação e da Inequação (7.3) obtemos $v_1 - v_2 < \left|\frac{L_1 - U_1}{2}\right|$. A prova da primeira desigualdade segue o mesmo raciocínio, mas desta vez substituindo $\vec{p}_1$ em $OBJ2$ e usando a expressão do objetivo não-aproximada. Assim, obtemos $v'_2 = d_0 + \hat{g} + v_1 - d_0 - eval(d_1 \Pi_j p_{1j}, \vec{p}_1)$. Uma vez que v_2 é ótima $v_2 \leq v'_2$ então:

$$v_2 - v_1 \leq \hat{g} - eval(d_1 \Pi_j p_{1j}, \vec{p}_1). \quad (7.4)$$

Além disso, para qualquer valor válido possível de $\vec{p}$ e $\vec{p}_1$ vale a desigualdade, $-\left|\frac{L_1 - U_1}{2}\right| < eval(d_1 \Pi_j p_{1j}, \vec{p}_1) - \hat{g} < \left|\frac{L_1 - U_1}{2}\right|$. A partir desta equação e da Inequação (7.4) obtemos $v_1 - v_2 > -\left|\frac{L_1 - U_1}{2}\right|$.

Isso limita o erro da aproximação do objetivo quando aproximamos um termo e, por indução simples, podemos limitar aditivamente o erro acumulado para múltiplas aproximações como as calculadas usando $curError$ no Algoritmo 10. ■

7.3 Resumo

Neste capítulo foi proposto um conjunto de algoritmos originais baseados em programação dinâmica que exploram a representação compacta de MDP-IPs fatorados através de DCNs e PADDs, a saber: SPUDD-IP, APRICODD-IP e OBJECTIVE-IP.

A escolha entre as soluções exatas e aproximadas para MDP-IPs é determinada pelo tamanho do espaço de estados. Se o número de variáveis de estado no problema não for muito grande, nossa solução exata de ITERAÇÃO DE VALOR SPUDD-IP pode ser usada. No entanto, se o espaço de estados cresce muito, podemos utilizar as soluções aproximadas propostas, APRICODD-IP ou OBJECTIVE-IP. Estas soluções utilizam diferentes formas de aproximar a função valor. Enquanto APRICODD-IP poda o ADD que representa o valor por iteração, OBJECTIVE-IP ataca a fonte da complexidade do tempo de execução em MDP-IPs através da agregação de estados com o mesmo objetivo e simplificando os polinômios das folhas do PADD original. No próximo capítulo é proposta outra solução aproximada, baseada em programação matemática.

Capítulo 8

Solução Baseada em Programação Matemática

No Capítulo 6 apresentamos um modelo para MDP-IP fatorado, em que os estados são representados por um conjunto de variáveis de estado e usamos Redes Credais Dinâmicas para representar a função de transição fatorada. Nesse capítulo, apresentamos duas formulações possíveis baseadas em programação matemática para um MDP-IP fatorado. Além disso, propomos um novo algoritmo aproximado, baseado no algoritmo ALP apresentado na Seção 3.4.

8.1 Formulação em Dois Níveis e Formulação Multilinear

Para derivar políticas maximin para MDP-IPs fatorados, começamos por especializar o Problema (5.3) (Seção 5.1.2). A *função valor fatorada* de um MDP-IP fatorado é dada pela Equação (4.6), restringindo o escopo de cada função base para um subconjunto de variáveis de estado $\mathbb{N}_i \subset \mathbf{X} = \{X_1, \dots, X_n\}$. Podemos inserir a função valor fatorada (4.6), a função recompensa fatorada (6.4), as probabilidades de transição (6.3) no Problema (5.3) para obter:

$$\begin{aligned}
 \min_w \quad & \sum_{\vec{x}} \sum_{i=1}^k w_i h_i(\vec{x}) \\
 s.t. \quad & \sum_{i=1}^k w_i h_i(\vec{x}) \geq R(\vec{x}, a) + \gamma \sum_{\vec{x}' \in \mathcal{S}} P(\vec{x}' | \vec{x}, \vec{p}, a) \sum_{i=1}^k w_i h_i(\vec{x}'), \forall \vec{x} \in \mathcal{S}, a \in \mathcal{A}. \\
 & p \in \operatorname{argmin} \sum_{\vec{x}' \in \mathcal{S}} P(\vec{x}' | \vec{x}, \vec{p}, a) \sum_{i=1}^k w_i h_i(\vec{x}') \\
 & s.t. : P(\vec{x}' | \vec{x}, \vec{p}, a) = \prod_i P(x'_i | pa(X'_i), \vec{p}, a), \\
 & P(x'_i | pa(X'_i), \vec{p}, a) \in K(x'_i | pa(X'_i), a).
 \end{aligned} \tag{8.1}$$

Algumas características importantes do Problema (8.1) são: existem $|\mathcal{S}| * |\mathcal{A}|$ restrições no primeiro nível; as restrições do primeiro nível são não-lineares; o objetivo no segundo nível é não-linear quando as funções base são baseadas em mais de uma variável; o primeiro e o segundo nível compartilham as mesmas variáveis (i.e., as variáveis que correspondem às probabilidades). O Problema (8.1) não é um problema de dois níveis simples e devido a essas características, a maioria dos métodos conhecidos para resolver problemas em dois níveis não podem ser utilizados,

tais como: o método *K-th Best* [Bialas and Karwan, 1978], o método *Branch-and-bound* [Bard, 1988], o método *Trust-Region* [Colson et al., 2005], o método *Inexact Restoration* [Andreani et al., 2009] e o método *Steepest Descent Direction* [Savard and Gauvin, 1994].

Uma outra formulação de um MDP-IP fatorado pode ser definida, substituindo-se a função valor fatorada no Problema (5.4) e obtendo-se o problema de programação multilinear fatorado:

$$\begin{aligned}
 \min_{w, \vec{p}} \quad & \sum_{\vec{x}} \sum_i w_i h_i(\vec{x}) \\
 s.t. \quad & \sum_{i=1}^k w_i h_i(\vec{x}) \geq \sum_{j=1}^{\psi} R_j(\vec{x}, a) + \gamma \sum_{\vec{x}' \in \mathcal{S}} P(\vec{x}' | \vec{x}, \vec{p}, a) \sum_{i=1}^k w_i h_i(\vec{x}'), \forall \vec{x} \in \mathcal{S}, a \in \mathcal{A}, \\
 & P(\vec{x}' | \vec{x}, \vec{p}, a) = \prod_i P(x'_i | pa(X'_i), \vec{p}, a), \\
 & P(x'_i | pa(X'_i), \vec{p}, a) \in K(x'_i | pa(X'_i), a).
 \end{aligned} \tag{8.2}$$

8.2 Como Resolver um MDP-IP Fatorado Baseado na Formulação Multilinear?

No problema (8.2), mesmo que pudéssemos calcular os coeficientes da função objetivo e as restrições numa forma eficiente, ainda trabalhamos com o conjunto completo de restrições ($|\mathcal{S}| * |\mathcal{A}| + m_2$), em que m_2 é o número de restrições relacionadas com as probabilidades p_{ij} , e o uso direto de otimizadores não-lineares gerais [Murtagh and Saunders, 1998, Andreani et al., 2007], otimizadores geométricos [Boyd et al., 2009] ou otimizadores multilineares [Sherali and Tuncbilek, 1992] para MDP-IPs fatorados, só podem resolver problemas com um espaço de estados pequeno.

Outra opção é usar algoritmos de otimização local para resolver o problema multilinear [Lukatskii and Shapot, 2000]. Esse algoritmo divide as variáveis em grupos e, a cada iteração, fixa os valores de cada grupo a fim de obter um problema linear. Um MDP-IP fatorado formulado como um problema de programação multilinear pode ser resolvido por esse algoritmo de otimização local se e somente se as funções base tem escopo restrito a uma variável, uma vez que podemos dividir as variáveis em dois grupos, por exemplo, podemos definir um grupo com as variáveis w_i para $i = 1$ até k e outro grupo com w_0 e as variáveis relacionadas com as probabilidades. Caso contrário, quando as funções base tem mais de uma variável, não é possível dividir as variáveis, a fim de obter um problema linear em cada iteração, assim o algoritmo [Lukatskii and Shapot, 2000] não pode ser aplicado.

Outra opção é tentar reduzir o número de restrições no problema. Na próxima seção, mostraremos uma outra contribuição principal deste trabalho: um algoritmo para a geração de políticas maximin em MDP-IPs fatorados que resolve o Problema (8.2) reduzindo o número de restrições a fim de resolver problemas com tamanhos grandes de espaço de estados.

Teorema 8.1. *Se o conjunto de funções base escolhidas admite uma solução exata, então podemos encontrá-la resolvendo o Problema (8.2).*

Prova: Um conjunto de funções base admite a solução exata se $V^*(\vec{x}) = \sum_{j=1}^k w_j h_j(\vec{x})$. Por-

tanto, substituindo V^* no Problema (8.2), este é reduzido para o Problema (5.4) e pelo Lema 5.1, é garantido que resolvendo o Problema (8.2) encontraremos a solução exata, $V^*(s)$. ■

8.3 Uma Solução Eficiente para MDP-IPs Fatorados: FactoredMPA

Com base nas idéias de Guestrin [Guestrin, 2003] (Seção 4.4), queremos resolver um MDP-IP fatorado formulado como um programa multilinear aproximado (*Multilinear Programming Approximated* - MPA) (Problema (8.2)). As mesmas técnicas gerais e eficientes que foram usadas para estender o algoritmo de Guestrin [Patrascu, 2004, de Farias and Roy, 2004, Dolgov and Durfee, 2006] não podem ser aplicadas diretamente ao problema multilinear. No entanto, o algoritmo FACTOREDMPA pode ser estendido para o algoritmo FACTOREDMPA [Delgado et al., 2009a, Delgado et al., 2010] para resolver MDP-IPs fatorados como mostraremos a seguir.

Para resolver um MDPIP fatorado temos que simplificar o conjunto de restrições do Problema (8.2) aplicando as mesmas idéias usadas por Guestrin no algoritmo FACTOREDMPA: o algoritmo cria um novo conjunto equivalente de restrições menor para cada ação do Problema de Programação Linear (4.7) antes de chamar o otimizador linear com o novo problema de minimização. Isso é feito em dois passos: (i) *passo de simplificação* e (ii) *passo de contração*.

8.3.1 Passo de simplificação: Explorando a estrutura fatorada de um MDPIP

Seja o MDPIP definido por $\langle S, A, R, K, \gamma \rangle$, em que S é o conjunto de estados, A é o conjunto de ações, R é a função recompensa associada a cada estado e ação e é definida por funções de recompensa local R_j , $K(s'|s, a)$ é o conjunto de transição credal definido por DCNs e restrições de probabilidade $\mathfrak{C}$ e $\gamma \in (0, 1)$ é o fator de desconto. O cálculo das restrições no Problema (8.2) deve ser beneficiado pelo fato que nosso MDPIP é fatorado e que o conjunto de funções base H tem escopo restrito. Do Problema (8.2); temos o seguinte conjunto de restrições:

$$\sum_i w_i h_i(\vec{x}) \geq \sum_{j=1}^{\psi} R_j(\vec{x}, a) + \gamma \sum_{\vec{x}' \in S} P(\vec{x}' | \vec{x}, \vec{p}, a) \sum_i w_i h_i(\vec{x}'), \forall \vec{x} \in S, a \in A. \quad (8.3)$$

Podemos reordenar os termos, obtendo:

$$\sum_i w_i h_i(\vec{x}) \geq \sum_{j=1}^{\psi} R_j(\vec{x}, a) + \gamma \sum_i w_i g_i^a(\vec{x}, \vec{p}), \quad (8.4)$$

em que

$$g_i^a(\vec{x}, \vec{p}) = \sum_{\vec{x}' \in S} P(\vec{x}' | \vec{x}, \vec{p}, a) h_i(\vec{x}'); \quad (8.5)$$

usamos $\vec{p}$ para denotar o vetor contendo todos os valores de probabilidade que são livres de variar dentro dos conjuntos credais dados (i.e., que satisfazem as restrições de probabilidade $\mathfrak{C}$ no DCN). Isto é, $\vec{p}$ contém todos os valores de probabilidade que definem as distribuições que estamos procurando.

Koller e Parr [Koller and Parr, 1999] mostraram que se h_i tem escopo restrito a um subconjunto

de variáveis de estado $\aleph_i \subset \mathbf{X} = \{X_1, \dots, X_n\}$, então g_i^a tem escopo restrito aos pais de $\aleph_i$ no DBN da ação a (esses pais são denotados por Γ).

