Processos de Decisão de Markov Limitados por Linguagem

Este exemplar corresponde à redação final da Tese devidamente corrigida e defendida por Jerônimo Pellegrini e aprovada pela Banca Examinadora.

Campinas, 01 de Dezembro de 2006.

Jacques Wainer – IC/Unicamp (Orientador)

Tese apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação.

Substitua pela ficha catalográfica

(Esta página deve ser o verso da página anterior mesmo no caso em que não se imprime frente e verso, i.é., até 100 páginas.)

Substitua pela folha com as assinaturas da banca

Instituto de Computação
Universidade Estadual de Campinas

Processos de Decisão de Markov Limitados por Linguagem

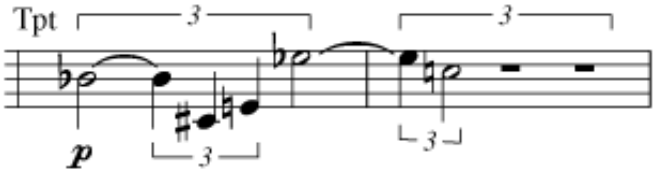
Jerônimo Pellegrini[1]

Dezembro de 2006

Banca Examinadora:

- Jacques Wainer – IC/Unicamp (Orientador)
- Leliane Nunes de Barros – IME/USP
- Carlos H. Ribeiro – ITA
- Siome Klein Goldenstein – IC/Unicamp
- Arnaldo Vieira Moura – IC/Unicamp

[1]**Parcialmente financiado pela CAPES**



Frase do trumpete em “The Unanswered Question”, de Charles Ives.

Resumo

Processos de decisão de Markov (MDPs) são usados para modelar situações onde é necessário executar ações em sequência em ambientes com incerteza. Este trabalho define uma nova formulação dos processos de decisão de Markov, adicionando a estes a possibilidade de restringir as ações e observações a serem consideradas a cada época de decisão. Estas restrições são descritas na forma de um autômato finito – assim, a sequência de possíveis ações e observações consideradas na busca pela política ótima passa a ser uma linguagem regular. Chamamos estes processos de Markov *limitados por linguagem* (LL-MDPs e LL-POMDPs). O uso de autômatos para a especificação de restrições facilita o processo de modelagem de problemas. Apresentamos diferentes abordagens para a solução destes problemas, e comparamos seus desempenhos, mostrando que a solução é viável, e mostramos também que em algumas situações o uso de restrições pode ser usado para acelerar a busca por uma solução. Além disso, apresentamos uma modificação nos LL-POMDPs de forma que seja possível especificar duração probabilística discreta para as ações e observações.

Abstract

Markov decision processes (MDPs) are used to model situations where one needs to execute sequences of actions under uncertainty. This work defines a new formulation of Markov decision processes, with the possibility of restricting the actions and observations to be considered at each decision epoch. These restrictions are described as a finite automaton, so the sequence of possible actions (and observations) considered during the search for an optimal policy is a regular language. We call these "*language limited Markov decision processes* (LL-MDPs and LL-POMDPs). The use of automata for specifying restrictions helps make the modeling process easier. We present different approaches to solve these problems, and compare their performance, showing that the solution is feasible, and we also show that in some situations some restrictions can be used to speed up the search for a solution. Besides that, we also present one modification on LL-POMDPs to make it possible to specify probabilistic discrete duration for actions and observations.

Agradecimentos

Como Tony Cassandra diz nos agradecimentos em sua tese de doutorado, “há tanta gente para agradecer que ao tentar incluir agradecimentos a todos sempre se corre o risco de esquecer alguém. A alternativa é deixar de lado os agradecimentos, mas com toda a ajuda que tive, seria absolutamente injusto. Farei os agradecimentos, então, àqueles que não esqueci”.

Agradeço ao meu orientador, Jacques, pelo trabalho de orientação e pela paciência. Eu não poderia deixar de agradecer também a todos os alunos, professores e funcionários do IC, pelo ambiente produtivo e pelo apoio que sepre tive quando precisei. As amizades que desenvolvi na Unicamp são inestimáveis.

Devo agradecimentos também ao professor Pedro Paulo Ayrosa, da Universidade Estadual de Londrina, por ter despertado meu interesse pela Inteligência Artificial.

Se Anthony Cassandra não tivesse desenvolvido o programa `pomdp-solve` esta tese provavelmente não teria sido escrita. O Tony sempre se mostrou disposto a ajudar com problemas relacionados ao `pomdp-solve`. Também agradeço a Trey Smith por disponibilizar o código do `zmdp` e a Joelle Pineau pelo código fonte do PBVI.

Agradeço à Mariana, meu amor, pela paciência e pelo apoio. Sem ela eu certamente não haveria chegado a lugar algum.

Os anos iniciais do doutorado, quando este trabalho começou a ser desenvolvido, foram financiados pela CAPES.

Agradeço finalmente a todas as outras pessoas que, de diversas maneiras, contribuíram para que este trabalho fosse concluído.

Sumário

Resumo **ix**

Abstract **xi**

Agradecimentos **xiii**

1 Introdução **1**

2 Processos de Decisão de Markov **5**
2.1 Políticas 6
2.2 Algoritmos 9
2.2.1 Iteração de Valores 9
2.2.2 Iteração de Políticas 12
2.2.3 Programação Linear 14
2.2.4 RTDP 15
2.2.5 Decomposição hierárquica 16
2.2.6 Outros métodos 19
2.3 Processos de Decisão Semi-Markovianos 20
2.3.1 Algoritmos 22
2.4 Sumário 23

3 Processos de Decisão de Markov Parcialmente Observáveis **25**
3.1 Dinâmica de sistemas modelados como POMDPs 28
3.1.1 Modelos de observação 29
3.2 POMDPs como MDPs sobre estados de crença 30
3.3 Política 31
3.3.1 Hiperplanos 32
3.3.2 Discretização 33
3.3.3 Planos condicionais 33
3.3.4 Controladores finitos 34

3.3.5 Aproximação por redes neurais . . . 35
3.3.6 Históricos limitados . . . 35
3.4 Algoritmos . . . 35
3.4.1 Iteração de Valores . . . 36
3.4.2 Iteração de Políticas . . . 42
3.4.3 Discretização do espaço de crença . . . 47
3.4.4 Decomposição . . . 55
3.4.5 Outros algoritmos . . . 59
3.5 Processos de Decisão Semi-Markovianos Parcialmente Observáveis . . . 60
3.5.1 POSMDPs . . . 60
3.5.2 Política para um POSMDP . . . 60
3.5.3 Algoritmos . . . 61
3.6 Sumário . . . 61

4 Processos de Decisão de Markov Limitados por Linguagem 63
4.1 Restrições para ações e observações . . . 63
4.2 LL-MDPs . . . 65
4.2.1 Política para um LL-MDP . . . 66
4.2.2 Recompensa para um LL-MDP . . . 69
4.3 Algoritmos . . . 70
4.3.1 Multiplicação de estados . . . 70
4.3.2 Adaptação de algoritmos existentes . . . 71
4.3.3 Iteração de valores (LLVI) . . . 71
4.3.4 Iteração de Políticas (LLPI) . . . 77
4.3.5 Programação Linear . . . 81
4.4 Sumário . . . 83

5 POMDPs Limitados por Linguagem 85
5.1 LL-POMDPs . . . 85
5.1.1 Política para um LL-POMDP . . . 86
5.1.2 Recompensa para um LL-POMDP . . . 88
5.2 Algoritmos . . . 88
5.2.1 Multiplicação estados . . . 88
5.2.2 Adaptações dos algoritmos existentes . . . 91
5.2.3 Iteração de valores (LLVI) . . . 91
5.2.4 Algoritmos baseados em discretização . . . 97
5.3 Duração Probabilística Discreta para Eventos . . . 98
5.3.1 Política . . . 99
5.3.2 Algoritmos . . . 99

5.3.3 Determinação das unidades de tempo . . . 101
5.3.4 Usando a política . . . 101
5.4 Sumário . . . 101

6 Resultados experimentais **103**
6.1 Problemas modelados como LL-MDPs . . . 104
6.1.1 Administrador de sistemas . . . 104
6.1.2 Robô coletor de rochas (I) . . . 105
6.2 Problemas modelados como LL-POMDPs . . . 106
6.2.1 O problema do Tigre . . . 106
6.2.2 Navegação de robô . . . 106
6.2.3 Diagnóstico e Tratamento de uma Doença . . . 108
6.2.4 Robô coletor de rochas (II) . . . 110
6.3 Representação dos LL-MDPs . . . 111
6.4 Tempo de execução para LL-MDPs . . . 112
6.4.1 Noção de convergência . . . 112
6.4.2 Horizonte finito . . . 112
6.4.3 Horizonte infinito . . . 113
6.5 Representação dos LL-POMDPs . . . 114
6.6 Tempos de execução para LL-POMDPs . . . 114
6.6.1 Noção de convergência . . . 114
6.6.2 Horizonte finito . . . 115
6.6.3 Horizonte infinito . . . 117
6.7 Discussão . . . 118
6.7.1 Limitações como otimizações . . . 118
6.7.2 Limitações como ferramenta de modelagem . . . 120
6.7.3 LL-POMDPs com duração para eventos . . . 120
6.8 Sumário . . . 121

7 Trabalhos Futuros **123**
7.1 Decomposição hierárquica . . . 124
7.2 Processos de decisão Semi-Markovianos . . . 124
7.3 Autômatos temporizados . . . 125
7.4 Linguagens livres de contexto . . . 125
7.5 Mais algoritmos . . . 125
7.6 Compressão de estados . . . 125
7.7 Sumário . . . 126

8 Conclusão **127**

Bibliografia **129**

Lista de Tabelas

2.1 Probabilidades de transição em um MDP. 15
2.2 Recompensas em um MDP. 15

3.1 Evolução do tamanho da representação da política gerada pelo algoritmo de Sondik, com $|A| = 4$ e $|\Omega| = 3$. 38
3.2 Algoritmos para solução de POMDPs. 62

6.1 Ações e recompensas para o exemplo de diagnóstico e tratamento médico. . 109
6.2 Função de transição para o exemplo de diagnóstico e tratamento médico. . 109
6.3 Função observação para o exemplo de diagnóstico e tratamento médico. . . 109
6.4 Número de estados nos LL-MDPs dos experimentos. 111
6.5 Desempenho dos algoritmos para a solução de LL-MDPs com horizonte finito. 113
6.6 Desempenho dos algoritmos para a solução de LL-MDPs com horizonte infinito. 113
6.7 Número de estados nos LL-POMDPs dos experimentos. 114
6.8 Desempenho dos algoritmos para a solução de LL-POMDPs com horizonte finito. 116
6.9 Desempenho dos algoritmos para a solução de LL-POMDPs com horizonte infinito. 119
6.10 Tempos de execução para obtenção de políticas de horizonte finito com durações nas ações. 121
6.11 Número de vetores α por iteração ao resolver o problema do diagnóstico. . 121
6.12 Problemas modelados para experimentos. 122

7.1 Implementações de algoritmos para solução de POMDPs. 123

Lista de Figuras

2.1 Funcionamento de um sistema modelado como MDP. 7
2.2 MDP decomposto em tarefas hierárquicas. 17
2.3 Mudanças de estado em um SMDP. 20

3.1 Funcionamento de um sistema modelado como POMDP. 28
3.2 Simplex definido por todos os possíveis estados de crença em um POMDP com quatro estados 29
3.3 Política para POMDP representada como conjunto de hiperplanos. 33
3.4 Política para POMDP representada como árvore e como plano recursivo. . 34
3.5 Política para POMDP representada como controlador de estados finitos. . . 34
3.6 Um passo de programação dinâmica do algoritmo de Sondik. 38
3.7 Construção da função valor usando o algoritmo de suporte linear de Cheng. 40
3.8 Remoção de um estado do controlador no algoritmo de Hansen. 44
3.9 Vetor dominado pela combinação convexa de outros dois vetores no algoritmo BPI. 46
3.10 Melhoria da política no algoritmo BPI. 48
3.11 Ótimo local no algoritmo BPI. 49
3.12 Representação de função valor no PBVI. 51
3.13 Limites superior e inferior da função valor no algoritmo HSVI. 54
3.14 Passo do algoritmo HSVI. 55

4.1 Autômato que restringe a ordem das ações em um MDP. 66
4.2 Função valor de um LL-MDP. 72
4.3 Autômato que limita as ações disponíveis no primeiro passo do MDP. . . . 82

5.1 Autômato $\mathcal{M}$ 89
5.2 POMDP resultante da multiplicação de estados. 89
5.3 POMDP resultante da multiplicação de estados (com probabilidades corretas). 90
5.4 Estados finais do autômato construído. 93
5.5 Autômato que causa operações desnecessárias de verificação de dominância. 96

5.6 Operações desnecessárias realizadas pelo LL-POMDP-DP. 97
5.7 Construindo uma política para LL-POMDPs com durações nas ações. . . . 100

6.1 Autômato que restringe o número de coletas antes do envio de dados por um robô. 105
6.2 Autômato para o problema do tigre, com três estados. 106
6.3 Autômato para o problema do Tigre, com oito estados. 106
6.4 O labirinto do exemplo de navegação de robô. 107
6.5 Autômato $\mathcal{M}$ usado no exemplo de navegação de robô. 108
6.6 O autômato $\mathcal{M}$ para o exemplo de diagnóstico e tratamento médico. 110
6.7 Autômato definindo número máximo de rochas coletadas pelo robô exploratório. 111

Lista de Algoritmos

2.1 Iteração de valores para MDPs. 10
2.2 Iteração de políticas para MDPs. 13
2.3 Avaliação da política no algoritmo modificado de iteração de políticas. . . . 14
2.4 Algoritmo RTDP para MDPs. 16
2.5 Algoritmo PolCa para decomposição de MDP em subtarefas. 18
2.6 Algoritmo PolCa para execução de política. 19
3.1 Iteração de valores para POMDPs com horizonte finito. 37
3.2 passo_dp - algoritmo de Sondik. 39
3.3 Construção da função valor Γ_i usando o algoritmo de suporte linear de Cheng. 41
3.4 passo_dp - incremental pruning. 42
3.5 Algoritmo de Hansen para POMDPs. 45
3.6 Inicialização dos estados de crença em uma grade. 50
3.7 Simulação de estados de crença para inicialização da grade. 51
3.8 Algoritmo RTDP para POMDPs. 52
3.9 Algoritmo PolCa+ para decomposição de POMDP em subtarefas. 56
3.10 Algoritmo PolCa+ para execução de política. 59
4.1 Execução de política para LL-MDP de horizonte finito. 68
4.2 Passo de programação dinâmica para LL-MDPs. 73
4.3 Iteração de políticas para LL-MDPs. 78
5.1 Execução de política para LL-POMDP de horizonte finito. 87
5.2 Passo de programação dinâmica para LL-POMDPs. 94
5.3 Algoritmo de execução de política para LL-POMDPs. 95
5.4 Usando uma política para LL-POMDPs com duração probabilística. 102

Lista de Lemas, Proposições e Teoremas

2.1 do ponto fixo de Banach 11
2.2 Erro de Bellman 12
2.3 Melhoria da política 12
2.4 Otimalidade da política 13
3.1 da forma das funções valor para POMDPs 32
4.1 Admissibilidade da linguagem do autômato gerado pelo LLVI 74
4.1 Admissibilidade da política gerada pelo LLVI 74
4.2 $\dot{\mathcal{V}}$ é um espaço de Banach 75
4.3 H para LL-MDPs é contração 75
4.2 Convergência do LLVI para LL-MDPs 76
4.3 Admissibilidade da política gerada pelo LLPI 79
4.4 Convergência do LLPI 79
4.5 Melhoria da política no LLPI 80
4.6 Corretude do critério de parada do LLPI 81
4.4 Otimalidade da política gerada pelo LLPI 81