Intuitivamente, em $\sum_{\vec{x}' \in \mathcal{S}} P(\vec{x}' | \vec{x}, \vec{p}, a) h_i(\vec{x}')$ podemos deslocar a soma sobre as variáveis $x'_j \notin \Gamma$ para dentro e obter:

$$\sum_{x'_i \in \Gamma} P(x'_i | \vec{x}, \vec{p}, a) h_i(\vec{x}') \sum_{x'_j \notin \Gamma} P(x'_j | \vec{x}, \vec{p}, a).$$

Como a soma interna é igual a 1, temos:

$$\sum_{x'_i \in \Gamma} P(x'_i | \vec{x}, \vec{p}, a) h_i(\vec{x}').$$

Para MDPIPs, $g_i^a(\vec{x}, \vec{p})$ é uma expressão polinomial e tem escopo restrito aos pais de $\aleph_i$ no DCN. Isto é, é descrita em termos de valores de probabilidade e tem a forma canônica $d_0 + \sum_i d_i \prod_j p_{ij}$. Assim, a fim de evitar cálculos repetidos, precisamos apenas calcular g_i^a para cada atribuição z de Γ . Para melhorar ainda mais os cálculos, o conjunto de restrições pode ser reescrito como:

$$0 \geq \sum_{j=1}^{\psi} R_j(\vec{x}, a) + \sum_i w_i c_i^a(\vec{x}, \vec{p}), \quad (8.6)$$

em que:

$$c_i^a(\vec{x}, \vec{p}) = \gamma g_i^a(\vec{x}, \vec{p}) - h_i(\vec{x}). \quad (8.7)$$

Esse último termo pode ser pré-calculado resultando também numa forma polinomial e tem escopo restrito a $\Theta = \aleph_i \cup \Gamma$. Finalmente obtemos:

$$0 \geq \sum_{j=1}^{\psi} R_j(\vec{x}, a) + \sum_i w_i c_i^a(\vec{x}, \vec{p}), \forall \vec{x} \in \mathcal{S}, a \in \mathcal{A}. \quad (8.8)$$

O algoritmo 11 chamado de COMPUTEC_i^a tem como entrada um MDPIP dado por $\langle S, A, R, K, \gamma \rangle$ e um conjunto de funções base H , e devolve o conjunto de funções $\mathcal{C}^a$. COMPUTEC_i^a calcula g_i^a e c_i^a para cada função base h_i como na Expressão (8.5) e (8.7) respectivamente, note que o termo g_i^a tem escopo restrito aos pais de $\aleph_i$ no DCN (denotado por Γ) e c_i^a tem escopo restrito a $\Theta = \aleph_i \cup \Gamma$.

Ainda que o pré-cálculo da expressão c_i^a ajude a simplificar as restrições, note que estamos trabalhando com o conjunto completo de restrições (isto é, $|S| * |A| + m_2$). Como a aplicação de otimizadores não-lineares gerais ao Problema (8.2) pode só resolver problemas com um espaço de estados pequeno, temos que encontrar uma maneira de reduzir o número de restrições (Section 8.3.2).

8.3.2 Passo de contração: Gerando um conjunto compacto de restrições

Guestrin [Guestrin, 2003] gera um conjunto compacto de restrições *lineares* para resolver de forma eficiente MDPs fatorados; podemos também gerar um conjunto compacto de restrições *multilineares* para resolver MDPIPs fatorados (Problem (8.2)). A idéia básica é primeiro substituir o

Algoritmo 11: COMPUTE $c_i^a(MDPIP, H, a)$

```

input : MDPIP (given by  $\langle S, A, R, K, \gamma \rangle$ ),
         $H$  (set of basis functions  $h_i$ ),
         $a$  (action)
output:  $\mathcal{C}^a$  (set of functions  $c_i^a$ , with  $0 \leq i \leq |H|$ )

1 begin
2   //  $k$  is the number of basis functions
3    $k = |H|$ ;
4   for  $i = 0$  to  $k$  do
5     //compute  $g_i^a$ 
6      $\aleph_i = \text{scope of } h_i$ ;
7      $\Gamma = \text{parents of } \aleph_i \text{ in the DCN}$ ;
8     foreach assignment  $z$  of  $\Gamma$  do
9       | calculate  $g_i^a(z, \vec{p})$ ; //Equation (8.5)
10    //compute  $c_i^a$  using  $g_i^a$ 
11     $\Theta = \aleph_i \cup \Gamma$ ;
12    foreach assignment  $\theta$  of  $\Theta$  do
13      |  $c_i^a(\theta, \vec{p}) = \gamma g_i^a(\theta, \vec{p}) - h_i(\theta)$ ;
14
15  return  $\mathcal{C}^a$  ;
16 end

```

conjunto de restrições da Expressão (8.8) por um conjunto equivalente de restrições não-lineares (maximizando sobre $\vec{x}$), dado por:

$$0 \geq \max_{\vec{x}} \left\{ \sum_{j=1}^{\psi} R_j(\vec{x}, a) + \sum_i w_i c_i^a(\vec{x}, \vec{p}) \right\}, \forall a \in \mathcal{A}. \quad (8.9)$$

Assim, para cada ação a , temos que satisfazer:

$$0 \geq \max_{\vec{x}} \left\{ \sum_{j=1}^{\psi} R_j^a(\vec{x}) + \sum_i w_i c_i^a(\vec{x}, \vec{p}) \right\}. \quad (8.10)$$

em que $R_j^a(\vec{x})$ representa $R_j(\vec{x}, a)$. Agora mostraremos como transformar as restrições dadas pela Inequação (8.10) num conjunto de restrições mais simples. Uma vez que na Expressão (8.10) temos que resolver a maximização sobre o conjunto completo de estados, seu cálculo é muito caro. Ao invés de adicionar todos os termos e fazer a maximização sobre todos os estados em S , podemos maximizar sobre as variáveis de estado uma a uma. Para isso, modificamos versão do algoritmo de eliminação de variáveis proposta por Guestrin [Guestrin, 2003]. Por exemplo, suponha que desejamos fazer a maximização sobre a variável $X_1 \in \vec{x}$. Para eliminar essa variável, fazemos o seguinte: se R_1^a é a única função de recompensa local para a ação a que depende de X_1 ; c_1^a é uma função que depende, por exemplo, de X_1 e X_4 e não existe outra função c_i^a que dependa de X_1 na Inequação (8.10); então podemos deslocar a maximização sobre X_1 para dentro da somatória

obtendo:

$$0 \geq \max_{X_2 \dots X_n} \left\{ \sum_{j=2}^{\psi} R_j^a(\vec{x}) + \sum_{i=2} w_i c_i^a(\vec{x}, \vec{p}) + \boxed{\max_{X_1} \{R_1^a(X_1) + w_1 c_1^a(X_1, X_4, \vec{p})\}} \right\}. \quad (8.11)$$

Chamamos $R_1^a(X_1)$ e $w_1 c_1^a(X_1, X_4, \vec{p})$ na Expressão (8.11), em nosso exemplo hipotético, de *funções relevantes* para a variável X_1 (uma vez que no exemplo R_1^a é a única função de recompensa local que depende de X_1 e o mesmo para a função c_1^a). Em geral, a função relevante é uma função cujo escopo contém a variável X_l que desejamos eliminar. No exemplo acima, as funções relevantes, renomeadas como $u_{Z_1}^{f_1}, \dots, u_{Z_L}^{f_L}$, são $u_{X_1}^{f_1} = R_1^a(X_1)$ e $u_{X_1, X_4}^{f_2} = w_1 c_1^a(X_1, X_4, \vec{p})$ (chamamos essas expressões de *restrições de igualdade*). Note que Z_i é o escopo de cada função relevante.

Assim, para cada variável X_l que desejamos eliminar (sobre algum critério de ordenação O), FACTOREDMP seleciona L funções relevantes. Então podemos substituir a maximização sobre essas funções relevantes por:

$$u_Z^{e_r} = \max_{X_l} \sum_{j=1}^L u_{Z_j}^{f_j}, \quad (8.12)$$

em que Z é a união de todas as variáveis nas funções $u_{Z_1}^{f_1}, \dots, u_{Z_L}^{f_L}$ menos X_l , uma vez que depois de fazer o max, a nova função é independente de X_l .

Voltando para nosso exemplo (Expressão (8.11)), o termo $u_Z^{e_r}$ dado pela maximização no quadro em que $Z = \{X_1\} \cup \{X_1, X_4\} \setminus \{X_1\}$, é renomeado com as funções relevantes como:

$$u_{X_4}^{e_r} = \max_{X_1} \left\{ u_{X_1}^{f_1} + u_{X_1, X_4}^{f_2} \right\}, \quad (8.13)$$

resultando na seguinte expressão:

$$0 \geq \max_{X_2 \dots X_n} \left\{ \sum_{j=2}^{\psi} R_j^a(\vec{x}) + \sum_{i=2} w_i c_i^a(\vec{x}, \vec{p}) + u_{X_4}^{e_r} \right\}.$$

Note que, em nosso exemplo, eliminar X_1 significa simplificar o cálculo da Expressão (8.11) uma vez que existem poucas funções relevantes para X_1 . Mas para forçar a definição de $u_{x_4}^{e_r}$ como na Expressão (8.13) devemos introduzir quatro novas restrições de desigualdade, uma para cada combinação das configurações de X_4 e X_1 , i.e.:

$$\begin{aligned} u_{x_4}^{e_r} &\geq u_{x_1}^{f_1} + u_{x_1, x_4}^{f_2}. \\ u_{\overline{x_4}}^{e_r} &\geq u_{x_1}^{f_1} + u_{x_1, \overline{x_4}}^{f_2}. \\ u_{x_4}^{e_r} &\geq u_{\overline{x_1}}^{f_1} + u_{\overline{x_1}, x_4}^{f_2}. \\ u_{\overline{x_4}}^{e_r} &\geq u_{\overline{x_1}}^{f_1} + u_{\overline{x_1}, \overline{x_4}}^{f_2}. \end{aligned}$$

No caso geral, para forçar a definição de $u_Z^{e_r}$ como o máximo sobre X_l (Expressão (8.12)), o algoritmo

FACTOREDMP introduz uma nova restrição para cada atribuição z de Z ; i.e.:

$$u_z^{er} \geq \sum_{j=1}^L u_{z,x_l}^{f_j} \quad \forall x_l. \quad (8.14)$$

Esse procedimento é repetido até que todas as variáveis tenham sido eliminadas. No final, todas as funções restantes u^{e_i} terão escopo vazio e a restrição de desigualdade a seguir deve ser adicionada:

$$0 \geq \sum_i u^{e_i}. \quad (8.15)$$

isto é, no final do processo de eliminação de variáveis, transformamos o conjunto original de restrições de desigualdade (Expression (8.10)) em um novo conjunto de restrições (Expression (8.14) e (8.15)) mais as restrições de igualdade para as funções renomeadas, o que corresponde a um conjunto menor de restrições mais simples (i.e., sem a maximização).

FACTOREDMP (Algoritmo 12) implementa o passo de contração criando um conjunto equivalente e menor de restrições $\mathbb{J}$ para **uma ação** $a \in \mathcal{A}$, como foi descrito. Ele tem como entrada um conjunto de funções $\mathcal{C}^a$, o conjunto de recompensa local R^a , um critério de ordenação O e o número de variáveis de estado n . O algoritmo começa com $\mathbb{F} = \{\}$ e $\mathbb{J} = \{\}$, em que $\mathbb{F}$ é o conjunto de funções novas que serão criadas durante o processo (como as funções relevantes renomeadas e as novas funções criadas na Expressão (8.14)) e $\mathbb{J} = \{\}$ é o conjunto de novas restrições, inicialmente vazia, que será de dois tipos: restrições de igualdade e desigualdade. FACTOREDMP chama as seguintes funções para gerar o conjunto de *restrições de igualdade*:

- GENERATEEQUALITYCONSFORREWARD (Algoritmo 13) gera um conjunto de restrições de igualdade, $u_z^{er} = R_j^a(Z)$, para cada atribuição z de Z (escopo) de cada recompensa local R_j^a e
- GENERATEEQUALITYCONSFORC_i (Algoritmo 14) gera um conjunto de restrições de igualdade multilinear, $u_z^{er} = c_i^a(Z, \vec{p}) * w_i$, para cada atribuição z de Z (escopo) de cada função c_i^a .

Finalmente, o algoritmo gera um conjunto de restrições de desigualdade chamando a função GENERATEINEQUALITYCONS (Algoritmo 15), a qual elimina as variáveis de estado uma a uma. Essas variáveis devem estar ordenadas por algum critério O (por exemplo, eliminando primeiro as variáveis que produzem funções menores). Para cada variável eliminada X_l , as funções relevantes $u_{Z_1}^{f_1}, \dots, u_{Z_L}^{f_L}$ são selecionadas e para cada atribuição z de Z uma nova restrição de desigualdade, $u_z^{er} \geq \sum_{j=1}^L u_{z,x_l}^{f_j} \quad \forall x_l$, é adicionada para $\mathbb{J}$. Finalmente uma restrição de desigualdade com a soma sobre todas as funções com escopo vazio, $0 \geq \sum_i u^{e_i}$ (Expressão (8.15)), é adicionada e um novo conjunto de restrições $\mathbb{J}$ é devolvido.