Lista de Definições

2.1 MDP 5
2.2 Horizonte 6
2.3 Política para um MDP 6
2.4 Função valor de uma política para um MDP 8
2.5 Espaço de políticas para MDPs 8
2.6 Política ótima para um MDP 9
2.7 Espaço de Banach 11
2.8 Contração 11
2.9 SMDP 20
3.1 POMDP 26
3.2 Estado de informação 27
3.3 Política para um POMDP 31
3.4 Função valor de uma política para um POMDP 31
3.5 PWLC 32
4.1 Evento em um MDP 64
4.2 Autômato 65
4.3 Autômato reverso 65
4.4 Entradas e saídas de um estado 65
4.5 LL-MDP 66
4.6 MDP subjacente a um LL-MDP 66
4.7 Política para um LL-MDP 66
4.8 Função valor para um LL-MDP 67
4.9 Política admissível para um LL-MDP 68
4.10 Política ótima para um LL-MDP 69
4.11 Função valor ótima para um LL-MDP 69
4.12 Linguagem de um autômato a partir de um estado 73
5.1 LL-POMDP 85
5.2 POMDP subjacente a um LL-POMDP 85
5.3 Evento em um POMDP 85

5.4 Política para LL-POMDP 86
5.5 Função valor para um LL-POMDP 86
5.6 Política admissível para um LL-POMDP 86
5.7 Política ótima para um LL-POMDP 87
5.8 Função valor ótima para um LL-POMDP 87
5.9 LL-POMDP com durações 98

Notação

$\cdot$	Concatenação de linguagens
$\oplus$	Soma de todos os pares de vetores ($\Gamma_1 \oplus \Gamma_2 = \{\alpha_1 + \alpha_2 \| \alpha_1 \in \Gamma, \alpha_2 \in \Gamma_2\}$)
$\wedge$	"E" lógico
α	Um hiperplano representando o valor de um plano condicional
β	Operador de melhoria para limite inferior no HSVI
Γ	O conjunto de todos os vetores α formando uma função valor
γ	Fator de desconto
δ	Medida da melhoria no algoritmo BPI;
	Função de transição do autômato $\mathcal{M}$
Δ	Função de transição do autômato $\mathcal{W}$
π	Política para um MDP ou POMDP
π^*	Política ótima
ρ	Recompensa imediata para uma ação em um ponto de crença
	Um plano condicional para um POMDP
Σ	Alfabeto do autômato $\mathcal{M}$
τ	Estimador de estados de crença
Υ	Limite superior no algoritmo HSVI
Ω	Conjunto de observações
ω	Cluster de observações no algoritmo PolCa+
A	Conjunto de ações
a	Uma ação
$\mathcal{B}$	Espaço de estados de crença em um POMDP
B	Conjunto de pontos de crença
b	Um estado de crença

C	Critério de convergência
c	Cluster de estados no algoritmo PolCa
c_a	Probabilidade agregada dos planos que iniciam com a (BPI)
$c_{a,k_1,\cdots,k_{\lvert\Omega\rvert}}$	Probabilidade de plano escolher a e escolher k_i para cada o_i (BPI)
c_{a,k_o}	Probabilidade agregada para a, o, k (BPI)
D	Função de probabilidade de duração de evento
d_k	Regra de decisão na época k
e	Um evento
$\mathcal{F}$	Conjunto de estados finais do autômato $\mathcal{W}$
F	Conjunto de estados finais do autômato $\mathcal{M}$
H	Operador de melhoria para funções valor de MDPs e POMDPs
$\dot{H}$	Operador de melhoria para funções valor de LL-MDPs e LL-POMDPs
H_q	Operador de melhoria local para políticas de LL-MDPs e LL-POMDPs
$\dot{H}^{\pi'}$	Operador que modifica a primeira ação de uma política $\dot{\pi}$
h	Subtarefa no algoritmo PolCa;
	Função recompensa esperada para s, a ou b, a em MDPs ou POMDPs
$\dot{h}$	Função recompensa esperada para s, a ou b, a em LL-MDPs ou LL-POMDPs
$\mathcal{I}$	O espaço de estados de informação para um POMDP
I	Um estado de informação para um POMDP
	Conjunto de estados iniciais de um autômato
i	Uma iteração de programação dinâmica
K	Conjunto de estados de um controlador
k	Estado de um controlador;
	Uma época de decisão
$\mathcal{L}(A, q)$	Linguagem de um autômato A a partir de um estado q
L_b	Operador de melhoria para limite inferior no HSVI
$\mathcal{M}$	Autômato usado para impor restrições em um LL-MDP ou MM-POMDP
$\mathcal{M}^R$	Autômato reverso
$M^{a,o}$	Matriz $\lvert S\rvert \times \lvert S\rvert$, dando $p(s'\lvert s)$, para a, o fixos
$\overline{M}$	Maior recompensa possível em uma época de decisão
$\underline{M}$	Menor recompensa possível em uma época de decisão
$\mathbb{N}$	Conjunto dos números naturais

$OUT(q)$	Função saída do estado q de um autômato
O	Função de probabilidade para observações
o	Uma observação
$Pr(A\|B)$	Probabilidade de A, dado B
p	Estado de um autômato
$\mathcal{E}$	Conjunto de estados do autômato $\mathcal{W}$
E	Estados do autômato $\mathcal{M}$
Q	"Função Q": recompensa imediata para ação mais recompensa esperada
q	Estado de um autômato
q_0	Estado inicial do autômato $\mathcal{M}$
R	Função de recompensa
$\underline{R}$	Limite inferior para recompensa no HSVI
$\overline{R}$	Limite superior para recompensa no HSVI;
	Função local de recompensa no algoritmo PolCa
$\mathbb{R}$	Conjunto dos números reais
r_a	Vetor de recompensas para a: $r_a(s) = R(s, a)$
S	Conjunto de estados
T	Função de probabilidade de transição entre estados
U_b	Operador de melhoria para limite superior no HSVI
$\mathcal{V}$	Espaço de funções valor para um MDP ou POMDP
$\dot{\mathcal{V}}$	Espaço de funções valor para um LL-MDP ou LL-POMDP
$\dot{V}$	Função valor de um LL-MDP ou LL-POMDP (vetor de funções valor V_q)
V	Função valor de um MDP ou POMDP
v	Vetor contendo a função valor de um MDP
$\mathcal{W}$	Autômato gerado com a política de um LL-POMDP
z	Horizonte de um MDP ou POMDP

Capítulo 1

Introdução

A tomada de decisões em sequência é uma atividade comum para pessoas e de grande importância em Inteligência Artificial. O diagnóstico médico e tratamento de enfermidades é um exemplo: um médico deve escolher entre diversas ações (exames, tratamento, alta) sem ter certeza do estado atual do paciente. Mesmo após uma ação ter sido escolhida, não há também certeza de como ela influenciará o paciente (um tratamento pode resultar em cura em alguns casos, mas não em outros). O médico só recebe, após cada ação, algum sinal (o resultado de um exame, a melhoria – ou não – do paciente etc) [57, 38]. Outros exemplos são o sistema de navegação de um robô (que deve decidir para que direção ir, contando apenas com sensores imprecisos) [75]; sistemas de ajuda *online* (que devem tentar ajudar o usuário ao mesmo tempo em que tentam determinar o que realmente o usuário quer) [42]; uma empresa que deve decidir periodicamente a respeito de promoções e mudanças de preço. Até mesmo um diálogo envolve tomada de ações em sequência e incerteza [62]. Em todas estas situações, decisões precisam ser tomadas sem que haja certeza nem sobre o estado atual do mundo e nem sobre o resultado de cada ação.

Esta tese tem como objeto de estudo os processos de decisão de Markov, que são usados para modelar situações onde é necessário executar ações em sequência em ambientes com incerteza (tanto incerteza quanto ao resultado das ações executadas como incerteza quanto ao estado atual do ambiente). Uma situação que pode ser modelada como processo de decisão de Markov envolve um sistema com vários estados, ações que (possivelmente) modificam o estado do sistema, e a possibilidade de perceber (talvez indiretamente) o resultado de cada ação executada. Dada a descrição do problema, resolvê-lo significa encontrar uma *política ótima* que determine a cada momento que ação tomar para maximizar a esperança de recompensa.

Da maneira como são tradicionalmente definidos, os processos de decisão de Markov (MDPs) e processos de decisão de Markov Parcialmente Observáveis (POMDPs) não permitem a modelagem direta de situações onde as ações executadas devem respeitar

uma determinada ordem. A especificação da ordem para ações é importante, por exemplo, quando um dado procedimento médico não deve ser realizado antes de outro; quando um robô que visita salas entregando lanches deve contar quantos lanches já entregou; quando um monitor de equipamentos eletrônicos precisa saber quantas vezes já reiniciou cada equipamento, e determinar a troca após reiniciar o equipamento um certo número de vezes.

Uma limitação específica dos POMDPs é a impossibilidade de se especificar a duração de cada ação (ou do tempo que uma observação demora para ser coletada). Um tratamento médico pode demorar seis meses e um exame pode demorar uma semana; mover-se para a frente pode demorar alguns segundos, mas abrir uma porta pode demorar mais tempo para um robô. O planejamento de ações em sequência idealmente levaria em consideração o tempo de cada ação e cada observação, uma vez que em situações reais o horizonte disponível normalmente é uma quantidade de tempo (e não um número de decisões, os POMDPs normalmente permitem modelar).

As principais contribuições desta tese são:

1. A definição dos processos de Markov limitados por linguagem (*language-limited Markov decision processes*) completamente observáveis e parcialmente observáveis (LL-MDPs e LL-POMDPs) [58];

2. A extensão dos LL-POMDPs de forma que se possa especificar duração probabilística discreta para ações e observações;

3. A apresentação de algoritmos para a solução destes problemas. Isto implica em uma demonstração de que os LL-MDPs e LL-POMDPs não são mais expressivos do que MDPs e POMDPs, porque um dos métodos para a resolução de LL-MDPs e LL-POMDPs é a transformação em MDPs e POMDPs comuns (embora com o espaço de estados multiplicado);

4. A implementação dos algoritmos e estudo dos resultados.

A idéia central dos processos de decisão limitados por linguagem é permitir que, durante a modelagem do problema, seja possível restringir as ações que podem ser executadas em diferentes momentos, descrevendo o conjunto de possíveis sequências de ações como uma linguagem regular. Além da restrição na ordem das ações, pode-se usar observações como "seletores": certas observações, quando acontecerem, determinam qual o conjunto de ações possível a partir daquele momento. Por exemplo, se um robô percebe que um sensor quebrou, ele deve deixar de usá-lo (e aquela ação deve ser desconsiderada a partir daquele momento). A separação entre processo de Markov e a definição da linguagem que

define a ordem das ações e observações facilita o processo de modelagem ao tratar os dois conceitos separadamente.

O texto está organizado da seguinte forma: nos Capítulos 2 e 3 os processos de Markov são formalmente definidos e algoritmos exatos e heurísticas são apresentados. Nos Capítulos 4 e 5 são definidos os processos de Markov limitados por linguagem e algoritmos são apresentados para resolvê-los. O Capítulo 6 traz a descrição de uma série de problemas modelados como LL-MDPs e LL-POMDPs. Todos os problemas foram resolvidos usando diversas abordagens diferentes, e a representação dos problemas e os tempos de execução são comparados. O Capítulo 7 mostra um número de trabalhos futuros relacionados ao tema da tese.

Capítulo 2

Processos de Decisão de Markov

Um processo de decisão de Markov (MDP - *Markov Decision Process*) é uma forma de modelar processos onde as transições entre estados são probabilísticas, é possível observar em que estado o processo está, e é possível interferir no processo periodicamente (em "épocas de decisão") executando ações [69, 14, 36]. Cada ação tem uma recompensa (ou custo), que depende do estado em que o processo se encontra. Alternativamente, pode-se definir recompensas por estado apenas, sem que estas dependam da ação executada. São ditos "de Markov" (ou "Markovianos") porque os processos modelados obedecem a *propriedade de Markov*: o efeito de uma ação em um estado depende apenas da ação e do estado atual do sistema (e não de como o processo chegou a tal estado); e são chamados de processos "de decisão" porque modelam a possibilidade de um agente (ou "tomador de decisões") interferir periodicamente no sistema executando ações, diferentemente de Cadeias de Markov, onde não se trata de como interferir no processo. MDPs podem ser aplicados em um grande número de áreas diferentes, como mostra White [84]. O exemplo a seguir é descrito por Puterman [69]:

Toda semana, um revendedor de carros faz um pedido para a fábrica, dizendo quantos carros ele quer; cada carro tem um custo c. A fábrica manda os carros rapidamente, de forma que podemos considerar que a entrega é imediata, e que os novos carros "aparecem" imediatamente no estoque. Então, durante a semana, ele vende k carros (k é aleatório), a um preço p cada. O revendedor tem um custo fixo para manter os carros no estoque (u por carro).

Este é um sistema que pode estar em vários estados (onde cada estado é um possível número de carros no estoque); as ações são de compra: o revendedor pode comprar diferentes quantidades de carros, e para cada quantidade temos uma ação; o revendedor não tem controle sobre quantos carros ele vende (este é um evento estocástico). Esta situação pode ser modelada como um processo de decisão de Markov.

Definição 2.1 (MDP)
Um processo de decisão de Markov (MDP) é uma tupla (S, A, T, R), onde:

- *S é um conjunto de estados em que o processo pode estar;*
- *A é um conjunto de ações que podem ser executadas em diferentes épocas de decisão;*
- *$T : S \times A \times S \mapsto [0, 1]$ é uma função que dá a probabilidade de o sistema passar para um estado $s' \in S$, dado que o processo estava em um estado $s \in S$ e o agente decidiu executar uma ação $a \in A$ (denotada $T(s'|s, a)$);*
- *$R : S \times A \mapsto \mathbb{R}$ é uma função que dá o custo (ou recompensa) por tomar uma decisão $a \in A$ quando o processo está em um estado $s \in S$.*

É também comum definir, para cada estado $s \in S$, um conjunto de ações possíveis naquele estado (A_s). Assim, o conjunto de todas as ações poderia ser $A = \cup_{s \in S} A_s$. Este trabalho usa apenas "A", a fim de simplificar a notação, exceto quando a distinção entre A e A_s for relevante.

Definição 2.2 (Horizonte)
O horizonte de um MDP, neste trabalho denotado por z, é o número de épocas de decisão *disponíveis para a tomada de decisões.*

O horizonte pode ser *finito* (quando há um número fixo de decisões a tomar), *infinito* (quando a tomada de decisão é feita repetidamente) ou *indefinido* (semelhante ao horizonte infinito, mas com a possibilidade do processo parar se chegar a algum estado que tenha sido marcado como final[1]).

2.1 Políticas

A Figura 2.1 mostra a dinâmica de funcionamento de um sistema modelado como processo de decisão de Markov. O tomador de decisões verifica o estado atual do sistema (s), consulta uma política (π) e executa uma ação (a). Esta ação pode ter um efeito sobre o ambiente, e modificar o estado atual. O tomador de decisões, então, verifica o novo estado para que possa tomar a próxima decisão.

Definição 2.3 (Política para um MDP)
Uma regra de decisão para um MDP em uma época de decisão k é uma função $d_k : S \mapsto A$, que determina a ação a ser executada, dado o estado do sistema.

Uma política para um MDP é uma sequência de regras de decisão $\pi = \{d_0, d_1, \cdots, d_{z-1}\}$, uma para cada época de decisão.

[1] É possível também definir ações finais.

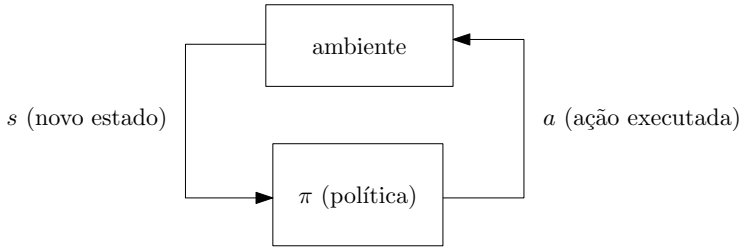


Figura 2.1: Funcionamento de um sistema modelado como MDP.

Normalmente se quer encontrar uma política que otimize um dado critério de desempenho das decisões. Uma política pode ser *determinística*, se recomenda uma ação para cada estado, ou *estocástica*, se a ação recomendada para um estado obedece a uma distribuição de probabilidades. Além disso, uma política pode ser:

- *Total*, se cada uma de suas regras de decisão é definida para todos os estados do MDP;
- *Parcial*, se alguma de suas regras de decisão é definida para apenas alguns estados do MDP.

Quanto à sua relação com as épocas de decisão, uma política pode ainda ser classificada como:

- *Estacionária*, se a ação recomendada independe da época de decisão (ou seja, $d_k = d_j$ para todo k e j). Políticas estacionárias podem ser usadas com horizonte finito ou infinito.
- *Não-estacionária*, se a ação tomada depende da época de decisão. Políticas não-estacionárias são usadas apenas com horizonte finito, uma vez que para horizonte infinito haveriam infinitas regras de decisão.

Quando uma política for estacionária, $\pi(s)$ pode ser usado como sinônimo de $d_k(s)$.

Ao executar uma política, o tomador de decisões receberá recompensas em cada época de decisão. Para comparar duas políticas, é necessário um critério de desempenho (de outra forma a comparação não é possível). Há diversos critérios de desempenho (ou "de otimalidade") que podem ser usados com MDPs, e entre os mais comuns podemos citar:

- A *recompensa média* por época de decisão, $\frac{1}{z}\sum_{k=0}^{z-1} r_k$;
- A *esperança da recompensa total*, $E\left[\sum_{k=0}^{z-1} r_k\right]$;
- A *esperança da recompensa descontada*, $E\left[\sum_{k=0}^{z-1} \gamma^k r_k\right]$.

A recompensa na época de decisão k é denotada por r_k, e $\gamma \in]0,1[$ é um fator de desconto. O fator de desconto é usado com horizonte infinito para garantir a convergência do valor da recompensa total esperada. Quando o horizonte é infinito, usa-se o limite no infinito sobre o critério de otimalidade (por exemplo, $\lim_{z\to\infty} \frac{1}{z}\sum_{k=0}^{z-1} r_k$ para recompensa média). Este trabalho se restringe ao critério da esperança da recompensa total descontada.

É importante também o conceito de *valor de uma política*, dado por uma *função valor*.

Definição 2.4 (Função valor de uma política para um MDP)
A função valor de uma política π para um MDP $M = (S, A, T, R)$ é uma função $V^\pi : S \mapsto \mathbb{R}$, tal que $V^\pi(s)$ dá o valor esperado da recompensa para esta política, de acordo com o critério de otimalidade.

Por exemplo, para horizonte finito com desconto seja $\pi = \{d_0, d_1, \cdots, d_{z-1}\}$,

$$V_k^\pi(s) = R(s, d_k(s)) + \gamma \sum_{s' \in S} T(s'|s, d_k(s)) V_{k+1}^\pi(s')$$

onde $V_z^\pi(s) = 0$ para todo s (porque após a última época de decisão não há mais ações que possam gerar recompensas).

Quando não houver ambiguidade, usaremos V ao invés de V^π.

Definição 2.5 (Espaço de políticas para MDPs)
Dado um MDP $M = (S, A, T, R)$, $\Pi(M)$ é o conjunto de todas as políticas para M e $\mathcal{V}(M)$ o conjunto de todas as funções valor $V : S \mapsto \mathbb{R}$ para M.

Se cada função valor for representada por um vetor (um elemento para o valor de cada estado) e as operações de soma e multiplicação por escalar forem definidas da mesma forma que normalmente se faz para $\mathbb{R}^n$, o espaço $\mathcal{V}$ é um espaço linear.

Neste trabalho, quando não houver ambiguidade (ou seja, quando houver um único MDP), Π e $\mathcal{V}$ serão usados no lugar de $\Pi(M)$ e $\mathcal{V}(M)$.

Dados um estado $s \in S$, uma ação $a \in A$ e uma política π para um MDP, pode-se definir o valor da ação a no estado s, considerando a recompensa imediata de a e a recompensa esperada após a, nas outras épocas de decisão, desde que as ações tomadas após a sejam determinadas pela política π. A função que dá este valor é denotada por Q. Para a esperança da recompensa total descontada, Q é definida como

$$Q^\pi(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s'|s, a) V^\pi(s'). \tag{2.1}$$

Para horizonte finito,

$$Q_k^\pi(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s'|s, a) V_{k+1}^\pi(s')$$

Definição 2.6 (Política ótima para um MDP)
Seja $M = (S, A, T, R)$ um MDP com horizonte z.

Uma função valor V^* é ótima para M se $\forall U \in \mathcal{V}, \forall s \in S,\ V_k(s) \geq U_k(s)$ para todo $k \in \mathbb{N}$ tal que $0 \leq k < z$.

Uma política π^* é ótima para M se $\forall \pi' \in \Pi, \forall s \in S,\ Q_k^{\pi^*}(s, d_k^{\pi^*}(s)) \geq Q_k^{\pi'}(s, d_k^{\pi'}(s))$ para todo $k \in \mathbb{N}$ tal que $0 \leq k < z$.

A definição de política ótima vale para horizonte infinito, bastando ignorar os índices de época de decisão.

Se π^* é uma política ótima, a notação pode ser simplificada usando Q^* ao invés de Q^{π^*}:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) V^*(s'),$$

$$\pi^*(s) = \arg\max_{a \in A} Q^*(s,a),$$

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

2.2 Algoritmos

Existe uma grande quantidade de algoritmos para a solução de MDPs (o problema é P-completo [55]). Apenas alguns são mostrados aqui. Alguns dos algoritmos trabalham diretamente com políticas, enquanto outros trabalham com funções valor.

2.2.1 Iteração de Valores

O algoritmo de iteração de valores usa programação dinâmica para determinar o valor $V^*(s)$ de cada estado $s \in S$ do MDP, em cada época de decisão. O valor de cada estado na última época de decisão é $V_{z-1}^*(s) = \max_{a \in A} R(s,a)$, uma vez que quando só há uma decisão a ser tomada, basta escolher aquela com a maior recompensa.

Um mapeamento $h : S \times A \times \mathcal{V} \mapsto \mathbb{R}$ é definido, tal que $h(s, a, V)$ dá o valor de executar a ação a no estado s e seguir uma dada política (derivada de uma função valor V) após esta ação:

$$h(s,a,V) = R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) V(s').$$

Define-se um operador $H : \mathcal{V} \mapsto \mathcal{V}$ de melhoria da política, que determina a melhor ação em um estado usando os valores V dos estados em uma época de decisão anterior, de tal forma que

$$HV_k(s) = \max_{a \in A} h(s, a, V_{k+1}).$$

Usando o operador H, a equação de otimalidade para MDPs determina uma função valor em uma época de decisão em função da função valor da época de decisão anterior:

$$\begin{aligned} V_k(s) &= HV_{k+1}(s) \\ &= \max_{a \in A} h(s, a, V_{k+1}) \\ &= \max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) V_{k+1}(s') \right]. \end{aligned} \tag{2.2}$$

A Equação 2.2 pode ser usada para implementar o algoritmo de *iteração de valores* (Algoritmo 2.1), que usa programação dinâmica para determinar a política ótima para um MDP.

Note que o valor máximo nas linhas 2 e 7 é obtido de A_s, e não de A, uma vez que $A_s \leq A$. Ainda que a descrição do MDP não inclua os conjuntos A_s explicitamente, é possível usar $A_s = \{a \in A \mid \exists s' \in S,\ T(s'|s,a) > 0\}$.

Entrada: Um MDP (S, A, T, R).
Saída: Uma função valor V para o MDP dado como entrada.

```
1  para cada s ∈ S faça
2  |  V_0(s) ← max_{a∈A_s} R(s,a)
3  fim
4  i ← 1
5  enquanto critério de parada não satisfeito faça
6  |  para cada s ∈ S faça
7  |  |  V_i(s) ← max_{a∈A_s} [R(s,a) + γ Σ_{s'∈S} T(s'|s,a) V_{i-1}(s')]
8  |  fim
9  |  i ← i + 1
10 fim
11 Devolva V
```

Algoritmo 2.1: Iteração de valores para MDPs.

No algoritmo de iteração de valores, i já não é a época de decisão, mas o passo de programação dinâmica (por isso o $k+1$ da equação de otimalidade foi trocado por $i-1$).

O algoritmo de iteração de valores pode ser usado para resolver MDPs com horizonte infinito, porque converge para a solução ótima (a prova está na subseção 2.2.1). A equação

de otimalidade para horizonte infinito é

$$\begin{aligned} V(s) &= HV(s) \\ &= \max_{a \in A} h(s, a, V) \\ &= \max_{a \in A} \left[R(s, a) + \gamma \sum_{s' \in S} T(s'|s, a) V(s') \right]. \end{aligned}$$

Convergência

O critério de parada no caso de horizonte finito é que o número de iterações seja igual ao horizonte do problema. Neste caso, o algoritmo gera uma política para cada época de decisão (a política é não-estacionária). A complexidade assintótica do algoritmo é $\mathcal{O}(z|A||S|^2)$.

Para processos com horizonte infinito (ou indefinido), o algoritmo para quando os valores para cada estado tiverem convergido.

Um conceito importante na prova de convergência dos algoritmos de iteração de valores é o de uma *contração em um espaço de Banach.*

Definição 2.7 (Espaço de Banach)
Um espaço vetorial X com uma norma $||\cdot||$ é um espaço de Banach se toda sequência de Cauchy $(x_0, x_1, \cdots, x_n)$ de elementos de X tem um limite $x_k \in X$.

Definição 2.8 (Contração)
Seja X um espaço de Banach. Um operador $F : X \mapsto X$ é uma contração quando para quaisquer dois elementos $U, V \in X$:

$$||FV - FU|| \leq \beta ||V - U||$$

onde $0 \leq \beta \leq 1$ é o fator de contração.

Teorema 2.1 (do ponto fixo de Banach)
Seja X um espaço de Banach. Seja $F : X \mapsto X$ uma contração e seja $N = (x_0, x_1, \cdots, x_k)$ uma sequência com um ponto inicial arbitrário $x_0 \in X$, tal que $x_i = Fx_{i-1}$. Então:

- *F tem um único ponto fixo x^* tal que $Fx^* = x^*$;*
- *A sequência N converge para x^*.*

Seja $||\cdot||$ uma norma no espaço $\mathcal{V}$ tal que $||V|| = \max_{s \in S} |V(s)|$. $\mathcal{V}$ é claramente um espaço de Banach. Além disso, o operador H é uma contração em $\mathcal{V}$ (a prova pode ser encontrada no livro de Puterman [69]). Assim, o teorema de Banach garante que H tem

um ponto fixo único V^*, e que a sequência $V_k = HV_{k-1}$, iniciando de uma função valor qualquer, converge para este ponto fixo.

A garantia de convergência não é suficiente para que se obtenha do algoritmo uma função valor precisa. É necessário determinar também quando parar de calcular aproximações sucessivas da função valor. Um critério de convergência muito usado usa o Teorema 2.2, cuja prova é dada também por Puterman [69].

Teorema 2.2 (Erro de Bellman)
Se a diferença entre $V_k(s)$ e $V_{k-1}(s)$ é no máximo δ para todo estado s, então V_k nunca difere de V^* por mais de $\frac{\delta}{1-\gamma}$, para qualquer estado.

Assim, o critério de parada deve ser

$$\forall s \in S, \ \ |V(s) - V'(s)| \leq \frac{\epsilon(1-\gamma)}{2\gamma} \tag{2.3}$$

para que a precisão da solução seja no mínimo ϵ.

2.2.2 Iteração de Políticas

No caso de horizonte infinito (e política estacionária), pode-se também usar um algoritmo chamado de *iteração de políticas*, que faz uma busca gulosa no espaço de políticas.

O algoritmo começa com uma política aleatória, determina o valor da política atual e depois, de maneira gulosa, melhora a política buscando modificar as ações recomendadas para cada estado.

Puterman [69] atribui a idéia de iteração de políticas a Howard [41] e Bellman [11, 9]. O algoritmo de iteração de políticas se baseia no seguintes Teoremas 2.3 e 2.4 de Bellman e Dreyfus [10]:

Teorema 2.3 (Melhoria da política)
Sejam duas políticas estacionárias, π e π', tais que

$$\forall s \in S, \ V^{\pi}(s) \leq Q^{\pi}(s, \pi'(s))$$

então π' é uniformemente melhor que π, ou seja:

$$\forall s \in S, \ V^{\pi}(s) \leq V^{\pi'}(s).$$

O teorema pode ser entendido da seguinte maneira. Dadas duas políticas (π e π'), temos para cada estado:

- $V^{\pi}(s)$ (o valor esperado de π para este estado);

- $Q^{\pi}(s, \pi'(s))$, que é o valor esperado para s usando a política π, *exceto que a primeira ação é dada por* π'.

O teorema diz que π' é uniformemente melhor que π. Isto pode ser usado para melhorar uma política que não seja ótima.

Teorema 2.4 (Otimalidade da política)
Seja uma política π *para um MDP* M *e uma função valor* V^{π} *associada a* π. *Se* π *não puder ser melhorada usando o teorema 2.3, ou seja, se*

$$\forall s \in S \;\; V^{\pi}(s) = \max_{a \in A} Q^{\pi}(s, a)$$

então π *é ótima para* M.

Para cada estado s, o algoritmo de iteração de políticas determina qual a melhor ação a tomar (ou seja, determina uma nova política π, dado que as ações seguintes serão tomadas de acordo com π' (a política definida no passo anterior):

$$\forall s \in S, \;\; \pi(s) = \arg\max_{a \in A} Q^{\pi'}(s, a).$$

Como a política é estacionária, π aparece na equação como uma função de estados em ações.

Entrada: Um MDP (S, A, T, R).
Saída: Uma política π^*, ótima para o MDP dado como entrada.

```
Inicialize π aleatoriamente
repita
    π' ← π
    Avalie a política atual resolvendo o sistema linear:
    V(s) = R(s, π'(s)) + γ Σ_{s'∈S} T(s'|s, π'(s)) V(s')
    para cada s ∈ S faça
        ∀a ∈ A,  Q^{π'}(s, a) ← R(s, a) + γ Σ_{s'∈S} T(s'|s, a) V(s')
    fim
    para cada s ∈ S faça
        Melhore a política:
        π(s) ← arg max_{a∈A_s} Q^{π'}(s, a)
    fim
até que π = π'
Devolva π
```

Algoritmo 2.2: Iteração de políticas para MDPs.