8.3.3 O algoritmo FACTOREDMPA

FACTOREDMP deve ser aplicado para todas as ações $a \in \mathcal{A}$. Isso é feito pelo algoritmo principal FACTOREDMPA (Algoritmo 16) que resolve o Problema (8.2) gerando um conjunto novo (menor) de restrições **para todas as ações** $a \in \mathcal{A}$. Ao fazer isso, o algoritmo FACTOREDMPA reduz o

Algoritmo 12: FACTOREDMP($\mathcal{C}^a, \mathcal{R}^a, O$)

```

input :  $\mathcal{C}^a$  (set of functions  $c_i^a$ ),
         $\mathcal{R}^a$  (set of local-reward  $R_j^a$ ),
         $O$  (order criterion),
         $n$  (number of state variables)
output:  $\mathbb{J}$  (set of constraints)

1 begin
2    $\mathbb{J} = \{\}$ ;
3    $\mathbb{F} = \{\}$ ; //set of new functions
4    $\mathbb{J}, \mathbb{F} \leftarrow \text{GENERATEEQUALITYCONSFORREWARD}(\mathbb{J}, \mathcal{R}^a, \mathbb{F})$ ; //Algorithm 13
5    $\mathbb{J}, \mathbb{F} \leftarrow \text{GENERATEEQUALITYCONSFOR}c_i(\mathbb{J}, \mathcal{C}^a, \mathbb{F})$ ; //Algorithm 14
6    $\mathbb{J} \leftarrow \text{GENERATEINEQUALITYCONS}(\mathbb{J}, \mathbb{F}, O, n)$ ; //Algorithm 15
7   return  $\mathbb{J}$ ;
8 end

```

Algoritmo 13: GENERATEEQUALITYCONSFORREWARD($\mathbb{J}, \mathcal{R}^a, \mathbb{F}$)

```

input :  $\mathbb{J}$  (set of constraints),
         $\mathcal{R}^a$  (set of local-reward  $R_j^a$ , with  $1 \leq j \leq \psi$ ),
         $\mathbb{F}$  (set of new functions)
output:  $\mathbb{J}, \mathbb{F}$ 

1 begin
2    $r = |\mathbb{F}| + 1$ ;
3   for  $j = 1$  to  $\psi$  do
4     //create a new equality constraint for each assignent to the scope of  $R_j^a$ 
5      $Z = \text{scope of } R_j^a$ ;
6     foreach assignment  $z$  of  $Z$  do
7       create a new variable  $u_z^{e_r}$ ;
8       add a equality constraint  $u_z^{e_r} = R_j^a(z)$  to  $\mathbb{J}$ ;
9        $r = r + 1$ ;
10       $\mathbb{F} = \mathbb{F} \cup u_z^{e_r}$ ;
11    end
12  return  $\mathbb{J}, \mathbb{F}$ ;
13 end

```

Problema de programação multilinear (8.2) com exponencialmente muitas restrições em um novo problema, que tem um conjunto equivalente de restrições, porém menor. (Esta propriedade é de fato herdada do algoritmo FACTOREDLP [Guestrin, 2003].)

O Algoritmo 16 tem como entrada um MDPIP, o conjunto de funções base H e o critério de ordenação O ; e devolve w e $\vec{p}$. Note que com esses valores podemos calcular a função valor de cada estado.

FACTOREDMPA começa construindo a função objetivo *obj* do Problema (8.2) chamando CALCULATEOBJECTIVE. Em seguida, para cada ação a e cada função base $h_i \in H$, calcula c_i^a chamando

Algoritmo 14: GENERATEEQUALITYCONSFORC_i($\mathbb{J}, \mathcal{C}^a, \mathbb{F}$)

```

input :  $\mathbb{J}$  (set of constraints),
         $\mathcal{C}^a$  (set of functions  $c_i^a$ , with  $0 \leq j \leq k$ ),
         $\mathbb{F}$  (set of new functions)
output:  $\mathbb{J}, \mathbb{F}$ 

1 begin
2    $r = |\mathbb{F}| + 1$ ;
3   for  $i = 0$  to  $k$  do
4     //create a new equality constraint for each assignment to the scope of  $c_i^a$ 
5      $Z = \text{scope of } c_i^a$ ;
6     foreach assignment  $z$  of  $Z$  do
7       create a new variable  $u_z^{e_r}$ ;
8       add a multilinear equality constraint  $u_z^{e_r} = c_i^a(z, \vec{p}) * w_i$  to  $\mathbb{J}$ ;
9        $r = r + 1$ ;
10       $\mathbb{F} = \mathbb{F} \cup u_z^{e_r}$ ;
11
12   return  $\mathbb{J}, \mathbb{F}$  ;
13 end

```

COMPUTEC_i^a (Algorithm 11). O conjunto de restrições $\mathbb{J}$ é inicializado com as restrições de probabilidade $\mathfrak{C}$. Após isso, para cada ação a o algoritmo FACTOREDMP é chamado para gerar um novo conjunto menor de restrições que são adicionadas para $\mathbb{J}$. Finalmente, o otimizador não-linear é chamado (algoritmo CALLNONLINEARSOLVER) com a função objetivo e o novo conjunto de restrições $\mathbb{J}$, para resolver um problema multilinear menor e mais simples, devolvendo w e $\vec{p}$.

8.4 Resumo

Neste capítulo mostramos duas formulações, baseadas em programação matemática, para MDP-IPs fatorados que usam uma combinação linear de funções base para aproximar a função valor: programação de dois níveis e programação multilinear.

Como discutimos, a solução do programa de dois níveis está longe de ser trivial, assim concentramos nossos esforços para resolver o programa multilinear. Assim, foi proposto um algoritmo original FACTOREDMPA que estende FACTOREDMPA [Guestrin, 2003] para MDP-IPs fatorados. Note que, se o conjunto de funções base admite uma solução exata, então FACTOREDMPA garante encontrá-la, caso contrário, FACTOREDMPA (assim como FACTOREDMPA [Guestrin, 2003]) não tem garantias de erro de aproximação. Ainda assim, esse método pode ser mais adequado para alguns domínios de aplicação, quando comparados aos métodos aproximados de programação dinâmica, como verificamos no Capítulo 10 de análise experimental.

Algoritmo 15: GENERATEINEQUALITYCONS($\mathbb{J}, \mathbb{F}, O, n$)

input : $\mathbb{J}$ (set of constraints),
 $\mathbb{F}$ (set of new functions),
 O (order criterion),
 n (number of state variables)
output: $\mathbb{J}$

```

1 begin
2    $r = |\mathbb{F}| + 1$ ;
3   for  $i = 1$  to  $n$  do
4     //select the variable to be eliminated
5      $X_l = O(i)$ ;
6     //select the relevant functions
7     select  $u_{Z_1}^{f_1}, \dots, u_{Z_L}^{f_L}$  from  $\mathbb{F}$  whose scope contains  $X_l$ ;
8      $Z = \cup_{j=1}^L Z_j - \{X_l\}$ ;
9      $r = r + 1$ ;
10    //define a new function  $u_Z^{e_r}$ 
11    foreach assignment  $z$  of  $Z$  do
12      create a new variable  $u_z^{e_r}$ ;
13      add a constraint  $u_z^{e_r} \geq \sum_{j=1}^L u_{\{z, x_l\}}^{f_j} \quad \forall x_l$  to  $\mathbb{J}$ ;
14      //add the new function and remove the relevant functions
15       $\mathbb{F} = \mathbb{F} \cup u_z^{e_r} \setminus \{u_{Z_1}^{f_1}, \dots, u_{Z_L}^{f_L}\}$ ;
16
17    //at the end all variables have been eliminated
18    //all remaining functions  $u^{e_i}$  have empty scope
19    add a constraint  $0 \geq \sum_{e_i \in \mathbb{F}} u^{e_i}$  to  $\mathbb{J}$ ;
20    return  $\mathbb{J}$  ;
21 end

```

Algoritmo 16: FACTOREDMPA($MDPIP, H, O$)

input : $MDPIP$ (given by $\langle S, A, T, R, K, \gamma \rangle$,
 H (set of basis functions),
 O (order criterion)
output: $\{w, \vec{p}\}$

```

1 begin
2    $obj = \text{CALCULATEOBJECTIVE}(MDPIP, H);$ 
3   foreach  $action\ a \in \mathcal{A}$  do
4      $\mathcal{C}^a = \text{COMPUTEC}_i^a(MDPIP, H);$  //where  $\mathcal{C}^a = \{c_1^a, \dots, c_k^a\}$ ;
5     //initialize the set of constraints  $\mathbb{J}$  with the probability constraints  $\mathfrak{C}$ 
6      $\mathbb{J} = \mathfrak{C};$ 
7     //the set of constraints  $\mathbb{J}$  is expanded for each action  $a \in \mathcal{A}$ 
8     foreach  $action\ a \in \mathcal{A}$  do
9        $\mathbb{J} = \mathbb{J} \cup \text{FACTOREDMP}(\{c_1^a, \dots, c_k^a\}, \{R_1^a, \dots, R_\psi^a\}, O);$ 
10    //the nonlinear solver is called with the new set of constrains  $\mathbb{J}$ 
11     $\{w, \vec{p}\} = \text{CALLNONLINEARSOLVER}(obj, \mathbb{J});$ 
12    return  $\{w, \vec{p}\}$  ;
13 end

```

Parte III

Resultados Empíricos e Considerações Finais

Capítulo 9

Domínios de Teste

Antes de apresentarmos a análise experimental dos algoritmos SPUDD-IP, APRICODD-IP, OBJECTIVE-IP e FACTOREDMPA, introduzidos nos capítulos anteriores, começamos por descrever os domínios para MDP-IPs usados em nossos experimentos.

Realizamos experimentos em três domínios para problemas modelados como MDP-IPs fatorados: FACTORY [St-aubin et al., 2000], SYSADMIN [Guestrin et al., 2003] e TRAFFIC [Delgado et al., 2009c]. A seguir, especificamos em detalhes a modelagem dos domínios FACTORY e TRAFFIC (SYSADMIN é o mesmo especificado no Capítulo 6, Exemplo 6.1).

9.1 Domínio FACTORY

O domínio FACTORY [St-aubin et al., 2000] é baseado num problema de montagem industrial, em que são produzidas peças conectadas com acabamento. As peças devem ser moldadas, polidas, pintadas e conectadas com parafusos, solda ou colagem. Suponha que a tarefa do agente é conectar dois objetos x e y . O agente pode escolher entre as seguintes ações:

- $shape(x)$, ação usada para moldar o objeto x ;
- $handPaint(x)$, ação para pintar à mão o objeto x ;
- $polish(x)$, ação usada para polir um objeto x ;
- $drill(x)$, ação usada para perfurar um objeto x ;
- $dip(x)$, pintar o objeto x por imersão;
- $bolt(x,y)$, conectar os objetos x e y com parafuso;
- $weld(x,y)$, conectar os objetos x e y com solda;
- $glue(x,y)$, conectar os objetos x e y com cola;
- $sprayPaint(x)$, pintar o objeto x com spray (produz uma pintura de pior qualidade do que $handPaint(x)$).

As propriedades usadas para descrever um estado nesse domínio são:

- *connected*(x, y) e *Wellconnected*(x, y), que representam se os objetos x e y estão conectados de maneira simples (e.g. por colagem) ou estão bem conectados (e.g. por solda ou parafuso). A única razão para que objetos bem conectados passem a estar não conectados é quando eles forem moldados após serem conectados.
- *painted*(x) e *Wellpainted*(x), que representam se o objeto x está pintado de maneira simples (e.g. com spray) ou está bem pintado (e.g. pintado à mão). Um objeto pintado permanece pintado se ele não for moldado, polido ou perfurado após a pintura.
- *shaped*(x), representa que o objeto x está moldado. Peças moldadas permanecem moldadas se elas não forem perfuradas após a moldagem.
- *smooth*(x), indica que o objeto x se torna polido se o agente executa a ação polir com sucesso. Um objeto polido permanece polido se ele não for moldado ou perfurado após o polimento.
- *drilled*(x), representa que um objeto x se torna perfurado se a ação perfurar é aplicada com sucesso.

Existem outras propriedades que descrevem o que está disponível no ambiente e o que pode ser utilizado pelo agente (recursos), tal como: *spraygun* (bomba para pintura à spray), *glue* (cola), *bolts* (parafusos), *drill* (furadeira) e *clamps* (grampos para fixar uma peça numa bancada). Além disso, a variável *skilledlab* representa a existência de trabalhadores qualificados.

A qualidade exigida para o produto acabado é representado pela variável *typeneeded* e pode ser de alta qualidade ou de baixa qualidade. A recompensa depende diretamente da qualidade requerida. Por exemplo, quando alta qualidade é exigida, peças pintadas à mão, perfuradas e aparafusadas terão maior recompensa, enquanto peças pintadas com spray e coladas terão pouca recompensa. As variáveis de estado são definidas criando-se instâncias das propriedades relacionadas para o conjunto de objetos definidos no problema.

A codificação desse domínio conforme descrito acima está disponível na página web do projeto SPUD [Hoey et al., 2009] e foi estendida com probabilidades imprecisas para testarmos os algoritmos propostos nessa tese.

Para obter um MDP-IP fatorado, introduzimos incerteza nas probabilidades de transição da ação *bolt* para a variável *connected-A-B* como segue. A probabilidade de sucesso da ação *bolt* para dois objetos A e B que estavam perfurados quando existem parafusos disponíveis, é p_1 . A probabilidade de sucesso, no caso em que os dois objetos não estão perfurados mas existem parafusos, é p_2 . Essas probabilidades estão limitadas por $0.2 + p_2 \leq p_1 \leq 1$ e $0.5 \leq p_2 \leq 1$. Note que p_1 deve ser sempre igual ou maior que p_2 (uma vez que o processo associado com p_1 é mais provável de ser bem sucedido).

9.2 O Novo Domínio: TRAFFIC

O domínio TRAFFIC [Delgado et al., 2009c] descrito no Capítulo 1 (Exemplo 1.1) foi modelado usando um *modelo dinâmico de transição celular* [Daganzo, 1994], ou seja, as pistas são modeladas

por uma fila de células discretas. Apesar de não ter sido projetado para ser um modelo preciso de tráfego para rodovias em grande escala, esse modelo permite representar a propagação de tráfego em cruzamentos muito movimentados, em que o trânsito de veículos é necessariamente limitado por filas, e por veículos que esperam para virar para direita e outros para esquerda.

Uma representação gráfica com exemplos das variáveis de estado usadas neste modelo é mostrada na Figura 9.1. Codificamos o estado de tráfego como $\vec{x} \in \{O, U\}^n$ que indica que cada célula de tráfego $\alpha \in \vec{x}$ está ocupada (O) ou desocupada (U).

Figura 9.1: Diagrama mostrando a intersecção de 2 pistas com células (quadros pontilhados) que representam as posições dos veículos e as variáveis de estado usadas na nossa descrição. Note que não modelamos células da pista na intersecção, uma vez que assumimos que os carros devam passar livremente pela intersecção, isto é, sem parar. Nesse modelo simplificado as filas possuem 3 células.

Para cada *célula de intersecção* $x_{j_i} \in \vec{x}$ (i.e., a primeira célula da fila), definimos uma variável de estado $t_{j_i} \in \{turn, no-turn\}$ indicando se o veículo na célula x_{j_i} pretende virar para direita ou não. A variável de estado t_{j_i} possui uma probabilidade imprecisa p_t , que descrevemos a seguir.

Como foi descrito no Exemplo 1.1 (Capítulo 1), veículos no início da fila (célula x_{j_i}) podem continuar na mesma pista, virar para direita ou para esquerda; veículos que não estão no início da fila podem avançar para uma posição à sua frente, sempre que esta estiver desocupada. As condições que devem ser satisfeitas para que um veículo no início da fila x_{j_i} possa avançar, virar para direita ou esquerda, dependem de dois fatores: do estado dos semáforos c_1 e c_2 , bem como da intenção do veículo no início da pista de sentido oposto (t_{j_i}). Na Figura 9.1, o veículo na posição x_{j_1} pode avançar ou virar para esquerda sempre que o semáforo c_1 estiver verde. No entanto, o veículo na célula x_{j_1} só pode virar para direita (manobra que cruza a pista oposta) nas seguintes situações: (i) se o semáforo c_1 estiver verde e o semáforo c_2 estiver vermelho; (ii) se c_1 e c_2 estiverem verde e o veículo da célula x_{j_2} também pretende virar para a sua direita. Na Figura 9.1, temos $\vec{c} = (c_1, c_2)$, em que, para simplificar, assumimos que cada variável c_i é binária, representando que

o semáforo está em verde ou não. São definidas variáveis de estado $\vec{c}$ para representar o estado atual dos semáforos.

A última célula de uma pista, x_{k_i} , é chamada de *célula alimentadora de pista*, uma vez que novos carros chegam nestes pontos. Quando a célula não está ocupada assumimos que carros novos chegam com probabilidade imprecisa p_a .

Finalmente, o conjunto de ações é $A = \{change, no-change\}$. Por simplicidade, a seguir especificamos as funções de transição de estados de forma analítica e não através de CPTs.

Transição do ciclo do semáforo O efeito da ação *no-change* e *change* sobre o estado dos semáforos é modelado de maneira simples por:

$$P(\vec{c}'|\vec{c}, a = no-change) \begin{cases} 1 & \vec{c}' = \vec{c} \\ 0 & \text{caso contrário} \end{cases}$$

$$P(\vec{c}'|\vec{c}, a = change) \begin{cases} 1 & \vec{c}' = next-state(\vec{c}) \\ 0 & \text{caso contrário} \end{cases}$$

que significa que a probabilidade do estado dos semáforos $\vec{c}$ permanecer o mesmo é 1 quando a ação *no-change* é executada, e 0, caso contrário. Note que o valor de $\vec{c}$ no próximo estado só depende de $\vec{c}$ no estado anterior.

Indicador do carro virar à direita A probabilidade de um veículo localizado na posição x_{j_i} virar para a direita é 1 se existir um veículo nessa posição ($x_{j_i} = O$) e sua intenção no estado anterior era de virar para direita. A probabilidade é 0, se existir um veículo em x_{j_i} e sua intenção é de não virar para direita. Finalmente a probabilidade de virar para direita é p_t se $x_{j_i} = U$, onde p_t é um parâmetro. Note que $x_{j_i} = U$ significa que essa posição está desocupada (no estado anterior) e assim não sabemos qual era a intenção do veículo que a ocupará no próximo estado.

Em modelos de tráfego é difícil obter uma estimativa precisa de p_t (que poderia mudar ao longo do dia); usando esse DCN permitimos que a probabilidade de virar para direita varie no tempo e assim modelamos como probabilidades imprecisas p_t através das seguintes restrições: $p_t^{\min} \leq p_t \leq p_t^{\max}$ para $0 \leq p_t^{\min} \leq p_t^{\max} \leq 1$ (sendo que $p_t^{\min}$ e $p_t^{\max}$ são definidos para instâncias específicas do problema).

$$P(t'_{j_i} = \text{turn} | t_{j_i}, x_{j_i}) = \begin{cases} 1.0 & x_{j_i} = O \wedge t_{j_i} = \text{turn} \\ 0.0 & x_{j_i} = O \wedge t_{j_i} = \text{no-turn} \\ p_t & x_{j_i} = U \end{cases}$$

Células intermediárias na pista O modelo básico de tráfego usado é que um carro pode avançar para a próxima célula sempre que a célula esteja desocupada. Assim, uma célula intermediária x_i na pista estará ocupada por um carro no próximo estado, se: i) o carro que está ocupando a célula não pode avançar para a próxima célula x_{i+1} , uma vez que ela está ocupada por outro carro ou ii) existe um carro na célula anterior x_{i-1} que pode avançar, para ocupar a célula x_i :

$$P(x'_i = O | x_{i-1}, x_i, x_{i+1}) = \begin{cases} 1.0 & x_i = O \wedge x_{i+1} = O \\ 1.0 & x_i = U \wedge x_{i-1} = O \\ 0.0 & x_i = \text{caso contrário} \end{cases}$$

Células de alimentação de pista Uma célula de alimentação de pista serve como uma entrada para a rede de tráfego. Os carros chegam em cada célula de alimentação desocupada com probabilidade p_a :

$$P(x'_{k_i} = O | x_{k_i}) = \begin{cases} 1.0 & x_{k_i} = O \wedge x_{k_i+1} = O \\ p_a & \text{caso contrário} \end{cases}$$

Em modelos de tráfego, é difícil, obter uma estimativa precisa das probabilidades de chegada p_a em qualquer hora do dia. Assim, permitimos que as probabilidades de chegada variem através de probabilidades imprecisas p_a , dadas por $p_a^{\min} \leq p_a \leq p_a^{\max}$ para $0 \leq p_a^{\min} \leq p_a^{\max} \leq 1$, em que $p_a^{\min}$ e $p_a^{\max}$ são definidas para instâncias específicas do problema.

Células de intersecção de pistas As células de intersecção são as células mais complexas a serem modeladas numa rede de tráfego, uma vez que o comportamento do tráfego depende do estado do semáforo, das células ocupadas com acesso verde à intersecção e das células de outras pistas cujos carros desejam virar. Um modelo simples de comportamento de tráfego foi implementado levando em conta todas essas contingências:

$$P(x'_{j_1} = O | x_{j_1}, t_{j_1}, x_{j_2}, t_{j_2}, \vec{c}) = \begin{cases} 0.0 & x_{j_1} = U \wedge x_{j_1-1} = U \\ 1.0 & x_{j_1} = U \wedge x_{j_1-1} = O \\ 0.0 & x_{j_1} = O \wedge t_{j_1} = no-turn \\ 0.0 & x_{j_1} = O \wedge t_{j_1} = turn \wedge x_{j_2} = U \\ 0.0 & x_{j_1} = O \wedge t_{j_1} = turn \wedge x_{j_2} = O \wedge \neg green(\vec{c}, o) \\ 1.0 & x_{j_1} = O \wedge t_{j_1} = turn \wedge x_{j_2} = O \wedge green(\vec{c}, o) \wedge \\ & t_{j_2} = no-turn \\ 0.0 & x_{j_1} = O \wedge t_{j_1} = turn \wedge x_{j_2} = O \wedge green(\vec{c}, o) \wedge \\ & t_{j_2} = turn \end{cases}$$

Nessa representação, assumimos que existem funções auxiliares $green(\vec{c}, j_i)$ que procuram no vetor $\vec{c}$ a informação se o semáforo está verde para a célula de intersecção j_i (ou não). Assumimos também que $\neg green(\vec{c}, o)$ verdadeira implica em $x'_{j_1} = x_{j_1}$ (uma vez que não se pode virar em vermelho).

Modelo de Recompensa de Traffic Neste domínio, a meta é reduzir o congestionamento de tráfego nas intersecções, o que é equivalente a minimizar a quantidade de células ocupadas na fila de cada cruzamento. Assim, uma recompensa apropriada para maximizar seria a quantidade de células *desocupadas*:

$$R(\vec{x}) = \sum_{i=1}^n \mathbb{I}[X_i = U]$$

que significa que o agente recebe uma recompensa de +1 por cada célula que está desocupada.

Figura 9.2: Rede Credal Dinâmica para o Problema TRAFFIC para as ações *change* e *no-change*.

Rede Credal Dinâmica completa de Traffic Com base nas variáveis descritas para a Figura 9.1, assumindo filas de tamanho 3, a Figura 9.2 especifica a Rede Credal Dinâmica para as ações *change* e *no-change*, uma vez que a dependência entre as variáveis é a mesma para as duas ações.

Instâncias do Problema Traffic Neste trabalho resolvemos instâncias do domínio TRAFFIC com uma pista única de sentido duplo. Nestas instâncias, em particular, definimos a probabilidade mínima de um veículo virar para direita como $p_t^{\min} = 0$ e a probabilidade máxima como $p_t^{\max} = 1$. Além disso, limitamos as probabilidades de virar para direita dos veículos nos dois sentidos, p_1 e p_2 , usando a restrição $|p_1 - p_2| \leq 0.1$, o que faz com que essas probabilidades estejam fortemente

correlacionadas. Adicionalmente, as probabilidades de um carro chegar nas células de alimentação das pistas, dadas por p_3 e p_4 , estão limitadas por $p_a^{\min} = 0.4$, $p_a^{\max} = 0.6$ e restringidas por $|p_3 - p_4| \leq 0.1$.

Capítulo 10

Análise dos Algoritmos Propostos

Neste capítulo, analisamos empiricamente 5 sistemas para resolver problemas formulados como MDP-IPs:

- EXACT FLAT VALUE ITERATION: implementa o algoritmo exato de ITERAÇÃO DE VALOR com estados enumerativos para MDP-IPs não fatorados, especificado formalmente na Solução 5.2 (Seção 5.1.1);
- SPUDD-IP, implementação de uma solução exata baseada em programação dinâmica que usa PADDs, especificado no Algoritmo 7 (Seção 7.1);
- APRICODD-IP, solução aproximada, também baseada em programação dinâmica usando PADDs, e especificada no Algoritmo 7, mas com a chamada adicional da função APPROXADD (Algoritmo 9). A aproximação é feita com erro limitado por $\delta \cdot Vmax$, fazendo-se uma poda na altura dos ADDs;
- OBJECTIVE-IP, outra solução aproximada baseada em programação dinâmica e especificada no Algoritmo 7, mas com a chamada adicional da função APPROXPADDLEAVES (Algoritmo 10). A aproximação é feita com erro limitado por $\delta \cdot Vmax$, fazendo-se uma simplificação dos polinômios nas folhas dos PADDs, satisfazendo o limite de erro;
- FACTOREDMPA, solução baseada em programação matemática que resolve um MDP-IP formulado como um programa multilinear, aproximando a função valor como a soma de funções base (Algoritmos 12 e 16, Capítulo 8), com 1 ou 2 variáveis.

Todos os algoritmos foram implementados em Java. Foi implementada uma biblioteca de PADDs, incluindo as operações soma ($\oplus$), subtração ($\ominus$), multiplicação ($\otimes$), *restrição*, *marginalização* e *redução* dos polinômios nas folhas de um PADD.

Para resolver os sistemas multilineares foi utilizado o otimizador não-linear MINOS [Murtagh and Saunders, 1998] (os algoritmos em Java fazem chamadas ao MINOS). Para os algoritmos de ITERAÇÃO DE VALOR, definimos: o fator de desconto como $\gamma = 0.9$; $maxIter = 50$, para o domínio SYSADMIN; e $maxIter = 75$, para os demais domínios.

A análise realizada dos sistemas implementados foi dividida em três partes:

1. **Comparação entre os métodos de ITERAÇÃO DE VALOR**, a saber: EXACT FLAT VALUE ITERATION Versus SPUDD-IP, em termos de tempo de execução; e APRICODD-IP Versus OBJECTIVE-IP, em termos de tempo de execução, da quantidade de nós do V_{ADD} na última iteração e do erro de aproximação. Esses experimentos foram realizados nos três domínios especificados na Seção 9;
2. **Análise da solução FACTOREDMPA**, em termos: (i) da capacidade de reduzir o número de restrições do programa multilinear (fatorado e aproximado) original; (ii) de tempo de execução quando comparamos FACTOREDMPA com EXACT FLAT VALUE ITERATION e (iii) do erro de aproximação com o uso de funções base com 1 e 2 variáveis de estado. Todos os experimentos foram realizados somente para o domínio SYSADMIN;
3. **Comparação entre os métodos aproximados**, isto é, FACTOREDMPA com os dois métodos aproximados de programação dinâmica, APRICODD-IP e OBJECTIVE-IP, em termos de tempo de execução e erro de aproximação, realizada para os 3 domínios de teste.