O Algoritmo 2.2 mostra uma versão detalhada da iteração de políticas. V^{π} é o valor esperado, usando a política π. Da mesma forma que no algoritmo de iteração de valores, A_s é usado ao invés de A. No resto deste trabalho apenas A, ficando subentendido que A_s deve ser usado sempre que possível.

Algoritmo modificado de Iteração de Políticas

Puterman [69] descreve uma versão modificada do algoritmo de iteração de políticas que converge com menos iterações do que o Algoritmo 2.2. O algoritmo modificado não computa a função valor V exata a cada passo, mas usa uma aproximação (que é suficiente para escolher a melhor ação no passo de melhoria). A solução do sistema linear é trocada por uma variante de iteração de valores onde a política é fixa (detalhada no Algoritmo 2.3).

```
1 repita
2 |  para cada s ∈ S faça
3 |  |  V(s) ← R(s, π(s)) + γ Σ_{s'∈S} T(s'|s, π(s))V(s')
4 |  fim
5 até Até que a maior modificação em um valor seja menor que algum ε
```

Algoritmo 2.3: Avaliação da política no algoritmo modificado de iteração de políticas.

2.2.3 Programação Linear

O problema de encontrar a política ótima para um MDP de horizonte infinito pode ser formulado como um problema de programação linear. A função objetivo é:

$$\text{minimizar: } \sum_{s \in S} v_s,$$

e para cada par (s, a), uma restrição é adicionada:

$$v_s \geq R(s, a) + \gamma \sum_{s' \in S} T(s'|s, a) v_{s'}.$$

As variáveis v_s no vetor v representam o valor associado a cada estado (ou seja, $v_s = V(s)$).

As restrições significam que o valor da política ótima para um estado não pode ser menor do que a recompensa imediata, mais a recompensa esperada para os próximos passos. Com a minimização da soma dos valores v_i para todos os estados, os valores da solução ótima no ponto fixo são encontrados.

Por exemplo, considere o seguinte MDP:

$$S = \{s_0, s_1\},$$
$$A = \{a_0, a_1, a_2\}.$$

A Tabela 2.1 mostra a função de transição, e a Tabela 2.2 mostra as recompensas.

	a_0	a_1	a_2
s_0	$(s_0, 0.3); (s_1, 0.7)$	$(s_0, 0.2); (s_1, 0.8)$	$(s_0, 0.6); (s_1, 0.4)$
s_1	$(s_0, 0.5); (s_1, 0.5)$	$(s_0, 0.6); (s_1, 0.4)$	$(s_0, 0.7); (s_1, 0.3)$

Tabela 2.1: Probabilidades de transição em um MDP.

	a_0	a_1	a_2
s_0	10	1	30
s_1	50	20	2

Tabela 2.2: Recompensas em um MDP.

A formulação do programa linear que determina a política ótima para este MDP é:

$$\text{minimizar: } v_0 + v_1$$

sujeito a:

$$\begin{aligned}
v_0 &\geq 10 + \gamma 0.3 v_0 + \gamma 0.7 v_1, \\
v_0 &\geq 1 + \gamma 0.2 v_0 + \gamma 0.8 v_1, \\
v_0 &\geq 30 + \gamma 0.6 v_0 + \gamma 0.4 v_1, \\
v_1 &\geq 50 + \gamma 0.5 v_0 + \gamma 0.5 v_1, \\
v_1 &\geq 20 + \gamma 0.6 v_0 + \gamma 0.4 v_1, \\
v_1 &\geq 2 + \gamma 0.7 v_0 + \gamma 0.3 v_1.
\end{aligned}$$

O programa linear tem $|S|$ variáveis e $|S||A|$ restrições. A solução deste programa linear é a política ótima para o MDP descrito.

2.2.4 RTDP

O algoritmo de iteração de valores determina uma função valor para um MDP através de aproximações sucessivas da equação de otimalidade, calculando $V(s)$ para todos os

estados a cada iteração do algoritmo. Barto, Bradtke e Singh propuseram um algoritmo chamado "*real time dynamic programming*", ou RTDP, que determina uma função valor para o MDP através de uma simulação [8]. O RTDP escolhe ações e percorre estados, atualizando o valor de cada estado visitado usando a equação de otimalidade.

O Algoritmo 2.4 mostra o RTDP. Para que haja garantia de convergência, todos os estados devem ser visitados periodicamente na simulação. Uma maneira de garantir isto é dividir a simulação em rodadas, sendo que cada uma inicia com um dos estados do MDP. Cada rodada da simulação pode terminar após um número de passos (ou até algum critério de convergência ser satisfeito). A simulação executa todas as rodadas em sequência, várias vezes, e termina quando um critério de convergência global for satisfeito (por exemplo, quando a diferença entre os valores de cada estado antes e depois de todas as rodadas for menor que algum ϵ).

Entrada: Um MDP (S, A, T, R);
Uma função valor V para o MDP;
Um estado inicial s_0.
Saída: Uma função valor V para o MDP dado como entrada.

$s \leftarrow s_0$
enquanto *o critério de final da rodada não for satisfeito* **faça**
  Avalie o valor de cada ação no estado atual:
  $Q(s,a) \leftarrow R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a)V(s')$
  $a^* \leftarrow \arg\max\ Q(s,a)$
  $V(s) \leftarrow Q(s,a^*)$
  Simule a execução da ação a^* no estado s e observe o estado resultante s'
  $s \leftarrow s'$
fim
Devolva V

Algoritmo 2.4: Algoritmo RTDP para MDPs.

Há outros algoritmos derivados do RTDP, como o LRTDP [15], o BRTDP [49] e o FRTDP [76].

2.2.5 Decomposição hierárquica

Um MDP pode ser decomposto em partes, de forma a permitir que estas partes sejam reusadas e também acelerar a busca pela solução ótima. Há diversos algoritmos para a decomposição de MDPs [25, 43, 24, 23]. O método de fatoração desenvolvido por Pineau e Thrun [63, 60] é particularmente interessante: parte das ações é composta de subtarefas, e cada subtarefa é recursivamente definida como um MDP. O algoritmo para decomposição

hierárquica é chamado de PolCa.

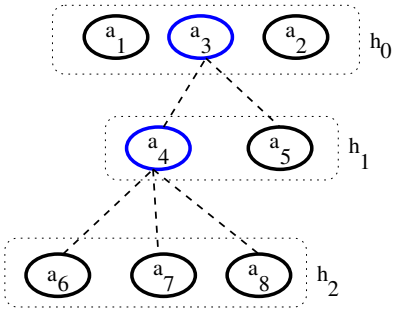


Figura 2.2: MDP decomposto em tarefas hierárquicas.

O algoritmo PolCa divide as ações do MDP original de forma a agrupar algumas delas em "subtarefas". A Figura 2.2 mostra as ações de um MDP, decompostas em subtarefas. Algumas ações são primitivas (pertencem ao conjunto A de ações do MDP original), e outras são abstratas (definidas para representar uma subtarefa). O MDP de nível zero (h_0) tem três ações, sendo uma delas (a_3) abstrata. No nível h_1 há um MDP com duas ações, sendo a_4 abstrata. Finalmente, no último nível da hierarquia de tarefas (h_2), há um MDP com três ações, todas primitivas. Durante a execução da política, o tomador de decisões deverá, a cada decisão, determinar uma subtarefa e então usar a política daquela subtarefa para decidir que ação primitiva executar.

Cada subtarefa hierárquica h é formada por:

- S_h: um conjunto de clusters de estados. Os estados do MDP original são agrupados em clusters;
- A_h: ações (tanto as primitivas como as abstratas);
- $\overline{R}_h : S_h \times A_h \mapsto \mathbb{R}$: uma função local de recompensa.

Depois da decomposição do MDP em subtarefas, os parâmetros das subtarefas são definidos pelo Algoritmo 2.5.

Os passos do algoritmo são detalhados a seguir.

Reformulação do espaço de estados Além do estado atual, uma nova variável "subtarefa atual" é incluída. Estados passam a ser então pares $(s, h) \in S \times H$, onde H é o conjunto de todas as tarefas (inclusive o MDP raiz). Para simplificar a notação, usamos S_h para $\{(s, h)|s \in S\}$.

```
1 Reformule o espaço de estados H · S
2 para cada tarefa h ∈ H faça
3 |   Defina parâmetros T e S
4 |   Minimize estados
5 |   Determine solução para h
6 fim
```

Algoritmo 2.5: Algoritmo PolCa para decomposição de MDP em subtarefas.

Definição de T e R A função de de probabilidades de transição é definida para as subtarefas. Para cada ação a:

Se a é uma ação primitiva,

$$T(h \cdot s'|h \cdot s, a_k) = T(s'|s, a_k),$$
$$R(h \cdot s, a_k) = \overline{R}(s, a_k).$$

Se $\overline{a}_p$ é uma ação abstrata que contém a subtarefa h_p, então

$$T(h \cdot s'|h \cdot s, \overline{a}_p) = T(s'|s, \pi^*_{h_p}(s)),$$
$$R(h \cdot s, \overline{a}_p) = \overline{R}(s, \pi^*_{h_p}(s)),$$

onde $\pi^*_{h_p}(s)$ é a política ótima para a subtarefa h_p no estado s.

Minimização de estados Os estados são divididos em clusters: Seja $z_h(s_i) = z_h(s_j)$ se

$$R(h \cdot s_i, a) = R(h \cdot s_j, a),\ \forall a \in A_h. \tag{2.4}$$

Em seguida, verifica-se a estabilidade de cada cluster c. Para todos os pares de estados originais s_i e s_j do cluster c, e cada ação de A_h, a soma das probabilidades de o sistema sair de s_i com a para outro cluster deve ser a mesma se o sistema sair de s_j. Ou seja, $c \in S_h$ é estável se e somente se

$$\forall (s_i, s_j) \in c,\ \forall c' \in S_h, \forall a \in A_h,$$
$$\sum_{s' \in c'} T(h \cdot s'|h \cdot s_i, a) = \sum_{s' \in c'} T(h \cdot s'|h \cdot s_j, a). \tag{2.5}$$

Se algum cluster não é estável, ele é dividido ($c \rightarrow \{c_k, c_{k+1}, \ldots\}$), de forma a satisfazer a equação acima.

Solução das subtarefas Como a passo de minimização de estados garante que todos os estados em cada cluster tem o mesmo valor, o cálculo da função valor é incluído no passo de minimização. Assim, após determinar R usando a Equação 2.4, verificar se o cluster é estável usando a Equação 2.5 e dividir o cluster, o valor de cada clusters é calculado:

$$\forall c \in C_h, \qquad V(c) = \max_a R(c,a) + \gamma \sum_{c' \in C_h} T(c'|c,a)V(c')$$

$$\forall c \in C_h, \qquad \pi(c) = \arg\max_a R(c,a) + \gamma \sum_{c' \in C_h} T(c'|c,a)V(c')$$

Estes passos se repetem até que V tenha convergido.

Execução da política hierárquica

Cada um dos MDPs relativos às subtarefas tem uma política ótima, que é definida sobre clusters (e não sobre estados do MDP original): a política π_h é definida para clusters da subtarefa h (como se fossem estados). O Algoritmo 2.6 mostra como a política hierárquica é usada para determinar a próxima ação.

O algoritmo determina a subtarefa corrente a cada nova iteração, não ficando "preso" em uma subtarefa entre iterações. O laço mostrado no algoritmo aprofunda-se na hierarquia até encontrar uma política local que recomende uma ação primitiva.

Entrada: h_k, a tarefa atual no tempo k, e b_k, o estado de crença global no tempo k.
Saída: a_k, a ação a ser executada no tempo k.

1 $a_k = \pi_h(b_k^h)$
2 **enquanto** a_k *é abstrata (ou seja,* $\overline{a}_k$*)* **faça**
3 Seja h_{sub} a tarefa indicada por $\overline{a}_k$
4 $b_k^h(c) \leftarrow \sum_{s \in c} b_k(s), \forall c \in S_h$
5 $a_k \leftarrow \pi_h(b_k^h)$
6 **fim**
7 Retorne a_k

Algoritmo 2.6: Algoritmo PolCa para execução de política.

2.2.6 Outros métodos

Uma exposição extensa sobre MDPs é dada por Puterman [69] e por Bertsekas [14], que dão detalhes de outros algoritmos. Hauskrecht [39] também mostra diversos algoritmos

para MDPs. Os métodos descritos neste capítulo são os métodos clássicos; há uma grande quantidade de métodos e algoritmos melhorados [81, 51, 85, 31] para a solução de MDPs que este trabalho não discute.

2.3 Processos de Decisão Semi-Markovianos

Esta seção apresenta os processos de decisão semi-Markovianos. Estes são semelhantes aos processos de Markov já apresentados, mas incluem a noção de tempo contínuo, definindo duração (possivelmente probabilística) para ações, e separando o *processo de decisão* (que pode ainda ser visto com um processo em tempo discreto) do *processo natural* (que acontece sem intervenção, em tempo contínuo, como se houvesse uma cadeia de Markov entre quaisquer duas épocas de decisão). Estes processos são ditos "semi-Markovianos" porque a rigor, o estado do sistema não depende apenas do último estado e ação: depende também do desenvolvimento do processo natural, desde a última época de decisão até o momento corrente. Em SMDPs, leva-se em conta o custo decorrente de o processo permanecer em um dado estado entre as diferentes épocas de decisão.

A Figura 2.3 mostra as mudanças de estados em um SMDP: as épocas de decisão acontecem em intervalos não regulares de tempo, e o estado do sistema pode mudar entre uma época de decisão e outra (as linhas curvas mostram as mudanças "espontâneas", modeladas como processo natural, e as linhas pontilhadas retas mostram o momento em que as mudanças são causadas por decisões tomadas no processo de Markov).

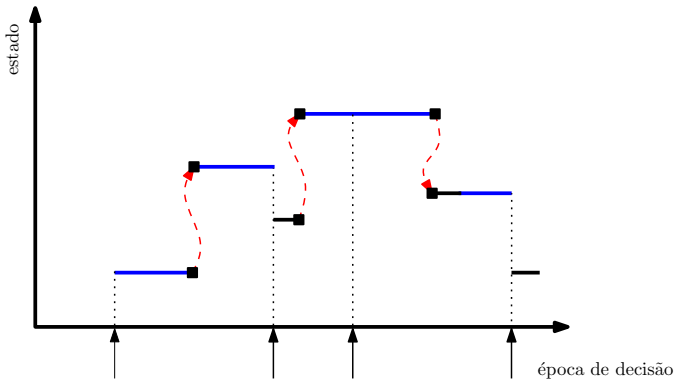


Figura 2.3: Mudanças de estado em um SMDP.

Neste trabalho, usaremos t e u para denotar tempo contínuo e k para denotar tempo discreto. Assim, t_k é o tempo da k-ésima época de decisão.