10.1 Avaliando Nosso Conjunto de Algoritmos de ITERAÇÃO DE VALOR

10.1.1 EXACT FLAT VALUE ITERATION vs. SPUDD-IP

Na Figura 10.1 comparamos o tempo de execução dos dois métodos de solução exatas: SPUD-IP e EXACT FLAT VALUE ITERATION, isto é, que calculam $V^*(\vec{x})$. Para isso foi preciso modificar o algoritmo de ITERAÇÃO DE VALOR ENUMERATIVO para MDP-IPs (Solução (5.2)), para permitir tratar expressões multilineares na tabela de transição de probabilidades geradas a partir da especificação fatorada dos domínios de teste. Os programas que não terminaram após cinco horas, estão marcadas como *não terminou* (*Did Not Finish - DNF*). Na figura, notamos que SPUD-IP não supera EXACT FLAT VALUE ITERATION no domínio SYSADMIN. Isso pode ser justificado pelo fato da função valor exata para esse domínio ser representada por ADDs com poucas independências do tipo CSI e poucas funções compartilhadas (ou seja, nesse domínio os ADDs reduzidos são ainda muito grandes). Entretanto, TRAFFIC e FACTORY têm funções valor com muitas independências do tipo CSI e funções compartilhadas. Por esse motivo, observamos na Figura 10.1 que SPUD-IP obteve *até duas ordens de magnitude de melhoria no tempo*, principalmente devido à capacidade dos PADDs de agregar objetivos não-lineares comuns, poupando assim um número considerável de chamadas ao otimizador não-linear. Note ainda que, para problemas maiores que traffic-12, o algoritmo EXACT FLAT VALUE ITERATION não consegue terminar.

10.1.2 APRICODD-IP vs. OBJECTIVE-IP

A fim de verificar a escalabilidade dos nossos algoritmos, na Figura 10.2 comparamos o tempo de execução de APRICODD-IP e OBJECTIVE-IP vs. o número de variáveis de estado usando $\delta = 0.1$ (recordando que δ é o máximo erro por iteração com respeito a V_{max}) para FACTORY, TRAFFIC, e três configurações de SYSADMIN. Notamos que OBJECTIVE-IP é mais rápido que APRICODD-IP em *todos* os problemas resolvidos, quando executamos com um limite fixo de erro máximo por iteração (i.e., $\delta = 0.1$).

Figura 10.1: Comparação do tempo de execução dos algoritmos SPUDD-IP e ITERAÇÃO DE VALOR ENUMERATIVA para os problemas TRAFFIC, SYSADMIN e FACTORY. Note que os nomes dos problemas incluem o número de variáveis. Assim, o número correspondente de estados é $2^{\#variables}$.

Para avaliar a política devolvida por nossas soluções AVI (ITERAÇÃO DE VALOR APROXIMADA, calculamos para cada valor fixo de δ , o *Erro de Aproximação Real* (TAE - *True Approximation Error*) dado por:

$$\max_{\vec{x}} |V^*(\vec{x}) - V_{approx}(\vec{x})| \quad (10.1)$$

em que $V_{approx}(\vec{x})$ é o valor devolvido por APRICODD-IP ou OBJECTIVE-IP.

Nos experimentos descritos a seguir executamos SOLVE para um intervalo de valores para δ , com o objetivo de avaliar como a relaxação do erro de aproximação interfere no desempenho dos sistemas AVI. Na Figura 10.3 apresentamos uma comparação detalhada do tempo, tamanho (número de nós no ADD na última iteração), e número de chamadas ao otimizador não-linear requeridas por APRICODD-IP e OBJECTIVE-IP vs. o TAE para *traffic-10*. Notamos pouca relação entre o espaço requerido pela representação ADD e os tempos de execução dos algoritmos (o espaço aumenta ligeiramente para OBJECTIVE-IP, enquanto o tempo de execução diminui), o que mostra que a diminuição de tamanho no ADD do APRICODD-IP não é tão importante quanto a simplificação das folhas do PADD realizada por OBJECTIVE-IP. Mas o que é impressionante sobre essas figuras é que o tempo de execução de cada algoritmo está diretamente relacionado com o número de chamadas ao otimizador não-linear feitas pelo algoritmo (levando até 100 ms em alguns casos), refletindo nossas intuições de que a complexidade de tempo para resolver MDP-IPs é regido pela sobrecarga computacional da otimização não-linear.

Nas Figuras 10.9, 10.10, 10.11, 10.12 e 10.13 mostramos a comparação do Erro de Aproximação Real (TAE) vs. o tempo de execução de três problemas e três diferentes tamanhos de cada problema

Figura 10.2: Tempo de execução de APRICODD-IP e OBJECTIVE-IP para os problemas TRAFFIC, SY-SADMIN e FACTORY com $\delta = 0.1$.

(variando δ). Os resultados aqui levam a uma conclusão: OBJECTIVE-IP consistentemente leva menos tempo do que APRICODD-IP para atingir o mesmo erro de aproximação, apresentando *até uma ordem de magnitude menos tempo* do que APRICODD-IP.

Figura 10.3: Comparação detalhada entre as aproximações realizadas por APRICODD-IP e OBJECTIVE-IP para o problema de tráfego com 10 variáveis. Variando o valor de δ , i.e., resolvemos um único problema variando o limite de erro dado por $\delta \cdot Vmax$

10.2 Avaliando o Algoritmo FACTOREDMPA

Instâncias do problema SYSADMIN introduzido no Capítulo 6 são complexas considerando o modelo de transição dado pela expressão (6.1), o qual tem muitas probabilidades de transição imprecisas. Resolvemos problemas usando dois tipos de funções base: (1) funções base sobre variáveis simples $h_i(X_i = 1) = 1$ e $h_i(X_i = 0) = 0$ e (2) funções base sobre pares de variáveis, que contêm indicadores para cada par de máquinas vizinhas (x_i, x_{i+1}) como a seguir:

$$h_1(x_i, x_{i+1}) = x_i \wedge x_{i+1}$$

$$h_2(x_i, x_{i+1}) = \neg x_i \wedge x_{i+1}$$

$$h_3(x_i, x_{i+1}) = x_i \wedge \neg x_{i+1}$$

$$h_4(x_i, x_{i+1}) = \neg x_i \wedge \neg x_{i+1}$$

Em ambos os casos temos a função base constante $h_0 = 1$.

10.2.1 Número de restrições no programa multilinear: Antes vs. depois de aplicar FACTOREDMPA

Para analisar a escalabilidade do algoritmo proposto, calculamos o número de restrições **antes** (i.e, o número original de restrições) para problemas SYSADMIN e **após** aplicar o algoritmo FACTOREDMPA. A Figura 10.4 mostra o número de restrições para os problemas anel unidirecional envolvendo diferentes números de computadores. Como podemos notar, o número de restrições

Figura 10.4: O número de restrições para o domínio do Administrador de Sistemas com topologia de anel unidirecional: (1) o número original de restrições; (2) as restrições após aplicar o algoritmo FACTOREDMPA com funções base simples; e (3) com funções base pares.

original cresce exponencialmente com o número de computadores, enquanto as restrições geradas após a aplicação do algoritmo FACTOREDMPA cresce quadraticamente. Podemos notar também que o número de restrições geradas por FACTOREDMPA com funções base simples (i.e., funções com uma única variável) e funções base pares têm a mesma taxa de crescimento. Mesmo que, o número de restrições para cada problema nestes dois casos tenha uma diferença de um fator constante (≈ 4), os problemas aproximados por funções pares têm restrições mais complexas. Podemos verificar isso pelos tempos resultantes obtidos ao aplicar nossos algoritmos para resolver esses problemas (Figura 10.5).

10.2.2 EXACT FLAT VALUE ITERATION vs. FACTOREDMPA

Para avaliar a melhoria de desempenho do algoritmo FACTOREDMPA com relação às soluções existentes baseadas em ITERAÇÃO DE VALOR, nas Figuras 10.5 e 10.6 mostramos o tempo de execução para problemas com topologia de anel unidirecional e estrela usando o algoritmo FACTOREDMPA para um conjunto de funções base simples e pares e comparamos esses resultados com as soluções exatas dadas pela nossa implementação do algoritmo de ITERAÇÃO DE VALOR [Satia and Lave Jr., 1973]. Os resultados mostram que a solução exata consome muito tempo quando comparada com a solução dada pelo algoritmo FACTOREDMPA, que é capaz de resolver problemas grandes e complexos (com até 2^{20} estados). Assim, o algoritmo FACTOREDMPA é várias ordens de magnitude mais rápido que a solução exata. Isto ocorre devido à sua capacidade de gerar um conjunto compacto de restrições e portanto termina resolvendo problemas menos complexos.

Para a configuração anel unidirecional (Figura 10.5), o tempo de execução para resolver o problema com 16 computadores (que originalmente incluiria 2^{16} vezes 17 restrições) foi menos do que 17 minutos e para problemas com mais de 16 variáveis o tempo de CPU cresce de maneira

Figura 10.5: Tempo de execução do algoritmo FACTOREDMPA usando funções base simples e pares, e a solução exata para o domínio do Administrador de Sistemas com configuração **Anel Unidirecional**.

Figura 10.6: Tempo de execução do algoritmo FACTOREDMPA usando funções base simples e pares, e a solução exata para o domínio do Administrador de Sistemas com configuração **Estrela**.

rápida. Um possível motivo para essa mudança de comportamento é que o otimizador multilinear alcançou o limite de restrições complexas com que pode lidar.

Para a configuração estrela (Figura 10.6), o tempo de execução para resolver o problema com 16 computadores (com 2^{16} vezes 17 restrições) foi menos do que 5 minutos. Porém, a limitação na memória para gerar entradas grandes não permitiu resolver problemas maiores.

Figura 10.7: Percentagem do Erro de Aproximação Real de FACTOREDMPA para a configuração **Anel Unidirecional** do domínio do Administrador de Sistemas usando funções base pares e simples.

10.2.3 FACTOREDMPA: Funções base pares vs. simples

Nas Figuras 10.7 e 10.8 mostramos a Percentagem do Erro de Aproximação Real (PTAE), que é dado por:

$$PTAE = \frac{TAE}{V_{max}} \quad (10.2)$$

usando funções base pares e simples, para calcular $V^*(\vec{x})$ usamos nossa implementação do algoritmo de ITERAÇÃO DE VALOR. Uma vez que esse algoritmo só pode resolver problemas de tamanho pequeno, mostramos apenas os resultados para problemas em que a solução exata [Satia and Lave Jr., 1973] leva até 10 horas. Note que a percentagem de perda no valor ao usar a aproximação é no máximo 18%, e como esperado, o uso de funções base pares resultou em melhores aproximações para este problema (exceto para o problema com 5 computadores com configuração unidirecional). Essa exceção mostra que, para uma instância específica do problema, o uso de funções base simples devolve uma solução aproximada melhor.

Se observamos adicionalmente o tempo e o percentual de perda no valor, podemos concluir que a percentagem de perda no valor ao usar a aproximação é no máximo 18%, em troca de um aumento de 1000 vezes na velocidade.

10.3 ITERAÇÃO DE VALOR Aproximada vs. Programação Multilinear Aproximada

Nas Figuras 10.9, 10.10, 10.11, 10.12 e 10.13 comparamos os dois métodos de solução de programação dinâmica aproximada, APRICODD-IP e OBJECTIVE-IP, com nossa implementação de programação multilinear aproximada (FACTOREDMPA) para MDP-IPs. Usamos funções base *simples* (uma para cada variável na descrição do problema) e funções base *pares* (uma para cada par

Figura 10.8: Percentagem do Erro de Aproximação Real de FACTOREDMPA para a configuração **Estrela** do domínio do Administrador de Sistemas usando funções base pares e simples.

de variáveis que tem uma variável filha comum no DCN).

FACTOREDMPA leva apenas poucos segundos para produzir uma solução aproximada para cada problema (exceto para o domínio FACTORY, para o qual não devolveu uma solução). Comparando os algoritmos em termos de Erro de Aproximação Real, observamos que: (a) no problema SY-ADMIN (Figuras 10.11, 10.12, 10.13), a solução FACTOREDMPA com funções base *pares* supera APRICODD-IP e obtém uma solução 2 ou 3 vezes maior que o erro de OBJECTIVE-IP, mas em significativamente menos tempo; (b) para o problema TRAFFIC (Figura 10.10), FACTOREDMPA com bases *simples* obtém uma solução com 2 ou 3 vezes mais erro que OBJECTIVE-IP, ainda que em significativamente menos tempo; e (c) no caso do problema FACTORY (Figura 10.9), FACTOREDMPA só foi capaz de resolver uma instância, enquanto OBJECTIVE-IP resolveu o resto dentro do prazo com erro muito menor. Esses resultados nos levam a concluir que OBJECTIVE-IP consistentemente oferece um erro pelo menos 2 ou 3 vezes menor que FACTOREDMPA e por vezes funciona tão rápido quanto FACTOREDMPA, enquanto em outros casos é mais lento.

10.4 Resumo dos Resultados

Para todos os problemas, dado o desempenho imprevisível de FACTOREDMPA (que não tem garantias de erro e muitas vezes não termina dentro do limite de 10 horas) e o pior desempenho geral de APRICODD-IP, o algoritmo OBJECTIVE-IP mostrou ser a opção mais confiável: oferece taxas de erro consistentemente inferiores (a mais baixa que qualquer outro algoritmo) e apresentou tempos de execução razoáveis (se não o mais veloz).

Figura 10.9: Erro de Aproximação Real vs. tempo requerido por APRICODD-IP, OBJECTIVE-IP e FACTOREDMPA com funções base simples (FACTOREDMPA com funções base pares não terminou dado um tempo limite de 10 horas e com funções base simples não terminou para dois problemas) para três problemas FACTORY.

Figura 10.10: Erro de Aproximação Real vs. tempo requerido por APRICODD-IP, OBJECTIVE-IP e FACTOREDMPA com funções base simples e pares para o problema TRAFFIC.

Figura 10.11: Erro de Aproximação Real vs. tempo requerido por APRICODD-IP, OBJECTIVE-IP e FACTOREDMPA com funções base simples e pares para o problema SYSADMIN com configuração Anel Unidirecional.

Figura 10.12: Erro de Aproximação Real vs. tempo requerido por APRICODD-IP, OBJECTIVE-IP e FACTOREDMPA com funções base simples e pares para o problema SYSADMIN com configuração Anel bidirecional.

Figura 10.13: Erro de Aproximação Real vs. tempo requerido por APRICODD-IP, OBJECTIVE-IP e FACTOREDMPA com funções base simples e pares para o problema SYSADMIN com configuração de Anéis bidirecionais independentes.

Capítulo 11

Trabalhos Correlatos

Como foi dito na Introdução, o MDP-IP foi formalmente proposto na década de 70 por Satia e Lave [Satia and Lave Jr., 1973] que introduziram duas soluções baseadas em programação dinâmica para o critério maximin. Por um período de cerca de 20 anos, não foram propostos trabalhos que trouxessem uma melhoria significativa a essas soluções a fim de lidar com a complexidade inerente dos MDP-IPs.

Discutiremos a seguir, os trabalhos correlatos mais relevantes da literatura, divididos em 4 linhas: programação dinâmica, controle robusto, jogos Markovianos e formulação Bayesiana.

11.1 BMDP e MDPST

No final dos anos 90 surgiu o trabalho de Givan et al motivado pela idéia de resolver MDPs para domínios com um número grande de estados através de agregações, definindo um novo modelo chamado de *Bounded-parameter Markov Decision Process (BMDP)* [Givan et al., 2000], um caso especial de um MDP-IP em que as probabilidades e recompensas são especificadas por intervalos constantes. Por exemplo, na Figura 11.1 mostramos um exemplo de BMDP com dois estados s_0 e s_1 em que $P(s_1|s_0, a_1) \in [0.1, 0.5]$. Explorando a estrutura de intervalos num BMDP, o algoritmo de Givan et al [Givan et al., 2000] evita chamar o otimizador não-linear. Isso é feito através de uma *função valor intervalo* usada para limitar a política do conjunto de MDPs exatos associados a um dado BMDP. Soluções recentes para BMDPs incluem extensões de programação dinâmica em tempo real (RTDP) [Buffet and Aberdeen, 2005] e extensões de algoritmos de busca heurística (LAO*) [Cui et al., 2006, Yin et al., 2007].

Markov Decision Process with Set-valued Transitions (MDPSTs) [Trevizan et al., 2007] formam outra sub-classe de MDP-IPs em que distribuições de probabilidade são dadas sobre conjuntos finitos de estados. Por exemplo, na Figura 11.2 mostramos um exemplo de MDPST com dois estados s_0 e s_1 em que $P(\{s_1, s_2, s_3\}|s_0, a) = 0.7$. Trevizan et al [Trevizan et al., 2007] mostraram que a Equação de Bellman (Equação (5.1)) para um MDPST pode ser reduzida para:

$$V^*(s) = \max_{a \in A} \left\{ R(s, a) + \gamma \sum_{conj \in 2^S} P(conj|s, a) \min_{s'} V^*(s') \right\}. \quad (11.1)$$

em que $conj$ é um conjunto de estados. Note que $\min$, comparado com a Equação (5.1), foi

Figura 11.1: Exemplo de BMDP.

Figura 11.2: Exemplo de MDPST.

empurrado para dentro da soma, o que significa que para resolver um MDP-ST é preciso fazer uma otimização local usando um algoritmo guloso. A Equação (11.1) indica que a complexidade de se resolver um MDPST é equivalente à de um MDP. Assim, foi feita uma extensão do RTDP para resolver MDPSTs.

Uma vez que BMDP e MDPST são casos especiais de MDP-IPs podemos representá-los com MDP-IPs enumerativos. Assim, os algoritmos definidos nesta tese claramente se aplicam tanto para BMDPs quanto para MDPSTs. No entanto as soluções para BMDP e MDPST não podem ser generalizadas para um MDP-IP como o ilustrado na Figura 11.3. MDP-IPs permitem restrições lineares gerais entre as probabilidades, que são proibidas nos modelos que usam probabilidades limitadas por intervalos como BMDPs. Esse uso de restrições lineares gerais é particularmente útil quando não sabemos as probabilidades, mas apenas as restrições relativas entre elas (e.g., duas probabilidades no problema TRAFFIC são desconhecidas, mas sabemos que estão altamente correlacionadas).

Trabalhos anteriores em MDP-IPs enumerativos [Satia and Lave Jr., 1973, White III and El-Deib, 1994, Bagnell et al., 2001] que permitem representar conjuntos credais como politopos, propuseram algoritmos baseados em programação dinâmica resolvendo apenas problemas muito pequenos. É importante notar que nosso modelo MDP-IP fatorado é mais expressivo que os MDP-IPs enumerativos simples referidos nesses trabalhos. Como vimos no Capítulo 6, as probabilidades de transição conjunta podem ser não-lineares, enquanto para MDP-IPs enumerativos, a probabilidade

Figura 11.3: Exemplo de MDP-IP.

de transição para qualquer estado sucessor, dado um estado atual e uma ação, pode ser apenas trivialmente linear (um parâmetro singular p_i).

É interessante notar que se permitimos parâmetros definidos por intervalos nas CPTs do MDP-IP fatorado, o resultado é um modelo ainda mais expressivo que um BMDP, i.e., a expressão de transição para qualquer estado sucessor, dado o estado anterior e a ação, pode ser uma expressão multilinear de $\vec{p}$. O modelo resultante pode ser resolvido pelos algoritmos propostos nessa tese.

11.2 Controle Robusto

Outros trabalhos correlatos são relacionados à área de Controle Robusto [Iyengar, 2005, Nilim and El Ghaoui, 2005] que se preocupam com a construção dos conjuntos de probabilidade. A motivação dessa linha de pesquisa é baseada no fato de que os erros estatísticos, originados por estimar as probabilidades precisas a partir de dados históricos, levam à degradação do desempenho de um MDP. Esse problema foi a inspiração de trabalhos na área de controle robusto [Iyengar, 2005, Nilim and El Ghaoui, 2005] que usam conjuntos para modelar ambiguidade nas probabilidades de transição e que adotam a abordagem maximin para obter políticas robustas em problemas de tomada de decisão com horizonte finito e infinito. Esses trabalhos demonstraram que conjuntos credais baseados em regiões de confiança sobre as estimativas de máxima verossimilhança ou limites de entropia permitem obter uma solução com complexidade semelhante a uma versão não-robusta. Assim, eles se concentram em como definir os conjuntos credais. Nesta tese, não estudamos como construir os conjuntos credais, sendo o nosso principal objetivo propor soluções eficientes para MDP-IPs, supondo que os conjuntos credais sejam conhecidos.

11.3 Jogos Markovianos

Uma outra abordagem que está relacionada com MDP-IPs é o *Jogo Alternado de Markov de soma zero de dois jogadores* [Littman, 1994] (*Stochastic Game* [Shapley, 1953]). Esse modelo pode ser visto como um subconjunto de MDP-IPs enumerativos se: um dos jogadores faz o papel da Na-

tureza, variáveis de estado intermediárias são introduzidas para representar as ações do adversário e se os parâmetros que especificam as distribuições sobre as ações do adversário variam no intervalo $[0, 1]$. No entanto, pode ser um desperdício computacional usar um algoritmo de MDP-IP enumerativo ou fatorado para resolver um Jogo Estocástico uma vez que uma minimização sobre um conjunto finito de ações do adversário provavelmente seria computacionalmente mais barato que uma otimização (não-linear) sobre os parâmetros de probabilidade requerida na solução de um MDP-IP. Por isso, computacionalmente parece mais vantajoso usar algoritmos especializados para a solução de Jogos Estocásticos com um conjunto de ações finito para explorar a estrutura específica do problema, do que tentar usar qualquer dos algoritmos de propósito geral para MDP-IP apresentados aqui.

11.4 Formulação Bayesiana

Na perspectiva Bayesiana de um MDP-IP, supomos que são conhecidas distribuições de probabilidade *a priori* sobre P e que elas são atualizadas com novas observações. Por exemplo, dada $p_1 \in K$, $f(p_1)$ é a função de probabilidade para a função de transição probabilística do MDP-IP. Um MDP-IP, nesta formulação, tem hiper-estados (s, Ξ) , em que Ξ é a informação adicional sobre o estado s , que caracteriza a evolução da incerteza, por exemplo, os parâmetros das distribuições *a priori*. As distribuições de probabilidade *a priori* podem ser coletadas a partir de dados empíricos e podem ser atualizadas mediante a aplicação de regra de Bayes. É importante notar que as distribuições são, em geral, contínuas e realizar a integração prevista pela regra de Bayes é uma operação custosa [Duff, 2002]. Por isso, é melhor usar as *famílias conjugadas de distribuições* [Duff, 2002]. Por exemplo, se a distribuição *a priori* é expressa como uma distribuição Beta com parâmetros (α, β) , a distribuição *a posteriori*, dada uma observação, é também uma distribuição Beta, mas com parâmetros que são incrementados para refletir a observação.

Na Figura 11.4 observamos um MDP-IP com 2 estados s_1 e s_2 e duas possíveis ações: *stay* e *change*. Se o agente está no estado s_1 e aplica a ação *stay*, a probabilidade de permanecer no mesmo estado é p_1 (sucesso) e a probabilidade de ir para o estado s_2 é $1 - p_1$ (falha). Neste exemplo, temos quatro processos de Bernoulli, um para cada par estado-ação, e as distribuições *a priori* são expressas como distribuições Beta. Os hiper-estados são representados como: $(s, (\alpha_1, \beta_1), (\alpha_2, \beta_2), (\alpha_3, \beta_3), (\alpha_4, \beta_4))$, em que (α_i, β_i) são os parâmetros da distribuição Beta relacionados à probabilidade de transição p_i . Uma parte do diagrama de transições de hiper-estados para esse exemplo é mostrado na Figura 11.5; em que os nós da árvore são hiper-estados. O hiper-estado inicial é representado por s_1 e os parâmetros das distribuições *a priori*. A execução das ações e os possíveis próximos hiper-estados observados são mostrados na figura, note que os parâmetros são atualizados depois da execução das ações, *stay* e *change*.

Se temos mais de dois estados, a família conjugada adequada a ser usada é uma extensão para muitas dimensões da distribuição Beta, que é chamada de Dirichlet. O problema com esta abordagem é que o número de hiper-estados alcançáveis cresce exponencialmente com o horizonte [Duff, 2002]. Satia [Satia and Lave Jr., 1973] propõe um método *branch and bound*, mantendo limites superior e inferior da recompensa descontada futura para cada hiper-estado (nó) na árvore. Se o

Figura 11.4: Exemplo de um MDP-IP e as suas distribuições *a priori*.

Figura 11.5: Exemplo de diagrama de transição de hiper-estados em que o estado inicial é s_1 .

limite superior do hiper-estado e a ação a_1 é menor que o limite inferior de uma outra ação a_2 , a_2 domina a_1 e como consequência a_1 não precisa ser mais considerada. Com este método Satia só resolve problemas muito pequenos. Duff [Duff, 2002] propõe soluções aproximadas para resolver esta formulação Bayesiana completa recorrendo a técnicas da teoria de aprendizagem por reforço [Duff, 2002]. No entanto, essa solução requer um elevado esforço computacional devido à necessidade de fazer simulações e amostragens.

Capítulo 12

Conclusões e Trabalhos Futuros

Neste trabalho, investigamos Processos de Decisão Markovianos com Probabilidades Imprecisas (MDP-IP), uma classe de modelos que agrega uma grande flexibilidade e realismo ao planejamento probabilístico permitindo a representação de probabilidades imprecisas de transição. Por muitas décadas MDP-IP não pode ser aplicado na solução de problemas reais, devido à sua complexidade computacional. Soluções mais eficientes para MDP-IPs só foram propostas para subclasses menos expressivas desse modelo. Para resolver isso, primeiro propomos um modelo de MDP-IP fatorado e compacto, que representa estados através de variáveis de estado e usa Redes Credais Dinâmicas para especificar as probabilidades imprecisas de transição. Além disso, introduzimos uma nova estrutura de dados, que chamamos de PADD, para representar as tabelas de probabilidades condicionais de uma Rede Credal Dinâmica de um MDP-IP. Assim, essa tese demonstrou que um MDP-IP fatorado é uma representação mais natural e compacta de um MDP-IP, capaz de revelar a estrutura de um domínio de aplicação e possibilitar a construção de soluções eficientes, tanto exatas como aproximadas.

12.1 Resumo das Contribuições

Dada a motivação inicial de resolver problemas do mundo real como MDPs com probabilidades imprecisas no modelo de transição, essa tese resultou em uma série de contribuições originais para as áreas de Inteligência Artificial, Pesquisa Operacional e Controle Ótimo, a saber:

- No Capítulo 6, introduzimos o modelo MDP-IP fatorado baseado em Redes Credais Dinâmicas (DCNs) em que as imprecisões nas probabilidades são representadas por um conjunto de parâmetros definidos nas tabelas de probabilidades condicionais (CPTs) e um conjunto de restrições sobre esses parâmetros.
- Na Seção 6.2, contribuimos com uma estrutura de dados original: o ADD Parametrizado (PADD) que contém folhas com expressões parametrizadas; mostramos como obter de forma eficiente uma representação canônica mínima de um PADD; e mostramos como executar de forma eficiente uma variedade de operações unárias e binárias em PADDs. Assim como os BDDs e ADDs, essa nova estrutura de dados, implementada na forma de uma biblioteca em JAVA, poderá ser usada na implementação de soluções eficientes para uma variedade de classes de problemas diferentes dos resolvidos nessa tese. Portanto, essa também é uma

contribuição em termos de fundamentos na área de Ciência da Computação.