Definição 2.9 (SMDP)
Um processo de decisão semi-Markoviano (SMDP) é uma tupla (S, A, T, R, F)*, onde:*

- S *é um conjunto de estados;*
- A *é um conjunto de ações;*
- $T : S \times A \times S \mapsto [0, 1]$ *é a função de probabilidade de transições.* $T(s'|s, a)$ *dá a probabilidade de o estado atual ser* s'*, dado que na última época de decisão o estado era* s *e a ação* a *foi executada;*
- $R : S \times A \mapsto \mathbb{R}$ *é uma função que dá uma recompensa contabilizada em cada época de decisão;*
- $F : \mathbb{R} \times S \times A \mapsto [0, 1]$ *é uma função que dá a probabilidade de a próxima época de decisão acontecer antes do tempo* t*, sendo que o processo está em uma época de decisão, no estado* s*, e a ação* a *passará a ser executada. Usaremos a notação "*$F(t|s, a)$*". Para garantir que não haja infinitas épocas de decisão em um espaço finito de tempo, é necessário que* $\exists \varepsilon > 0, \delta > 0$ *tal que* $F(s, a, \delta) \leq 1 - \varepsilon$ *para todo estado* s *e ação* a*.*

A função R é composta de:

- $r : S \times A \mapsto \mathbb{R}$, uma recompensa imediata para a decisão de executar uma ação a estando em um estado s (equivalente a R para MDPs);
- $\mathcal{R} : S \times S \times A \mapsto \mathbb{R}$, uma função que dá a recompensa ou custo por permanecer em um estado s' depois de ter estado no estado s e decidido executar uma ação a.

O custo entre duas épocas de decisão k e $k + 1$, que se encontram nos tempos u_k e u_{k+1} (sem contabilizar a recompensa imediata) é dado por

$$\int_{u_k}^{u_{k+1}} e^{-\alpha(t-u_k)} \mathcal{R}(s, s_t, a) dt$$

onde s_t é o estado no tempo t.

A recompensa total descontada é dada por

$$V(s) = E\left\{ \sum_{k=0}^{\infty} e^{-\alpha u_k} \left[r(s, a) + \int_{u_k}^{u_{k+1}} e^{-\alpha(t-u_k)} \mathcal{R}(s, s_t, a) dt \right] \right\}$$

onde E é a esperança relativa a $F(s, a, u)$ no tempo entre u_k e u_{k+1}. O termo $e^{-\alpha u_k}$ é usado para transformar cada recompensa no que ela valeria na primeira época de decisão, levando em conta o fator de desconto.

A recompensa (ou custo) em cada época de decisão é contabilizada usando tanto r como $\mathcal{R}$:

$$R(s,a) = r(s,a) + \int_0^\infty \sum_{s' \in S} \left[\int_0^u e^{-\alpha t} \mathcal{R}(s,s',a) T(s'|s,a) dt \right] F(du|s,a).$$

A função R já inclui o fator de desconto, que para SMDPs é $\gamma = e^{-\alpha t}$.

Pode-se também definir $Q : S \times \mathbb{R} \times S \times A \mapsto [0,1]$, uma função dá a probabilidade de a próxima época de decisão acontecer antes ou exatamente em um momento t, e o sistema estar no estado s' nesta próxima época de decisão, dado que o estado na última época de decisão era s, e a ação executada foi a. Esta função é definida usando T e F: $Q(s', t|s, a) = T(s'|s, a)F(t|s, a)$.

Políticas e funções valor para SMDP tem a mesma forma que seus equivalentes para MDPs, e neste trabalho a notação usada para políticas e funções valor será a mesma para MDPs e SMDPs. Critérios de performance normalmente definidos para SMDPs incluem recompensa média e recompensa total descontada ao longo do tempo.

2.3.1 Algoritmos

O mapeamento h pode ser definido para SMDPs como

$$h(s,a,V) = R(s,a) + \sum_{s' \in S} m(s'|s,a) V(s')$$

onde $m(s'|s,a)$ é a probabilidade de, estando em s e executando a, o próximo estado ser s':

$$m(s'|s,a) = \int_0^\infty e^{-\alpha t} Q(dt, s'|s,a).$$

O operador H é:

$$HV(s) = \max_{a \in A} h(s,a,V).$$

É possível provar que H para SMDPs é uma contração, da mesma forma que para MDPs [69].

A equação de otimalidade para SMDPs é:

$$\begin{aligned} V_k(s) &= HV_{k+1}(s) \\ &= \max_{a \in A} \left[R(s,a) + \sum_{s' \in S} m(s'|s,a) V_{k+1}(s') \right]. \end{aligned}$$

Como descrito por Puterman [69] e Bertsekas [14], é possível usar iteração de valores, iteração de políticas e programação linear para determinar a solução ótima de SMDPs simplesmente trocando $\gamma T(s'|s,a)$ por $m(s'|s,a)$. Singh, Tadić e Doucet mostram também um método para a solução de SMDPs que usa busca pelo gradiente [80].

2.4 Sumário

Este capítulo apresentou os processos de Markov completamente observáveis, os processos semi-Markovianos e os algoritmos clássicos para solução destes. MDPs e SMDPs são estudados extensamente por Puterman [69] e Bertsekas [14]. Boutilier, Dean e Hanks [16] oferecem uma visão geral de planejamento, processos de Markov e complexidade de soluções.

Capítulo 3

Processos de Decisão de Markov Parcialmente Observáveis

Este capítulo descreve os processos de decisão de Markov parcialmente observáveis (*partially observable Markov decision processes* – POMDPs). Um POMDP é uma generalização de MDPs onde o estado atual do sistema não necessariamente é conhecido. Ao invés disso, o tomador de decisões se lembra das ações que executou e das observações que percebeu ao longo do tempo, e tenta usar estas informações para tomar a próxima decisão. É comum, por exemplo, que ao invés de um "estado atual do sistema", uma distribuição de probabilidades sobre os estados seja mantida enquanto as decisões são tomadas. POMDPs são mais difíceis de resolver que os MDPs, mas são mais expressivos.

POMDPs podem ser usados para modelar problemas em muitas áreas diferentes. Anthony Cassandra [20] mostra possibilidades em manutenção de máquinas, navegação de robôs, controle de elevadores, visão computacional, modelagem de comportamento em ecossistemas, problemas militares, diagnóstico médico, educação e várias outras áreas. Alguns casos notáveis de aplicação são o trabalho de Hauskrecht com a modelagem de doenças isquêmicas do coração [35, 38, 36]; o trabalho de Joelle Pineau, que usou POMDPs para modelar o comportamento de um robô para ajudar idosos a se lembrarem de seus compromissos, acompanhá-los e guiar (de maneira limitada) diálogos [60]; e o trabalho de Pascal Poupart, que implementou um sistema que acompanha o comportamento de pacientes com demência, monitorando-os com uma câmera e ajudando com os passos para lavar as mãos, entre outras coisas [65]. Como um exemplo, podemos considerar o seguinte problema:

Uma pessoa está em uma sala onde há duas portas. Atrás de uma das portas há um tigre, que a pessoa deve evitar, e a outra porta é uma saída segura do local onde ela está. Por um certo número de vezes, a pessoa pode escolher entre três ações possíveis:

- Ouvir (para tentar determinar atrás de qual porta o tigre está);

- Abrir a porta à esquerda;
- Abrir a porta à direita.

Depois de algum tempo, o tomador de decisões decidirá por abrir uma das portas, quando deverá finalmente encontrar-se com o tigre ou fugir em segurança. Ao ouvir, no entanto, ele pode mudar seu grau de certeza sobre qual porta esconde o tigre. Cada vez que ouve, ele obtém uma de duas observações:

- O tigre parece estar atrás da porta da esquerda;
- O tigre parece estar atrás da porta da direita.

Estas observações, no entanto, podem não corresponder à realidade, porque ele pode ter se confundido ao tentar determinar de que porta vem o ruído produzido pelo tigre. Ele pode, no entanto, ouvir mais vezes, a fim de aumentar seu grau de certeza.

Há, como podemos ver, incerteza quanto ao estado atual do sistema (o tomador de decisões não sabe atrás de qual porta o tigre está); após cada ação, uma observação é recebida (que não necessariamente informa com total certeza o estado atual); e cada ação pode resultar em uma recompensa (positiva ou negativa). Veremos mais adiante como modelar este problema formalmente como um POMDP.

Definição 3.1 (POMDP)
Um processo de decisão de Markov parcialmente observável (POMDP) é uma tupla (S, A, T, R, Ω, O)*, onde:*

- S *é um conjunto de estados em que o processo pode estar;*
- A *é um conjunto de ações que podem ser executadas em diferentes épocas de decisão;*
- $T : S \times A \times S \mapsto [0, 1]$ *é uma função que dá a probabilidade de o sistema passar para um estado* s'*, dado que estava no estado* s *e a ação executada foi* a*;*
- $R : S \times A \mapsto \mathbb{R}$ *é uma função que dá o custo (ou recompensa) por tomar uma decisão* a *quando o processo está em um estado* s*;*
- Ω *é um conjunto de observações que são obtidas em cada época de decisão;*
- $O : S \times A \times \Omega \mapsto [0, 1]$ *é uma função que dá a probabilidade de uma observação o ser verificada, dados um estado* s *e a última ação* a *executada.*

Em um POMDP não há a possibilidade de se observar diretamente o estado em que o sistema está em um dado momento, ou seja, o tomador de decisões não sabe o estado atual s (ao contrário do que acontece em um problema modelado como MDP). No entanto, cada ação tem como resultado alguma observação que é probabilisticamente relacionada ao estado do sistema.

Definição 3.2 (Estado de informação)
O estado de informação I_k *representa o conhecimento que se tem sobre o sistema na época de decisão* k*, e consiste de:*

- *Uma distribuição de probabilidades sobre os estados na primeira época de decisão, denotado por* b_0*;*
- *O histórico completo de ações e observações, desde a primeira época de decisão.*

O espaço de todos os estados de informação para um POMDP P será denotado por $\mathcal{I}(P)$. Quando não houver ambiguidade, $\mathcal{I}$ será usado no lugar de $\mathcal{I}(P)$.

Uma política para um POMDP deve mapear estados de informação em ações.

Agora podemos formular o problema do tigre como um POMDP:

- S: há dois estados, `tigre_esq` e `tigre_dir`;
- A: há três ações, `ouvir`, `abrir_esq` e `abrir_dir`;
- T: nenhuma ação altera o estado do sistema (por mais que o tomador de decisões goste da idéia, abrir portas ou ouvir não farão o tigre mudar de lugar!);
- R: há uma recompensa positiva para abrir a porta onde o tigre não está, e uma recompensa negativa (custo) para abrir a posta onde o tigre está;
- Ω: há duas observações, `tigre_esq` e `tigre_dir`;
- O: a ação `ouvir` dá a observação correta com probabilidade 0.9.

O problema modelado desta forma pode ser resolvido por algoritmos que determinam a política ótima para POMDPs. Esta política receberá como entrada o estado de informação atual, e a saída será a recomendação de uma ação que maximiza a recompensa esperada do tomador de decisões.

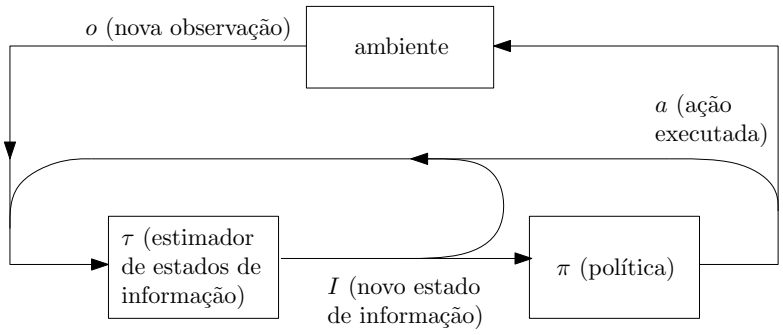


Figura 3.1: Funcionamento de um sistema modelado como POMDP.

3.1 Dinâmica de sistemas modelados como POMDPs

A dinâmica de um sistema modelado como POMDP é mostrada na Figura 3.1: a cada época de decisão, o tomador de decisões verifica o estado de informação atual (I na figura), a última ação executada e a última observação obtida do ambiente. A partir desses dados, um estimador de estados de informação (τ na figura) determina um novo estado de informação, que é usado como entrada para uma política π, que determina a próxima ação a ser tomada. Esta ação tem um efeito sobre o ambiente, que retorna uma observação o. Na próxima época de decisão, esta observação (e a última ação executada) serão usadas para determinar o próximo estado de informação.

A maneira mais direta de representar estados de informação é como uma lista de ações e observações executadas desde a primeira época de decisão (além da distribuição inicial de probabilidades sobre os possíveis estados): o estado de informação na época de decisão k poderia ser denotado $I_k = (b_0, a_0, o_0, a_1, o_1, \cdots, a_{k-1}, o_{k-1})$, onde b_0 é a distribuição inicial de probabilidades sobre os estados. Modificar um estado de informação representado desta forma é trivial: basta adicionar o par (a, o) com a última ação e a observação resultante.

No entanto, uma política para POMDP teria que mapear o espaço $\mathcal{I}$ de todos os estados de informação em ações, e a enumeração de todos os possíveis históricos de um processo é inviável. Usa-se, então, uma *estatística suficiente* para representar o estado de informação atual. Uma estatística muito usada para representar estados de informação é o *estado de crença*, uma distribuição de probabilidades sobre os possíveis estados do sistema. POMDPs onde o estado de informação é representado desta forma são muitas vezes chamados de *belief POMDPs* (POMDPs de crença). Em um POMDP com $|S|$ estados, o espaço dos estados de crença é um $(|S| - 1)$-simplex. A Figura 3.2 mostra o espaço de estados de crença para um POMDP com quatro estados, s_0, s_1, s_2 e s_3. A figura mostra apenas os valores para os estados s_1, s_2 e s_3, já que o valor de s_0 pode ser calculado

a partir dos outros (porque a soma das probabilidades associadas aos estados deve ser um). Este trabalho denota estados de crença por b, e quando conveniente b é usado como um vetor, sendo $b(s)$ o s-ésimo elemento do vetor (que é o valor da probabilidade de o sistema estar no estado s).

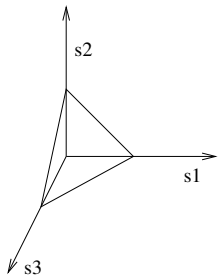


Figura 3.2: Simplex definido por todos os possíveis estados de crença em um POMDP com quatro estados

3.1.1 Modelos de observação

Em POMDPs, observações estão sempre associadas a estados e ações. Há várias maneiras de definir como esta relação ocorre. Algumas delas são:

- *Modelo de observações para a frente* **(forward triggered)**: a observação na época de decisão k está relacionada à ação anterior (em $k-1$) e ao estado resultante (em k). Ou seja, a função de probabilidade de observação poderia ser denotada $O(o^k|s^k, a^{k-1})$, se índices fossem usados para indicar as épocas de decisão de o, s e a;

- *Modelo de observações para trás* **(backward triggered)**: a observação na época de decisão k depende do estado e da ação na época de decisão $k-1$. Ou seja, a função de probabilidade de observação poderia ser denotada como $O(o^k|s^{k-1}, a^{k-1})$;

- *Modelo de observações com atraso* **(delayed)**: uma ação na época de decisão k acarreta uma observação em alguma outra época de decisão $k+j$ (possivelmente várias épocas à frente); a função O poderia ser denotada por $O(o^k|s^{k-j}, a^{k-j})$.

É possível formular problemas como POMDPs de crença para os modelos para trás, para a frente e também para combinações de ambos (onde algumas ações acarretam observações no futuro, e outras imediatamente). Para o modelo com atraso, não é possível usar apenas um estado de crença como estatística suficiente para o estado de informação, mas Hauskrecht mostra que para este caso é possível usar o estado de crença e a lista das últimas k observações, onde k é a maior demora possível para uma observação [36].

A escolha do modelo de observação depende do problema sendo modelado, e é possível usar diferentes modelos de observação para diferentes observações no mesmo POMDP. Neste trabalho, o modelo "para a frente" é usado nas discussões teóricas.