- Na Seção 7.1, desenvolvemos um algoritmo de solução exata para MDP-IP fatorado SPUDD-IP e mostramos como fazer uso de forma eficiente de um PADD em todos os passos do algoritmo de ITERAÇÃO DE VALOR para MDP-IP fatorado. O algoritmo SPUDD-IP produz até duas ordens de magnitude de aumento na velocidade quando comparado às técnicas existentes de programação dinâmica para MDP-IPs conhecidas na literatura.
- Para melhorar ainda mais o algoritmo SPUDD-IP, na Seção 7.2, foram desenvolvidas duas novas extensões de Iteração de Valor aproximada: APRICODD-IP e OBJECTIVE-IP. APRICODD-IP é uma extensão natural baseada em trabalhos anteriores, porém ela não ataca diretamente a principal fonte de complexidade de tempo das soluções de MDP-IPs, isto é, as chamadas ao otimizador não-linear durante a iteração de valor. Com base nessa observação, desenvolvemos um método de aproximação original e alternativo, OBJECTIVE-IP, que aproxima o objetivo das chamadas ao otimizador multilinear. Além disso, provamos a corretude dessa nova abordagem aproximada, tanto na teoria como na prática, em termos da garantia de erro limitado e redução substancial no número de chamadas ao otimizador. Consequentemente, OBJECTIVE-IP apresentou o melhor tempo de execução para Iteração de Valor aproximada, quando comparado a qualquer outro algoritmo existente nessa linha.
- Na Seção 8.3, contribuimos com uma solução aproximada FACTOREDMPA para MDP-IP fatorado formulado como um programa multilinear que usa uma combinação linear de funções base para aproximar a função valor. Esse algoritmo encontra políticas maximin explorando a estrutura dos domínios a fim de reduzir o número de restrições no programa multilinear.
- No Capítulo 10, executamos comparações dos algoritmos propostos em três grandes etapas de experimentos: comparação entre os métodos de ITERAÇÃO DE VALOR, análise da solução FACTOREDMPA e comparação entre os métodos aproximados.

Todas essas contribuições originais — e em particular a criação do algoritmo OBJECTIVE-IP — permitiram a solução (aproximada com erro limitado) de MDP-IPs fatorados que podem *melhorar ordens de magnitude* sobre as abordagens existentes de iteração de valor enumerativa para MDP-IPs e produzem *erros substancialmente menores* que o algoritmo aproximado proposto para MDP-IP (FACTOREDMPA) que não tem garantias de erro *a priori* e depende das funções base dadas. É importante observar que, a princípio, FACTOREDMPA seria a proposta de solução mais promissora para resolver MDP-IPs fatorados para grandes espaços de estados, dado o recente sucesso alcançado pelo trabalho de Guestrin [Guestrin, 2003] para MDPs fatorados.

Devido ao conjunto de contribuições dessa tese, a autora recebeu a premiação de "*Silver IJAR Young Researcher Award for demonstrating excellence in research on imprecise probabilities*" concedida pelos editores do periódico *International Journal of Approximate Reasoning* da Elsevier.

12.2 Publicações Geradas

Os algoritmos baseados em programação dinâmica, SPUDD-IP, APRICODD-IP e OBJECTIVE-IP, foram publicados na conferência internacional ICAPS 2009 (*International Conference on Automated Planning and Scheduling*), principal conferência internacional da área de planejamento em Inteligência Artificial. Uma versão revisada desse artigo foi submetida para o periódico *AI Journal* (*Artificial Intelligence*), principal periódico internacional da área de Inteligência Artificial e um dos mais citados da Ciência da Computação.

A solução baseada em programação matemática, FACTOREDMPA, foi publicada na conferência internacional ISIPTA 2009 (*International Symposium on Imprecise Probability: Theories and Applications*), que reúne os principais pesquisadores da área. A versão estendida desse artigo foi submetida para o periódico IJAR - *International Journal of Approximate Reasoning*.

12.3 Trabalhos Futuros

Entre as possíveis extensões deste trabalho estão:

- **Estender PADD:** Notamos que PADDs representam a ponta do *iceberg* na utilização de técnicas de diagramas de decisão avançadas para resolver MDP-IPs fatorados. Após o sucesso de Affine ADDs (uma extensão de ADDs) para resolver MDPs fatorados [Sanner and McAlister, 2005] com estrutura multiplicativa e aditiva, seria interessante estender essa técnica para PADDs e explorar a mesma estrutura em MDP-IPs. Tais avanços idealmente reduziriam o tempo de execução das soluções para problemas MDP-IP fatorados como TRAFFIC e SYSADMIN que contém estrutura aditiva significativa na sua definição da recompensa e poderiam ser responsáveis por explorar ainda mais a estrutura do problema MDP-IP fatorado.
- **Estender os algoritmos de iteração de valor assíncronos enumerativos:** RTDP [Barto et al., 1995] e BRTDP [McMahan et al., 2005] são algoritmos bem conhecidos, que trabalham com estados enumerativos, para resolver MDPs com estados inicial e final. Para melhorar o desempenho do RTDP, uma variante baseada numa representação fatorada foi proposta, chamada de sRTDP [Feng et al., 2003]. Estamos trabalhando atualmente na implementação de um novo algoritmo, chamado de sBRTDP que explora também a representação compacta de MDPs fatorados e além disso, sorteia as variáveis de estado usando a estrutura dos ADDs. Com base na pesquisa realizada sobre trabalhos correlatos, essa é a primeira tentativa para implementar uma versão fatorada de BRTDP [McMahan et al., 2005] para resolver MDPs fatorados.
- **Investigar outros critérios:** Na formulação de Teoria de Jogos de MDP-IPs existem outros critérios como, “maximax” e “maximix” que merecem maior exploração.
- **Investigar outras soluções para MDP-IP fatorado:** Podemos adaptar algoritmos de iteração de política exato [Boutilier et al., 1995] e aproximados [Koller and Parr, 2000, Patrascu, 2004, Guestrin et al., 2003] de MDPs fatorados para resolver MDP-IPs fatorados.

- **Resolver um MDPST fatorado:** Sendo o MDPST uma subclasse de MDP-IP e sabendo que ele pode ser resolvido com custo computacionalmente semelhante ao de resolver um MDP [Trevizan et al., 2007], pretendemos modificar o BRTDP fatorado descrito acima para resolver problemas modelados por um MDPST, usando ADDs.

Apêndice A

Algoritmos RTDP e BRTDP

Neste apêndice mostramos em mais detalhes os algoritmos RTDP e BRTDP. RTDP (Algoritmo 17) primeiro inicializa $\bar{V}_u$ (valor estimado de V^*) com um limite superior *admissível* V_{u_0} , i.e., $\bar{V}_u(s) \geq V^*(s) \forall s \in S$ e a seguir, executa várias sessões de simulações (passos 5 a 13). Cada sessão começa com um estado aleatório do conjunto de estados iniciais I . Para cada estado encontrado durante uma simulação, o valor do limite superior $\bar{V}_u(s)$ é atualizado e uma ação gulosa é escolhida. Durante uma sessão de simulação, para obter o próximo estado a ser visitado, RTDP sorteia um estado a partir da distribuição da função de transição $P(\cdot|s, a)$, i.e.:

$$\text{CHOOSENEXTSTATE}(s, a) = s' \sim P(\cdot|s, a). \quad (\text{A.1})$$

RTDP termina uma simulação se encontrar um estado meta ou quando uma profundidade limitada é alcançada. Depois disso, todos os valores superiores dos estados visitados armazenados na pilha *visitedStates* são atualizados (passos 16 a 18). RTDP é finalizado quando a convergência for atingida ou quando o tempo destinado à sua execução for excedido.

BRTDP (Algoritmo 18) primeiro inicializa $\bar{V}_u$ e $\bar{V}_l$ com valores superiores e inferiores admissíveis e então executa várias simulações. Cada simulação (passos 6 a 16) começa escolhendo um estado inicial. Para cada estado visitado, os valores superior e inferior são atualizados e uma ação gulosa é escolhida. BRTDP prioriza a escolha do próximo estado de acordo com a diferença entre os valores superior e inferior (Algoritmo 19). BRTDP termina uma simulação se: encontrar um estado meta, se uma profundidade limite for atingida ou se o valor da incerteza for pequeno. Após isso, todos os valores superiores e inferiores dos estados visitados são atualizados (passos 19 a 22). BRTDP é finalizado quando a convergência for atingida ou quando o tempo destinado à sua execução for excedido.

Algoritmo 17: RTDP(MDP, I , G , max_depth , V_{u_0}) [Sanner et al., 2009]

input : MDP, I , G , max_depth , V_{u_0} (Admissible upper bound)

output: $\bar{V}_u$

```

1 begin
2    $\bar{V}_u = V_{u_0}$ ;
3   while convergence not detected and not out of time do
4     //start trial
5     depth=0;
6     visitedStates.Clear();
7     Draw  $s$  from  $I$  at random;
8     while ( $s \notin G$ ) and ( $s \neq null$ ) and ( $depth < max\_depth$ ) do
9       depth=depth+1;
10      visitedStates.Push( $s$ );
11       $\bar{V}_u(s) = \text{BACKUP}(\bar{V}_u, s)$  see Equations (3.7) and (3.8);
12       $a = \text{GREEDYACTION}(\bar{V}_u, s)$  see Equation (3.3);
13       $s = \text{CHOOSENEXTSTATE}(s, a)$  see A.1;
14    //end trial
15    //update visited states
16    while  $\neg \text{visitedStates.Empty}()$  do
17       $s = \text{visitedStates.Pop}()$ ;
18       $\bar{V}_u(s) = \text{BACKUP}(\bar{V}_u, s)$ ;
19
20  return  $\bar{V}_u$ ;
21 end

```

Algoritmo 18: BRTDP(MDP, $I, G, \max_depth, V_{u_0}, V_{l_0}, \tau$) [Sanner et al., 2009]

input : MDP, $I, G, \max_depth, V_{u_0}$ (Admissible upper bound), V_{l_0} (Admissible lower bound), τ (constant greater than 1)
output: $\bar{V}_u$

```

1 begin
2    $\bar{V}_u = V_{u_0};$ 
3    $\bar{V}_l = V_{l_0};$ 
4   while convergence not detected and not out of time do
5     //start trial
6     depth=0;
7     visitedStates.Clear();
8     Draw  $s$  from  $I$  at random;
9      $s_0 = s;$ 
10    while ( $s \notin G$ ) and ( $s \neq \text{null}$ ) and ( $\text{depth} < \max\_depth$ ) do
11      depth=depth+1;
12      visitedStates.Push( $s$ );
13       $\bar{V}_u(s) = \text{BACKUP}(\bar{V}_u, s)$  see Equations (3.7) and (3.8);
14       $\bar{V}_l(s) = \text{BACKUP}(\bar{V}_l, s)$  see Equations (3.7) and (3.8);
15       $a = \text{GREEDYACTION}(\bar{V}_u, s)$  see Equation (3.3);
16       $s = \text{CHOOSENEXTSTATEBRTDP}(s_0, s, a, \tau)$  see Algorithm 19;
17    //end trial
18    //update visited states
19    while  $\neg \text{visitedStates.Empty}()$  do
20       $s = \text{visitedStates.Pop}();$ 
21       $\bar{V}_u(s) = \text{BACKUP}(\bar{V}_u, s);$ 
22       $\bar{V}_l(s) = \text{BACKUP}(\bar{V}_l, s);$ 
23
24  return  $\bar{V}_u;$ 
25 end

```

Algoritmo 19: CHOOSENEXTSTATEBRTDP(s_0, s, a, τ)

input : s_0, s, a, τ
output: s'

```

1 begin
2    $\forall s', b(s') = P(s'|s, a)(\bar{V}_u(s') - \bar{V}_l(s'));$ 
3    $B = \sum_{s'} b(s');$ 
4   if  $B < \frac{\bar{V}_u(s_0) - \bar{V}_l(s_0)}{\tau}$  then
5     return null;
6   return  $s' \sim \frac{b(\cdot)}{B};$ 
7 end

```

Referências Bibliográficas

- [Andreani et al., 2007] Andreani, R., Birgin, E. G., Martinez, J. M., e Schuverdt, M. L. (2007). Augmented Lagrangian methods under the constant positive linear dependence constraint qualification. *Mathematical Programming*, 111(1):5–32.
- [Andreani et al., 2009] Andreani, R., Castro, S. L., Chela, J. L., Friedlander, A., e Santos, S. A. (2009). An inexact-restoration method for nonlinear bilevel programming problems. *Computational Optimization and Applications*, 43(3):307–328.
- [Bagnell et al., 2001] Bagnell, J. A., Ng, A. Y., e Schneider, J. G. (2001). Solving uncertain Markov decision processes. Technical report, Carnegie Mellon University.
- [Bahar et al., 1993] Bahar, R. I., Frohm, E. A., Gaona, C. M., Hachtel, G., Macii, E., Pardo, A., e Somenzi, F. (1993). Algebraic decision diagrams and their applications. *International Conference on Computer-Aided Design (ICCAD)*, pages 188–191, Los Alamitos, CA, USA. IEEE Computer Society Press.
- [Bard, 1988] Bard, J. F. (1988). Convex two-level optimization. *Mathematical Programming*, 40(1):15–27.
- [Barto et al., 1995] Barto, A., Bradtke, S., e Singh, S. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1-2):81–138.
- [Bellman, 1957] Bellman, R. E. (1957). *Dynamic Programming*. Princeton University Press, USA.
- [Berger, 1985] Berger, J. (1985). *Statistical Decision Theory and Bayesian Analysis*. Springer-Verlag.
- [Bialas and Karwan, 1978] Bialas, W. e Karwan, M. (1978). Multilevel linear programming. Technical Report 78-1, Operations Research Program, Department of Industrial Engineering, State University of New York at Buffalo.
- [Bonet and Geffner, 2003] Bonet, B. e Geffner, H. (2003). Labeled RTDP: Improving the convergence of real-time dynamic programming. *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 12–21. AAAI Press.
- [Bonet and Geffner, 2005] Bonet, B. e Geffner, H. (2005). mGPT: A probabilistic planner based on heuristic search. *Journal of Artificial Intelligence Research*, 24(1):933–944.
- [Boutilier et al., 1995] Boutilier, C., Dearden, R., e Goldszmidt, M. (1995). Exploiting structure in policy construction. *Fourteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1104–1111.