3.2 POMDPs como MDPs sobre estados de crença

Um POMDP pode ser formulado como um MDP onde o conjunto de estados é o simplex de estados de crença, da seguinte maneira:

- $\mathcal{B} = \{b \in \mathbb{R}^{|S|} | \sum_{k=0}^{|S|-1} b_k = 1,\ b_k \geq 0\}$ contém todos os possíveis estados de crença (o espaço de estados é infinito);
- As ações A são as mesmas que no POMDP original;
- O estimador de estados $\tau : \mathcal{B} \times A \times \Omega \mapsto \mathcal{B}$ define o próximo estado de crença, dados o estado de crença anterior (b), a última ação (a) e a última observação (o). Se as observações sempre forem causadas pela ação executada na época de decisão imediatamente anterior, e sabemos que o estado de crença atual é b, a última ação executada foi a, e a observação resultante foi o, o estimador de estados pode calcular o próximo estado de crença b' a partir do estado anterior b usando o teorema de Bayes. Seja $b_a(s)$ a probabilidade de o novo estado ser s, dado que a ação a foi executada:
$$b_a(s) = \sum_{s' \in S} T(s|s', a) b(s'),$$
e seja $b_a(o)$ a probabilidade de a próxima observação ser o, sendo que a foi executada:
$$b_a(o) = \sum_{s \in S} O(o|s, a) b_a(s).$$
O novo estado de crença b' é composto pelas probabilidades $b'(s)$:
$$\begin{aligned} b'(s) &= \frac{O(o|s,a) b_a(s)}{b_a(o)} \\ &= \frac{O(o|s,a) \sum_{s' \in S} T(s|s',a) b(s')}{\sum_{s' \in S} \left[O(o|s',a) \sum_{s'' \in S} T(s'|s'',a) b(s'') \right]}; \end{aligned}$$
- A função T' dá a probabilidade de o sistema passar de um estado de crença b para outro, b', após executar uma ação a:
$$T'(b'|b,a) = P(b'|b,a) = \sum_{o \in \Omega} P(b'|b,a,o) P(o|b,a),$$

onde

$$P(b'|b,a,o) = \begin{cases} 1 & \text{se } \tau(b,a,o) = b'; \\ 0 & \text{em caso contrário;} \end{cases}$$

- A função de recompensa ρ é definida para estados de crença, e dá a esperança da recompensa de cada ação, dadas as probabilidades de o sistema estar em cada estado:

$$\rho(b,a) = \sum_{s \in S} b(s)R(s,a). \tag{3.1}$$

A solução para o MDP sobre espaço contínuo de estados $(\mathcal{B}, A, T', \rho)$ é a solução para o POMDP usado para construí-lo [36, 19].

3.3 Política

Os mesmos critérios de otimalidade usados para MDPs podem ser usados para POMDPs, e a definição de política ótima para POMDP é a mesma que para MDPs, exceto por um fato: a política para POMDPs deve mapear estados de informação, e não estados, em ações. Similarmente, funções valor são definidas para POMDPs (mapeando estados de informação em valores). Como a Definição 3.3 mostra, uma política para POMDP pode ser vista como uma política para o MDP sobre estados de informação.

Definição 3.3 (Política para um POMDP)
Dado um POMDP $P = (S, A, T, R, \Omega, O)$, uma regra de decisão para P em uma época de decisão k é uma função $d_k : \mathcal{I} \mapsto A$ que determina uma ação, dado um estado de informação para P. Para POMDPs de crença, a regra de decisão pode ser $d_k : \mathcal{B} \mapsto A$.

Uma política para um POMDP é um conjunto $\pi = \{d_0, d_1, \cdots, d_{z-1}\}$ de regras de decisão para P.

Definição 3.4 (Função valor de uma política para um POMDP)
Seja P um POMDP e π uma política para P. A função valor da política P é um mapeamento $V^\pi : \mathcal{I} \mapsto \mathbb{R}$ de estados de informação em recompensas esperadas para a política π, de acordo com o critério de otimalidade escolhido.

As definições de espaço de estados e política ótima para MDPs podem ser usadas para POMDPs, com $\mathcal{I}$ (ou $\mathcal{B}$) no lugar de S. Também a função Q é definida para POMDPs como

$$Q^\pi(b,a) = \rho(b,a) + \gamma \sum_{o \in \Omega} Pr(o|b,a)V^\pi(\tau(b,a,o)), \tag{3.2}$$

onde τ é a função de transição para estados de crença (definida na página 30), e $\rho(b,a)$ é a recompensa imediata para uma ação em um estado de crença b, definida na Equação 3.1 (na página 31).

Enquanto políticas para MDPs podem ser representadas de maneira simples, políticas para POMDPs devem representar o mapeamento de um espaço infinito de estados em ações (ou valores). A Definição 3.3 para política é direta e simples, mas a representação usada para a política depende da maneira como estados de informação são representados e dos algoritmos usados para a solução do POMDP. Na discussão a seguir o termo "política" tem seu significado extendido para funções valor, planos condicionais e outras formas de representar o mapeamento de estados de informação em ações.

3.3.1 Hiperplanos

Esta é a representação normalmente usada em algoritmos exatos. Mantém-se um conjunto de vetores associado a cada ação, onde cada vetor define um hiperplano, dando a recompensa esperada para aquela ação, dado o estado de crença. Cada vetor, quando multiplicado por um estado de crença b, dará a recompensa esperada desde que a ação associada a ele seja tomada, e a política ótima seja seguida até a última época de decisão. Este conjunto de hiperplanos é normalmente denotado por Γ.

A representação por hiperplanos só é possível devido a um fato demonstrado por Sondik [78]: a política ótima para POMDPs de horizonte finito é formada por segmentos de hiperplanos, conforme o Teorema 3.1.

Definição 3.5 (PWLC)
*Uma função valor V (que pode ser descrita por um conjunto Γ de vetores) é PWLC ("*piecewise convex and linear*", ou "convexa e composta de segmentos de hiperplanos") se pode ser calculada como*

$$V(b) = \max_{\alpha \in \Gamma} \sum_{s \in S} b(s)\alpha(s).$$

Teorema 3.1 (da forma das funções valor para POMDPs)
A função valor ótima em qualquer época de decisão para um POMDP de horizonte finito é PWLC.

A Figura 3.3 mostra um exemplo de política usando esta representação. Nesta figura, o estado de crença vai de zero a um, representando $p(s_0)$. Temos dois estados, e $p(s_1) = 1 - p(s_0)$.

Cada hiperplano representa o valor esperado para uma ação; o trecho onde um hiperplano domina todos os outros é aquele onde a ação representada por ele é ótima. Na figura, o vetor α_4 é inútil (porque é dominado pelos outros). A ação representada pelo vetor α_1 é ótima sempre que o estado de crença tiver $p(s_0) \geq 0.7$. A ação do vetor α_2 é ótima quando $0.4 \leq p(s_1) \leq 0.7$, e a do vetor α_3 quando $p(s_0) \leq 0.4$.

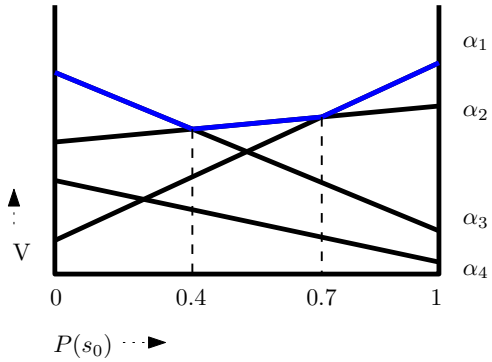


Figura 3.3: Política para POMDP representada como conjunto de hiperplanos.

Para determinar a ação ótima dado um estado de crença b, basta escolher o vetor α que, multiplicado por b, dá o maior valor:

$$d^*(b) = \arg\max_{\alpha \in \Gamma} \sum_{s \in S} b(s)\alpha(s).$$

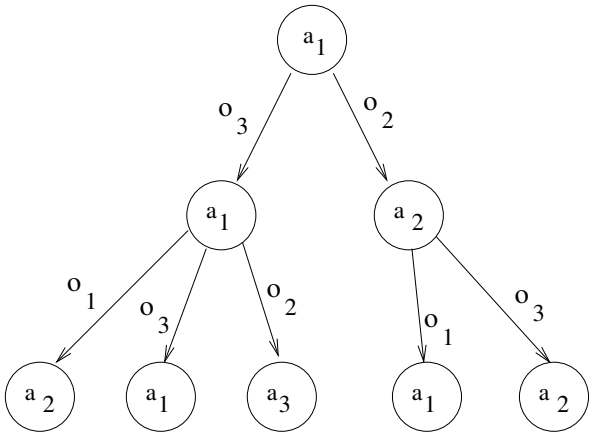
Para horizonte infinito, a função valor ótima pode não ser PWLC, mas é aproximável por uma função PWLC.

3.3.2 Discretização

O simplex do espaço dos estados de crença pode ser discretizado e mapeado em ações [47, 90, 72]. Esta técnica é utilizada com algoritmos de aprendizado, e claramente leva a uma solução não-exata. A grade obtida pela discretização pode ter granularidade e regularidade diferentes dependendo do algoritmo. Alguns destes algoritmos são apresentados na subseção 3.4.3.

3.3.3 Planos condicionais

Uma política pode ser vista como um plano condicional [44], possível de se representar como uma árvore ou como função recursiva (como na Figura 3.4). Nas duas representações, o plano (ρ) começa recomendando a ação a_1 e indicando o próximo plano (ρ_1 nas funções recursivas). O novo plano, dependendo da observação (o_3 ou o_2), pode recomendar a_1 ou a_2, e assim por diante.



$\rho =< a_1, \rho_1 >$
$\rho_1(o_3) =< a_1, \rho_2 >$
$\rho_1(o_2) =< a_2, \rho_3 >$
$\rho_2(o_1) =< a_2 >$
$\rho_2(o_3) =< a_1 >$
$\rho_2(o_2) =< a_3 >$
$\rho_3(o_1) =< a_1 >$
$\rho_3(o_3) =< a_2 >$

Figura 3.4: Política para POMDP representada como árvore e como plano recursivo.

3.3.4 Controladores finitos

Uma forma de representar políticas de horizonte infinito é usando controladores finitos [32]. Estes controladores são similares aos planos condicionais representados como árvores, exceto que formam ciclos fechados. Cada estado representa uma ação, e as arestas representam observações. Ao executar uma ação em um estado e obter uma observação, o próximo estado já é determinado. Um exemplo de controlador é mostrado na Figura 3.5.

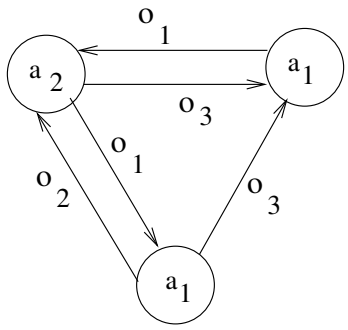


Figura 3.5: Política para POMDP representada como controlador de estados finitos.

O controlador pode também ser estocástico: para cada estado do controlador, há uma distribuição de probabilidade sobre as ações. A ação tomada naquele estado é escolhida pelo controlador de acordo com esta distribuição. A distribuição das probabilidades de geração de cada ação é escolhida de forma a maximizar a recompensa esperada em um horizonte infinito.

3.3.5 Aproximação por redes neurais

Redes neurais podem ser treinadas para aproximar a política ótima, e quando alimentadas com um estado de crença, responderão com a ação ótima [45].

3.3.6 Históricos limitados

Uma outra abordagem é manter apenas parte do histórico (por exemplo, as últimas k ações e observações) como estado de informação, e usar uma política que faça o mapeamento destas sequências em ações [59, 46].

3.4 Algoritmos

O problema de se encontrar uma política ótima para um POMDP é P-ESPAÇO difícil [55] para horizonte finito, e indecidível para horizonte infinito (mas decidível e P-ESPAÇO difícil para ϵ-aproximações [36, 2]). A dificuldade em resolver POMDPs está principalmente em dois fatores:

- A *maldição da dimensionalidade*: para um problema com $|S|$ estados, a política ótima deve ser definida sobre um espaço contínuo de dimensão $|S| - 1$;
- A *maldição do histórico*: a busca pela política ótima assemelha-se a uma busca em largura por todos os possíveis históricos de ação e observação, simulando o POMDP. O número de sequências de ação e observação cresce exponencialmente com o horizonte de planejamento.

Há, no entanto, uma série de algoritmos aproximados e heurísticas que apresentam bom desempenho na prática [39, 60, 65, 68]. Os algoritmos apresentados neste trabalho foram desenvolvidos para POMDPs de crença. Há uma distinção importante a ser feita entre dois problemas que estes algoritmos resolvem:

- *Horizonte finito*: para horizonte finito e política não-estacionária, conhecemos apenas os algoritmos de iteração de valores (que também convergem para a solução ótima no caso de horizonte infinito) e alguns dos algoritmos baseados em pontos de crença;
- *Horizonte infinito*: Todos os outros são algoritmos para horizonte infinito, e muitos convergem para uma ϵ-aproximação da política ótima.

3.4.1 Iteração de Valores

A iteração de valores é a forma clássica de resolução de POMDPs. Estes são algoritmos de programação dinâmica que resolvem o problema para horizonte finito de forma exata (e por isso são muito ineficientes), e conseguem ϵ-aproximações para problemas de horizonte infinito.

O valor de um estado de crença na época de decisão k pode ser dado por

$$V_k^*(b) = \max_{\alpha \in \Gamma_k} \sum_{s \in S} b(s)\alpha(s),$$

onde Γ_k é um conjunto de hiperplanos de dimensão $|S|$ representando a função valor para a política. Se a política for estacionária, Γ_k será o mesmo para todo valor de k.

Assim, $\mathcal{V}$ é para POMDPs o conjunto de todas as funções $f : \mathcal{B} \mapsto \mathbb{R}$. Da mesma forma que para MDPs, um mapeamento $h : \mathcal{B} \times A \times \mathcal{V} \mapsto \mathbb{R}$ dá o valor de uma ação em um estado de crença, somado ao valor de se prosseguir com a política ótima:

$$h(b, a, V) = \sum_{s \in S} \rho(b, a) + \gamma \sum_{o \in \Omega} Pr(o|b, a)V(\tau(b, a, o)).$$

O operador $H : \mathcal{V} \mapsto \mathcal{V}$ de melhoria de política para POMDPs é:

$$HV(b) = \max_{a \in A} h(b, a, V).$$

e a equação de otimalidade para POMDPs é

$$\begin{aligned} HV_k(b) &= \max_{a \in A} h(b, a, V_{k+1}) \\ &= \max_{a \in A} \sum_{s \in S} \rho(b, a) + \gamma \sum_{o \in \Omega} Pr(o|b, a)V_{k+1}(\tau(b, a, o)) \end{aligned}$$

Os diferentes algoritmos de iteração de valores baseiam-se nesta equação de otimalidade, e funcionam basicamente da mesma forma, mostrada no Algoritmo 3.1. A função valor nestes algoritmos é representada por conjuntos de hiperplanos denotados Γ_i, onde i é o número da iteração do algoritmo. A diferença entre os algoritmos de iteração de valores está apenas na forma como calculam Γ_i a partir de Γ_{i-1}. Hiperplanos evitam a representação de pontos de crença diretamente, uma vez que o espaço de crença é infinito (e apenas uma parte dos pontos poderia ser representada). Usando o fato de que

$$V_k(b) = \max_{\alpha \in \Gamma_k} \sum_{s \in S} b(s)\alpha(s),$$

o mapeamento h e o operador H são redefinidos, e a equação de otimalidade pode ser escrita como

$$\Gamma_t = \bigcup_{\substack{a \in A \\ \alpha \in \Gamma_{k+1}}} \left\{ r_a + \gamma \sum_{o \in \Omega} M^{a,o}\alpha \right\} \cup \Gamma_{k+1}.$$

Após fixar a e o, $M^{a,o}$ é uma matriz $|S| \times |S|$, que para cada par de estados s_i e s_j, dá a probabilidade de se chegar a s_j observando o (ou seja, $p(s_j, o|s_i, a)$). Além disso, r_a é um vetor coluna com as recompensas imediatas ($r_a(i) = R(s_i, a)$).

Entrada: Um POMDP (S, A, T, R, Ω, O), e horizonte z.
Saída: Um a política não-estacionária $\mathbf{\Gamma}$. O conjunto Γ_i representa a função valor para a época de decisão $z - i$.

```
para cada a ∈ A faça
    para cada s ∈ S faça
        r_a(s) ← R(s, a)
    fim
    Γ_0 ← Γ_0 ∪ {r_a}
fim
para i ← 1 até z faça
    Γ_i ← passo_dp(Γ_{i−1})
fim
Devolva(Γ)
```

Algoritmo 3.1: Iteração de valores para POMDPs com horizonte finito.

Algoritmo de Sondik

O algoritmo mais simples (e menos eficiente) que se conhece é o algoritmo de enumeração de Sondik [77], que enumera todas as observações e ações possíveis em cada passo de programação dinâmica.

Para construir o conjunto Γ_i, o algoritmo de Sondik considera todos os planos condicionais para serem usados depois de cada ação que poderia ser executada na época de decisão. Cada possível mapeamento de observações em vetores de Γ_{i-1} é um dos possíveis planos a serem seguidos. Assim, um novo vetor em Γ_i será gerado para cada possível permutação de vetores α de Γ_{i-1}, sendo que as permutações tem tamanho $|\Omega|$. Isso porque temos que considerar cada possível ação a ser tomada na atual época de decisão, e para cada uma delas, todos os planos que mapeiam observações resultantes em vetores α da próxima época de decisão. A Figura 3.6 mostra a construção de Γ_1 a partir de Γ_0, com $|A| = 4$ e $|\Omega| = 3$. O número de permutações de observações e vetores é $|\Gamma_{i-1}|^{|\Omega|}$; como todas as permutações devem ser geradas para cada uma das ações possíveis, o número de vetores em Γ_i é $|A||\Gamma_{i-1}|^{|\Omega|}$. A Tabela 3.1 mostra o tamanho dos conjuntos Γ_i para alguns passos, usando os parâmetros do exemplo mostrado.

A equação de recorrência para o número de vetores (cuja geração é a parte mais custosa do algoritmo) é $T(|A|, |\Omega|, z) = |A|T(|A|, |\Omega|, z-1)^{|\Omega|}$. Se considerarmos que a construção

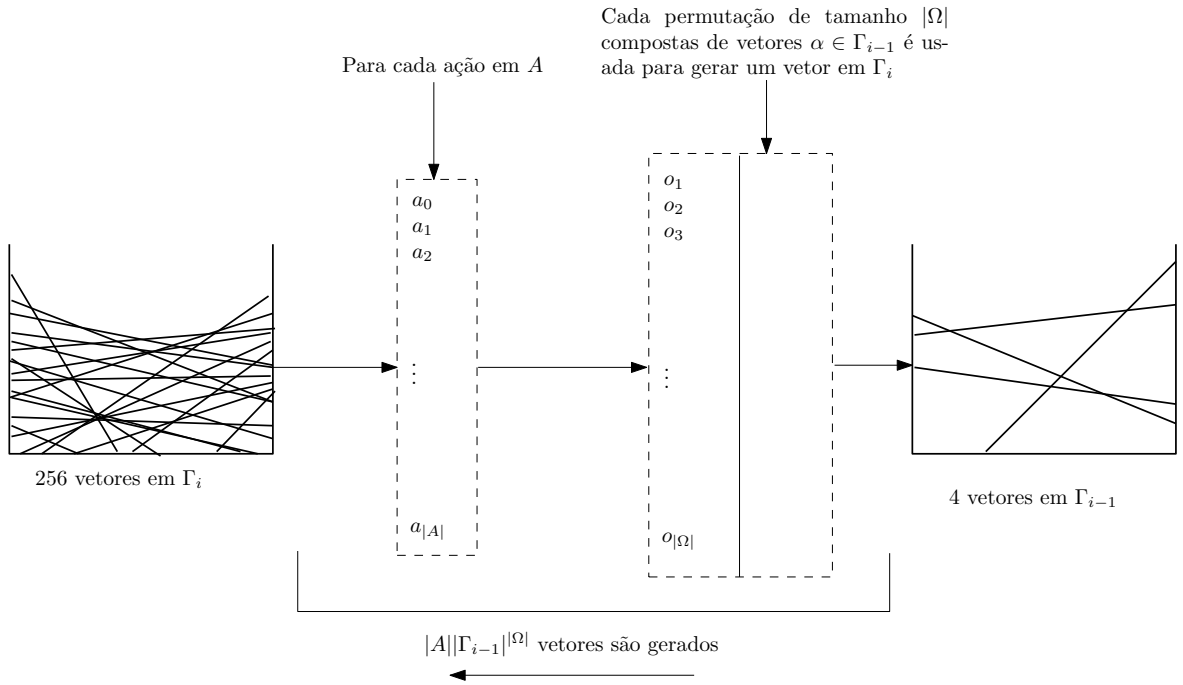


Figura 3.6: Um passo de programação dinâmica do algoritmo de Sondik.

Passo	Número de vetores
0	$\|\Gamma_0\| = 4$
1	$\|\Gamma_1\| = 256$
2	$\|\Gamma_2\| = 67108864$
3	$\|\Gamma_3\| = 1.2089 \times 10^{24}$

Tabela 3.1: Evolução do tamanho da representação da política gerada pelo algoritmo de Sondik, com $|A| = 4$ e $|\Omega| = 3$.

de cada vetor α toma tempo $\mathcal{O}(|S|)$, temos a complexidade de tempo do algoritmo igual a $|S||A|^z|A|^{|\Omega|^z} = \mathcal{O}(|S||A|^{|\Omega|^z})$ – exponencial em $|\Omega|$ e duplamente exponencial no horizonte z.

Algoritmo de Monahan

Ao gerar um novo conjunto Γ_k, normalmente são gerados mais vetores que o necessário, havendo então uma grande quantidade de vetores dominados que, se eliminados da representação, reduzem o tempo de execução do algoritmo. Uma representação da função objetivo sem vetores dominados é chamada de *representação parcimoniosa*, e pode ser obtida por programação linear [19].

O algoritmo de Monahan [52] é semelhante ao de Sondik, exceto por fazer um teste de

Entrada: Conjunto Γ_{k-1} de vetores, descrevendo a função objetivo no passo $k-1$.
Saída: Conjunto Γ_k de vetores, descrevendo a função objetivo no passo k.

$\Gamma \leftarrow \emptyset$
para cada $a \in A$ **faça**
  para cada $\alpha \in \Gamma_{k-1}$ **faça**
    $\Gamma \leftarrow \Gamma \cup \left\{ r_a + \gamma \sum_{o \in \Omega} M^{a,o} \alpha \right\}$
  fim
fim
Devolva Γ

Algoritmo 3.2: passo_dp - algoritmo de Sondik.

dominância no conjunto de vetores ao final do passo de programação dinâmica, retornando uma representação parcimoniosa da função.

Algoritmo de suporte linear de Cheng

O algoritmo de suporte linear, proposto por Cheng [22], constrói a função valor gradualmente, procurando pela vizinhança de vetores úteis. Em uma função objetivo representada por um conjunto de vetores Γ, para cada vetor α não-dominado, existe um estado de crença b onde $\forall \alpha' \in \Gamma,\ \alpha b \geq \alpha' b$. Este estado de crença é chamado de testemunha (*witness*) do vetor α, porque prova que o vetor é útil.

O algoritmo escolhe um vetor inicial, como se este representasse perfeitamente a função valor. Em seguida, procura outros vetores, que mostrem que aquele vetor inicial não é ótimo em todo o espaço de crença, pesquisando uma sequência de vetores "geometricamente vizinhos" que também devem ser incluídos na função valor. Na Figura 3.7, por exemplo, o vetor α_0 foi adicionado inicialmente a $\widehat{\Gamma}_i$. Com um único vetor, há dois pontos extremos. Um fica na borda do estado de crença, e o outro em um ponto interno. O algoritmo seleciona o ponto interno, e verifica que α_0 não é ótimo naquela região: a testemunha disso é α_1, que é ótimo. O novo vetor α_1 é adicionado a $\widehat{\Gamma}_i$. O Algoritmo 3.3 detalha este procedimento.

O algoritmo de Cheng depende de dois pontos importantes:

- Dado um vetor útil $\alpha_i^k \in \widehat{\Gamma}_i$, computar todos pontos extremos da região definida por α_i^k tais que α_i^k seja ótimo na região definida por $\widehat{\Gamma}_i$. Idealmente, o algoritmo deveria verificar apenas os vértices a partir dos quais possa gerar vetores úteis;

- Computar vetores úteis para um dado estado de crença (pode haver vetores que são ótimos apenas para um ponto).

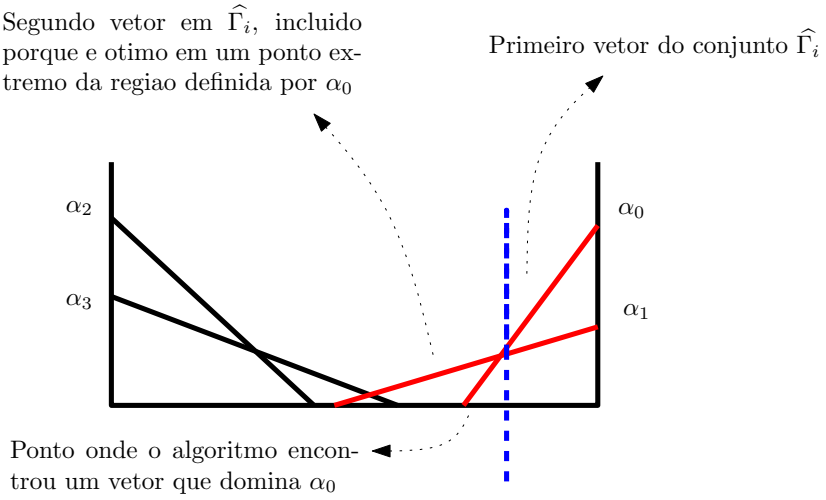


Figura 3.7: Construção da função valor usando o algoritmo de suporte linear de Cheng.

A discussão dos detalhes destas duas tarefas está fora do escopo deste trabalho, mas pode ser encontrada no trabalho de Hauskrecht [36].

O algoritmo *"Witness"* de Cassandra, Kaelbling e Littman

Cassandra, Kaelbling e Littman propuseram um algoritmo chamado de "witness" [19], parecido com o de Cheng. A diferença entre o algoritmo de Cassandra e o de Cheng é que o algoritmo de Cassandra encontra vetores Γ_i^a para cada ação, uma por vez, determinando funções valor $Q(\cdot, a)$, e depois as combina para formar uma única função valor Γ_i. Este algoritmo tem a vantagem de sempre conseguir, para um dado vértice, determinar apenas pontos no estado de crença para os quais há a certeza de haver vetores úteis a serem incluídos em Γ_i (o algoritmo de Cheng faz uma busca exaustiva, testando pontos que podem não gerar qualquer vetor útil). A parte mais custosa do algoritmo é a enumeração de todos os vértices da região.

O algoritmo *incremental pruning*, de Zhang e Liu

Outro algoritmo, proposto por Zhang e Liu [87, 21] é o *incremental pruning*. Os autores do algoritmo verificaram que o teste de dominância é muito caro se realizado para todos os vetores de uma só vez. O algoritmo faz testes de dominância após calcular vetores para cada ação e conjunto de observações. O Algoritmo 3.4 mostra a construção da função valor por *incremental pruning*. O símbolo $\oplus$ representa adição de pares ($\Gamma_1 \oplus \Gamma_2 = \{\alpha_1 + \alpha_2 | \alpha_1 \in \Gamma, \alpha_2 \in \Gamma_2\}$).

Entrada: Um conjunto Γ_{i-1} de vetores representando a função valor da iteração anterior.
Saída: Um conjunto Γ_i de vetores representando a função valor da i-ésima iteração.

```
b ← ponto qualquer do espaço de crença
Γ̂_i ← conjunto de vetores úteis para b
marque todos os vetores em Γ̂_i
enquanto há um vetor marcado em Γ̂_i faça
    α ← algum vetor de Γ̂_i que tenha sido marcado
    E ← todos os pontos extremos da região para qual α dá valor ótimo (usando
    outros valores de Γ̂_i)
    para cada b' ∈ E faça
        α' ← algum vetor útil para b'
        se α' ∉ Γ̂_i então
            Γ̂_i ← Γ̂_i ∪ {α'}
            marque α'
        fim
    fim
fim
Devolva Γ̂_i
```

Algoritmo 3.3: Construção da função valor Γ_i usando o algoritmo de suporte linear de Cheng.

Convergência

Da mesma forma que para MDPs, o operador H para POMDPs é uma contração. Isto garante a convergência do algoritmo de iteração de valores para horizonte infinito. No entanto, a função valor converge para uma aproximação da solução ótima. A função valor ótima para um POMDP não é necessariamente PWLC, como no caso de horizonte finito, mas pode ser aproximada por uma função PWLC para o critério da recompensa total descontada.

Diferentes critérios de parada podem ser usados com o algoritmo de iteração de valores, resultando em diferentes aproximações. Alguns dos critérios normalmente usados são:

- *Convergência exata*: pode-se exigir que as funções valor Γ_i e Γ_{i-1} sejam idênticas para que o algoritmo pare. No entanto, este critério não será sempre satisfeito;

- *Resíduo de Bellman* (também chamado de *erro de Bellman*): o erro e é a distância

Entrada: Conjunto de vetores Γ_{i-1}, descrevendo a função objetivo no passo $i-1$.
Saída: Conjunto de vetores Γ_i, descrevendo a função objetivo no passo i.

$\Gamma \leftarrow \emptyset$
para cada $a \in A$ **faça**
  $\Gamma_a \leftarrow \emptyset$
  para cada $o \in \Omega$ **faça**
    para cada $\alpha \in \Gamma_{i-1}$ **faça**
      $\Gamma_{a,o} \leftarrow \text{prune}(\frac{1}{|\Omega|} r_a + \gamma M^{a,o} \alpha)$
      $\Gamma_a \leftarrow \text{prune}(\Gamma_a \oplus \Gamma_{a,o})$
    fim
    $\Gamma \leftarrow \text{prune}(\Gamma \cup \Gamma_a)$
  fim
fim
Devolva Γ

Algoritmo 3.4: passo_dp - incremental pruning.

máxima entre as duas funções valor:

$$e = \max_{\substack{\alpha \in \Gamma_i \\ \alpha' \in \Gamma_{i-1} \\ b \in \mathcal{B}}} \left[b\alpha - b\alpha' \right];$$

- *Convergência fraca*: pode-se usar como erro também a distância máxima entre os valores de Γ_i e Γ_{i-1} para crença igual a 1 em cada estado:

$$e = \max_{\substack{\alpha \in \Gamma_i \\ \alpha' \in \Gamma_{i-1} \\ s \in S}} \left[\alpha(s) - \alpha'(s) \right].$$

Para o resíduo de Bellman e convergência fraca, o algoritmo para quando $e \leq Z\frac{1-\gamma}{2\gamma}$, onde Z é um parâmetro de entrada.