- [Boutilier et al., 1996] Boutilier, C., Friedman, N., Goldszmidt, M., e Koller, D. (1996). Context-specific independence in Bayesian networks. *12th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 115–123.
- [Boutilier et al., 1999] Boutilier, C., Hanks, S., e Dean, T. (1999). Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94.
- [Boyd et al., 2009] Boyd, S., Kim, S.-J., Vandenberghe, L., e Hassibi, A. (2009). A tutorial on geometric programming, http://www.stanford.edu/~boyd/papers/gp_tutorial.html.
- [Bryant, 1986] Bryant, R. E. (1986). Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691.
- [Bryant, 1992] Bryant, R. E. (1992). Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318.
- [Buffet and Aberdeen, 2005] Buffet, O. e Aberdeen, D. (2005). Robust planning with LRTDP. *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1214–1219.
- [Cimatti et al., 1997] Cimatti, A., Giunchiglia, F., Giunchiglia, E., e Traverso, P. (1997). Planning via model checking: A decision procedure for $\mathcal{AR}$. *European Conference on Planning (ECP)*, pages 130–142.
- [Cimatti et al., 1998] Cimatti, A., Roveri, M., e Traverso, P. (1998). Strong planning in non-deterministic domains via model checking. *Artificial Intelligence Planning Systems*, pages 36–43.
- [Colson et al., 2005] Colson, B., Marcotte, P., e Savard, G. (2005). A trust-region method for nonlinear bilevel programming: Algorithm and computational experience. *Computational Optimization and Applications*, 30(3):211–227.
- [Colson et al., 2007] Colson, B., Marcotte, P., e Savard, G. (2007). An overview of bilevel optimization. *Annals of Operations Research*, 153(1):235–256.
- [Cozman, 2000] Cozman, F. G. (2000). Credal networks. *Artificial Intelligence*, 120(2):199–233.
- [Cozman, 2005a] Cozman, F. G. (2005a). Graphical models for imprecise probabilities. *International Journal of Approximate Reasoning*, 39(2-3):167–184.
- [Cozman, 2005b] Cozman, F. G. (2005b). Notas de aula da disciplina Probabilidades em Inteligência Artificial, Escola Politécnica - USP.
- [Cui et al., 2006] Cui, S., Sun, J., Yin, M., e Lu, S. (2006). Solving uncertain Markov decision problems: An interval-based method. *Second International Conference Advances in Natural Computation (ICNC)*, pages 948–957.
- [Daganzo, 1994] Daganzo, C. F. (1994). The cell transmission model: A dynamic representation of highway traffic consistent with the hydrodynamic theory. *Transportation Research, Part B*, 28(4):269–287.
- [Daniele et al., 1999] Daniele, M., Traverso, P., e Vardi, M. Y. (1999). Strong cyclic planning revisited. *European Conference on Planning (ECP)*, pages 35–48.

- [de Campos, 2005] de Campos, C. P. (2005). *Redes credais e qualitativas: Complexidade e algoritmos*. PhD thesis, University of São Paulo, Brazil.
- [de Farias and Roy, 2004] de Farias, D. P. e Roy, B. V. (2004). On constraint sampling in the linear programming approach to approximate dynamic programming. *Mathematics of Operations Research*, 29(3):462–478.
- [Dean and Kanazawa, 1990] Dean, T. e Kanazawa, K. (1990). A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150.
- [Delgado et al., 2008] Delgado, K. V., de Barros, L. N., e Cozman, F. G. (2008). Factored Markov decision processes with imprecise probabilities: A multilinear solution. *Doctoral Consortium in International Conference on Automated Planning and Scheduling (ICAPS)*, Australia. Poster.
- [Delgado et al., 2010] Delgado, K. V., de Barros, L. N., Cozman, F. G., e Sanner, S. (2010). Using mathematical programming to solve factored Markov decision processes with imprecise probabilities. *submetido para International Journal of Approximate Reasoning*.
- [Delgado et al., 2009a] Delgado, K. V., de Barros, L. N., Cozman, F. G., e Shirota, R. (2009a). Representing and solving factored Markov decision processes with imprecise probabilities. *6th International Symposium on Imprecise Probability: Theories and Applications (ISIPTA)*, pages 169–178, Durham, United Kingdom. SIPTA.
- [Delgado et al., 2009b] Delgado, K. V., Sanner, S., e de Barros, L. N. (2009b). Efficient solutions to factored MDPs with imprecise transition probabilities. *submetido para Artificial Intelligence Journal*.
- [Delgado et al., 2009c] Delgado, K. V., Sanner, S., de Barros, L. N., e Cozman, F. G. (2009c). Efficient solutions to factored MDPs with imprecise transition probabilities. *19th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 98–105, Thessaloniki, Greece.
- [Dolgov and Durfee, 2006] Dolgov, D. A. e Durfee, E. H. (2006). Symmetric approximate linear programming for factored MDPs with application to constrained problems. *Annals of Mathematics and Artificial Intelligence*, 47(3-4):273–293.
- [Drenick, 1992] Drenick, R. F. (1992). Multilinear programming: Duality theories. *J. Optim. Theory Appl.*, 72(3):459–486.
- [Duff, 2002] Duff, M. O. (2002). *Optimal learning: Computational procedures for Bayes-adaptive Markov decision processes*. PhD thesis, University of Massachusetts Amherst.
- [Feng et al., 2003] Feng, Z., Hansen, E. A., e Zilberstein, S. (2003). Symbolic generalization for on-line planning. *19th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 209–216.
- [Givan et al., 2000] Givan, R., Leach, S., e Dean, T. (2000). Bounded-parameter Markov decision processes. *Artificial Intelligence*, 122(1-2):71–109.
- [Guestrin, 2003] Guestrin, C. (2003). *Planning under uncertainty in complex structured environments*. PhD thesis, Stanford University. Adviser-Daphne Koller.
- [Guestrin et al., 2003] Guestrin, C., Koller, D., Parr, R., e Venkataraman, S. (2003). Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research*, 19:399–468.

- [Hoey et al., 1999] Hoey, J., St-aubin, R., Hu, A., e Boutilier, C. (1999). SPUDD: Stochastic planning using decision diagrams. *Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 279–288. Morgan Kaufmann.
- [Hoey et al., 2009] Hoey, J., St-aubin, R., Hu, A., e Boutilier, C. (2009). Projeto SPUDD, <http://www.computing.dundee.ac.uk/staff/jessehoey/spudd/index.html>.
- [Howard, 1960] Howard, R. A. (1960). *Dynamic Programming and Markov Process*. The MIT Press.
- [Iyengar, 2005] Iyengar, G. (2005). Robust dynamic programming. *Math. Oper. Res*, 30(2):257–280.
- [Kikuti et al., 2005] Kikuti, D., Cozman, F. G., e de Campos, C. P. (2005). Partially ordered preferences in decision trees: Computing strategies with imprecision in probabilities. *IJCAI Workshop on Advances in Preference Handling*, pages 118–123, Edinburgh, United Kingdom.
- [Knight, 1921] Knight, F. H. (1921). *Risk, Uncertainty and Profit*. Houghton Mifflin Company, Boston.
- [Koller and Parr, 1999] Koller, D. e Parr, R. (1999). Computing factored value functions for policies in structured MDPs. *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1332–1339.
- [Koller and Parr, 2000] Koller, D. e Parr, R. (2000). Policy iteration for factored MDPs. *16th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 326–334, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- [Littman, 1994] Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. *Eleventh International Conference on Machine Learning*, pages 157–163. Morgan Kaufmann.
- [Littman et al., 1995] Littman, M. L., Dean, T. L., e Kaelbling, L. P. (1995). On the complexity of solving Markov decision problems. *Eleventh International Conference on Uncertainty in Artificial Intelligence*, pages 394–402.
- [Lukatskii and Shapot, 2000] Lukatskii, A. M. e Shapot, D. V. (2000). Problems in multilinear programming. *Computational Mathematics and Mathematical Physics*, 41(5):638–648.
- [Magalhães and de Lima, 2008] Magalhães, M. N. e de Lima, A. P. (2008). *Noções de Probabilidade e Estatística*. Editora da Universidade de São Paulo.
- [Mahadevan, 2005] Mahadevan, S. (2005). Samuel meets Amarel: Automating value function approximation using global state space analysis. *AAAI Conference on Artificial Intelligence*, pages 1000–1005.
- [Manne, 1960] Manne, A. S. (1960). Linear programming and sequential decision models. *Management Science*, volume 6(3), pages 259–267.
- [McCarthy, 1968] McCarthy, J. (1968). Situations, actions, and causal laws. In Minsky, M., editor, *Semantic Information Processing*, pages 410–418. The MIT Press, Cambridge, MA.

- [McMahan et al., 2005] McMahan, H. B., Likhachev, M., e Gordon, G. J. (2005). Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. *22nd International Conference on Machine Learning (ICML)*, pages 569–576, New York, NY, USA. ACM.
- [Murtagh and Saunders, 1998] Murtagh, B. A. e Saunders, M. A. (1998). MINOS 5.5 user’s guide. Technical Report SOL 83-20R, Systems Optimization Laboratory, Department of Operations Research, Stanford University, California.
- [Newell and Simon, 1963] Newell, A. e Simon, H. (1963). GPS, a program that simulates human thought. In Feigenbaum, E. e Feldman, J., editors, *Computers and Thought*, pages 279–293. McGraw-Hill, New York.
- [Nilim and El Ghaoui, 2005] Nilim, A. e El Ghaoui, L. (2005). Robust control of Markov decision processes with uncertain transition matrices. *Oper. Res.*, 53(5):780–798.
- [Papadimitriou and Tsitsiklis, 1987] Papadimitriou, C. e Tsitsiklis, J. N. (1987). The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450.
- [Patrascu, 2004] Patrascu, R. (2004). *Linear approximations for factored Markov decision processes*. PhD thesis, University of Waterloo.
- [Pearl, 1988] Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- [Pereira and de Barros, 2008] Pereira, S. L. e de Barros, L. N. (2008). A logic-based agent that plans for extended reachability goals. *Autonomous Agents and Multi-Agent Systems*, 16(3):327–344.
- [Puterman, 1994] Puterman, M. L. (1994). *Markov Decision Processes*. John Wiley and Sons, New York.
- [Russell and Norvig, 2002] Russell, S. e Norvig, P. (2002). *Artificial Intelligence: A Modern Approach*, 2nd ed. Prentice-Hall, NJ, USA.
- [Sanner et al., 2009] Sanner, S., Goetschalckx, R., Driessens, K., e Shani, G. (2009). Bayesian real-time dynamic programming. *21st International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1784–1789, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- [Sanner and McAllester, 2005] Sanner, S. e McAllester, D. (2005). Affine algebraic decision diagrams (AADDs) and their application to structured probabilistic inference. *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1384–1390.
- [Satia and Lave Jr., 1973] Satia, J. K. e Lave Jr., R. E. (1973). Markovian decision processes with uncertain transition probabilities. *Operations Research*, 21:728–740.
- [Savard and Gauvin, 1994] Savard, G. e Gauvin, J. (1994). The steepest descent direction for the nonlinear bilevel programming problem. *Operations Research Letters*, 15(5):265–272.
- [Shapley, 1953] Shapley, L. S. (1953). Stochastic games. *National Academy of Sciences*, 39:327–332.

- [Sherali and Tuncbilek, 1992] Sherali, H. D. e Tuncbilek, C. H. (1992). A global optimization algorithm for polynomial programming problems using a reformulation-linearization technique. *Global Optimization*, 2:101–112.
- [Shirota et al., 2007] Shirota, R., Cozman, F. G., Trevizan, F. W., de Campos, C. P., e de Barros, L. N. (2007). Multilinear and integer programming for Markov decision processes with imprecise probabilities. *5th International Symposium on Imprecise Probability: Theories and Applications (ISIPTA)*, pages 395–404, Prague, Czech Republic.
- [Smith and Simmons, 2006] Smith, T. e Simmons, R. (2006). Focused real-time dynamic programming for MDPs: Squeezing more out of a heuristic. *21st National Conference on Artificial Intelligence (AAAI)*, pages 1227–1232. AAAI Press.
- [St-aubin et al., 2000] St-aubin, R., Hoey, J., e Boutilier, C. (2000). APRICODD: Approximate policy construction using decision diagrams. *Advances in Neural Information Processing Systems (NIPS)*, pages 1089–1095. MIT Press.
- [Trevizan et al., 2007] Trevizan, F. W., Cozman, F. G., e de Barros, L. N. (2007). Planning under risk and Knightian uncertainty. *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2023–2028, Hyderabad, India.
- [White III and El-Deib, 1994] White III, C. C. e El-Deib, H. K. (1994). Markov decision processes with imprecise transition probabilities. *Operations Research*, 42(4):739–749.
- [Yin et al., 2007] Yin, M., Wang, J., e Gu, W. (2007). Solving planning under uncertainty: quantitative and qualitative approach. *IFSA (2)*, pages 612–620.
- [Zhang and Poole, 1994] Zhang, N. L. e Poole, D. (1994). A simple approach to Bayesian network computations. *Tenth Canadian Conference on Artificial Intelligence*, pages 171–178.