3.4.2 Iteração de Políticas

Os métodos de iteração de valores representam a política como conjunto de hiperplanos, realizando uma busca no espaço de funções valor. Como já visto, a política para um POMDP pode ser também representada como um controlador de estados finitos. Há algoritmos que representam a política como controladores, mantendo paralelamente uma representação por hiperplanos para poder avaliar o controlador. Estes algoritmos realizam a busca no espaço de políticas. Aqui são apresentados os algoritmos de iteração de políticas de Hansen e o BPI de Poupart. Além destes, há também métodos de subida pelo gradiente

para determinação de um controlador para POMDPs, como o de Aberdeen e Baxter [1, 3], o de Meuleau e outros [50], e o Pegasus de Ng e Jordan [54].

Algoritmo de Hansen

O método apresentado a seguir tem muito em comum com os algoritmos de iteração de valor.

Sondik [77] desenvolveu um algoritmo de iteração de políticas para POMDPs, pouco usado na prática por ser difícil de implementar. Hansen [33, 34] propôs um algoritmo de iteração de políticas mais simples e eficiente que o de Sondik; o algoritmo de Hansen representa a política como um controlador de estados finitos, e pode ser usado para resolver POMDPs de horizonte infinito. Cada estado k do controlador representa um plano condicional que inicia com uma ação $a(k)$, e as arestas entre estados são observações. Quando o tomador de decisões estiver em um estado k, executará a ação a, e escolherá o próximo estado de acordo com a observação obtida.

O algoritmo de Hansen computa uma série de controladores finitos $(\pi_0, \pi_1, \ldots, \pi_n)$ tais que no limite (quando $n \to \infty$), π_n converge para a política ótima π^*. Para melhorar uma política, o algoritmo repete dois passos: a *avaliação da política* e a *melhora da política*.

Durante a avaliação de um controlador de estados finitos π_i, o algoritmo calcula um conjunto Γ_i de vetores α correspondente a π_i (este passo é uma simples tradução da política de uma representação para outra) Para isto, resolve o seguinte sistema linear:

$$\forall s \in S,\ \forall k \in K,$$
$$\alpha_s^k = R(s, a(k)) + \gamma \sum_{s' \in S, o \in \Omega} T(s'|s, a(k)) O(o|s, a(k)) \alpha_{s'}^{t(k,o)} \tag{3.3}$$

onde K é o conjunto de estados do controlador finito e $t(k, o)$ é o índice do próximo estado do controlador caso a observação o aconteça, e α_s^k é o valor do vetor α associado ao estado k do controlador e o estado s do POMDP.

No passo de melhora da política, um passo de programação dinâmica é aplicado à função valor V do controlador atual, e uma nova função valor V' é obtida. Cada vetor α da nova função valor tem uma ação e um plano condicional associados – e é adicionado como um novo estado do controlador. Após a adição de novos estados, a função valor passa por teste de dominância: se o valor (o vetor α) em um estado é completamente dominado por algum outro vetor, este estado é removido. Quando um estado k é removido e há arestas entrando em k, estas são redirecionadas para o estado que dominava k (e causou sua remoção). A Figura 3.8, da tese de doutorado de Pascal Poupart [65], mostra a eliminação de um estado: uma função valor V continha dois vetores, n_1 e n_2. Após a melhora da política, dois novos vetores (e dois novos estados do controlador) foram adicionados: n_3 e n_4. O vetor n_1 é completamente dominado por n_4, e por isso é removido.

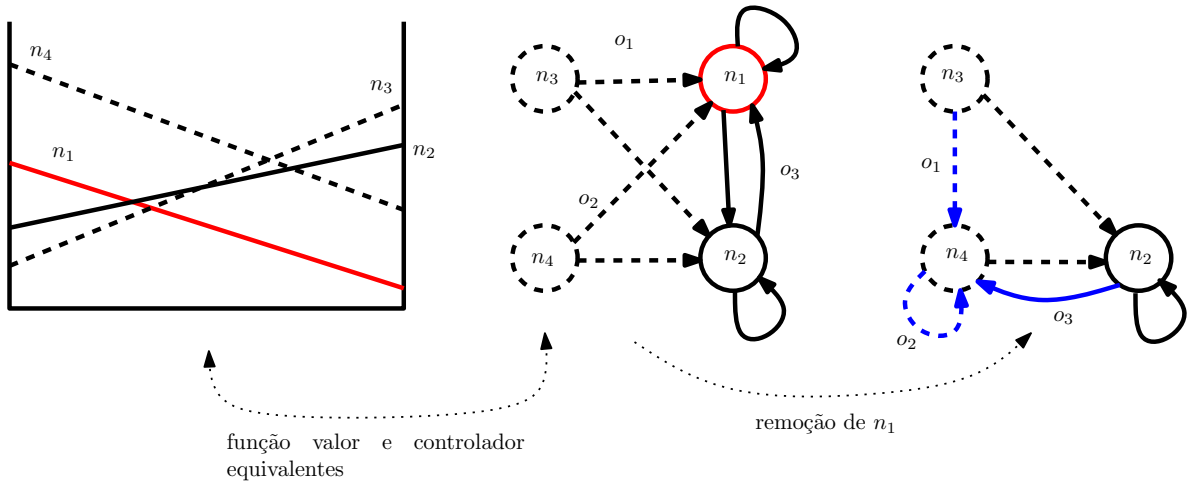


Figura 3.8: Remoção de um estado do controlador no algoritmo de Hansen.

Como havia arestas entrando em n_1, elas foram todas direcionadas para n_4 (as arestas são com a observação o_1, vindo de n_3; com a observação o_2, vindo do próprio n_4; e com a observação o_3, vindo de n_2).

O Algoritmo 3.5 mostra ambos os passos. O critério de convergência pode ser o mesmo usado nos algoritmos de iteração de valores para horizonte infinito.

O algoritmo de Hansen tem muita semelhança com os algoritmos de iteração de valores, gerando uma quantidade exponencial de vetores α (até $|A||K|^{|\Omega|}$ por passo de programação dinâmica), e um mesmo número de estados no controlador. Com isto o controlador acaba crescendo com um número exponencial de estados. Da mesma forma que nos algoritmos de iteração de valores, estados dominados são removidos, mas na prática o desempenho do algoritmo é exponencial.

BPI de Poupart

O próximo algoritmo funciona apenas para horizonte infinito, e não há prova de convergência (como há para iteração de valores), mas há a garantia de que a política cresce monotonicamente, e há um método para escapar de ótimos locais.

Pascal Poupart desenvolveu um algoritmo de iteração de políticas que limita o crescimento do número de estados do controlador chamado *Bounded Policy Iteration* (BPI) [65, 67, 68]. O BPI também alterna passos de avaliação e melhoria da política. O passo de avaliação da política é feito da mesma forma que no algoritmo de Hansen. O passo de melhoria da política é mais eficiente, evitando incluir novos estados no controlador. Ao contrário do algoritmo de Hansen, o BPI pode ficar preso em um ótimo local, mas Poupart descreve um procedimento para escapar de ótimos locais e relata que o algoritmo

Entrada: Um POMDP (S, A, T, R, Ω, O).
Saída: Uma política para o POMDP, representada como um controlador C.

$C_0 \leftarrow$ controlador inicial escolhido aleatoriamente, cujos estados são denotados por K
$i \leftarrow 0$
repita
    $\Gamma_i \leftarrow$ avaliação do controlador C_i usando o sistema de equações 3.3
    Melhore a política: **para cada** $\alpha \in \Gamma_i$ **faça**
        se *ação e estratégia de* α *são idênticos a algum* $k \in K$ **então**
            Mantenha k no controlador
        fim
        senão se α *domina* α' *associado a algum estado* $k \in K$ **então**
            Mude a ação a e as transições de estratégia no controlador para que sejam as mesmas de α
        fim
        senão
            Adicione um novo estado k' no controlador, com a ação e a estratégia de α
        fim
    fim
    Realize o *prune* de quaisquer vértices que não tenham vetor α correspondente em Γ_i, desde que não seja alcançável a partir de um vértice com vetor correspondente em Γ_i
    $i \leftarrow i + 1$
até *que um critério de convergência seja atingido*

Algoritmo 3.5: Algoritmo de Hansen para POMDPs.

tem muito bom desempenho na prática.

A observação chave de Poupart é que o algoritmo de Hansen só remove vetores completamente dominados por um outro vetor, e não os vetores dominados por toda a nova função valor. Isto é necessário porque o algoritmo de Hansen foi projetado para funcionar com políticas determinísticas quanto à estratégia de observação: uma vez que uma ação é executada e uma observação é obtida, a próxima ação já está determinada. Isto é diferente do que acontece com os algoritmos de iteração de valores: após executar uma ação e obter uma observação, o próximo passo é calcular o próximo estado de crença e então escolher a próxima ação.

O algoritmo BPI gera um controlador estocástico de estados finitos: ao invés de uma observação ser representada como uma única aresta no controlador (e levar a um novo estado que representa uma única ação), a mesma observação pode levar a vários estados

diferentes do controlador. A distribuição de sucessores de um par k, o (isto é diferente de um par ação/observação, já que vários estados podem representar a mesma ação) é dada por $\sigma(k, o, k') = P(k'|k, o)$ (ou seja, $\sigma(k, o, k')$ é a probabilidade de que o próximo estado do controlador seja k', dado que o estado atual é k e a observação o foi obtida). Uma estratégia estocástica de observação encontra combinações convexas dos possíveis estados sucessores que dominem completamente vetores de estados antigos. A Figura 3.9 mostra os vetores α de três estados: n_1, n_2 e n_3. O estado n_2 claramente poderia ser removido, mas não o seria pelo algoritmo de Hansen. No entanto, a combinação convexa de n_1 e n_3 domina completamente n_2. O algoritmo BPI elimina n_2 e redireciona as arestas deste estado para esta combinação convexa.

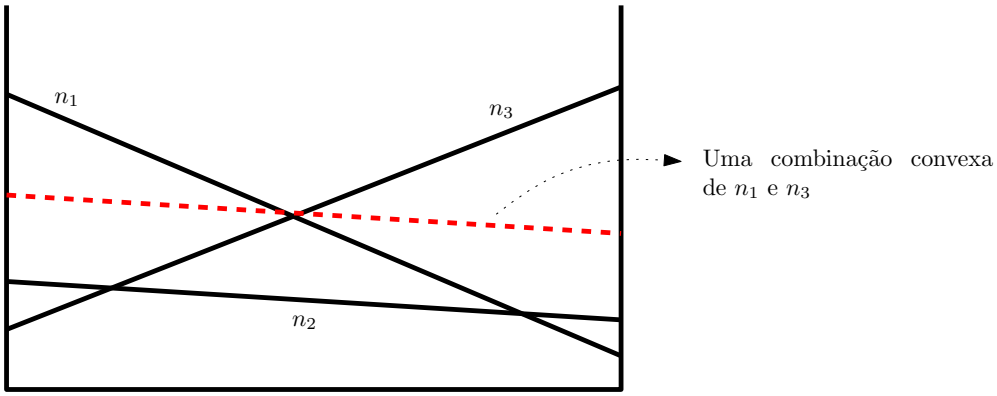


Figura 3.9: Vetor dominado pela combinação convexa de outros dois vetores no algoritmo BPI.

Enquanto o algoritmo de Hansen precisa de um passo de programação dinâmica no início da melhoria da política, o BPI evita fazê-lo para não adicionar estados demais ao controlador. A melhoria é feita considerando um estado por vez, e substituindo este estado por uma combinação convexa dos estados que *seriam obtidos* em um passo de programação dinâmica, mas sem computar explicitamente este passo.

O BPI usa as seguintes variáveis:

- $c_{a,k_1,k_2,\cdots,k_{|\Omega|}}$ denota a probabilidade de que após a ação a ser executada, o plano seja descrito por $k_1 \cdots k_{|\Omega|}$, da seguinte forma: o próximo estado é k_1 se a observação for o_1, k_i se a observação for o_i, etc. Por exemplo, se $\Omega = \{o_0, o_1, o_2\}$, c_{a,k_3,k_6,k_1} dá a probabilidade de que este estado do controlador execute a e que em seguida o próximo estado dependa da observação obtida: $o_0 \mapsto k_3, o_1 \mapsto k_6, o_2 \mapsto k_1$;

- $c_a = \sum_{k_1,k_2,\cdots,k_{|\Omega|}} c_{a,k_1,k_2,\cdots,k_{|\Omega|}}$ é a probabilidade agregada de todos os planos que iniciam com a ação a;

- $c_{a,k_o} = \sum_{k_1,k_2,\cdots,k_{o-1},k_o+1,\cdots,k_{|\Omega|}} c_{a,k_1,k_2,\cdots,k_{|\Omega|}}$ é a probabilidade agregada de todos os planos que executam a e chegam ao estado k_o após observar o;

O seguinte programa linear então é resolvido para cada estado k:

maximize δ, sujeito a

$$\begin{aligned}
\forall s \in S,\ \alpha_n(s) + \delta &\leq \sum_{a\in A}\left[c_a R(s,a) + \gamma \sum_{s'\in S, o\in\Omega} T(s'|s,a)O(o|s',a)\sum_{k\in K} c_{a,k_o}\alpha_{k_o}(s')\right], \\
\sum_{a\in A} c_a &= 1, \\
\forall a \in A,\ \sum_{k_o} c_{a,k_o} &= c_a, \\
\forall a \in A,\ c_a &\geq 0, \\
\forall a \in A, o \in \Omega,\ c_{a,k_o} &\geq 0.
\end{aligned}$$

Há $|A||K||\Omega| + |A| + 1$ variáveis no programa linear (as variáveis c_{a,k_o}, as variáveis c_a e δ). O valor δ mede a melhoria da função valor para este estado do controlador, e as variáveis obtidas são usadas para o estado na construção do controlador estocástico: o estado k do controlador escolherá a com probabilidade c_a, e depois de receber uma observação o, mudará para o estado do controlador k_o com probabilidade c_{a,k_o}.

A cada passo do algoritmo, todos os estados antigos do controlador são descartados e substituídos por estados novos com função valor melhor. A Figura 3.10 mostra o passo do algoritmo BPI: há uma função valor V com um vetor para cada estado do controlador (n_1, n_2 e n_3). O segundo diagrama da figura mostra a melhoria de um dos estados: a função valor que *resultaria* de um passo de programação dinâmica é mostrada (n_4 até n_7), e o vetor n_2 é dominado por uma combinação convexa de n_4 e n_5. O terceiro diagrama mostra todos os vetores melhorados. O último diagrama mostra a nova função valor (que *não* é igual à função valor que o passo de programação dinâmica teria gerado).

O BPI pode ficar preso em um ótimo local, como mostra a Figura 3.11, mas Poupart descreve um método para escapar desta situação, que adiciona um novo estado ao controlador.

Amato, Bernstein e Zilberstein reportam melhorias no BPI com o uso de programas lineares com restrições quadráticas ao invés de programação linear pura [7].

3.4.3 Discretização do espaço de crença

Há uma classe de heurísticas para resolver POMDPs baseadas na idéia de que deve ser suficiente planejar para apenas alguns pontos de crença, e não necessariamente para todos. Estes métodos variam na forma como determinam a grade de pontos de crença usada.

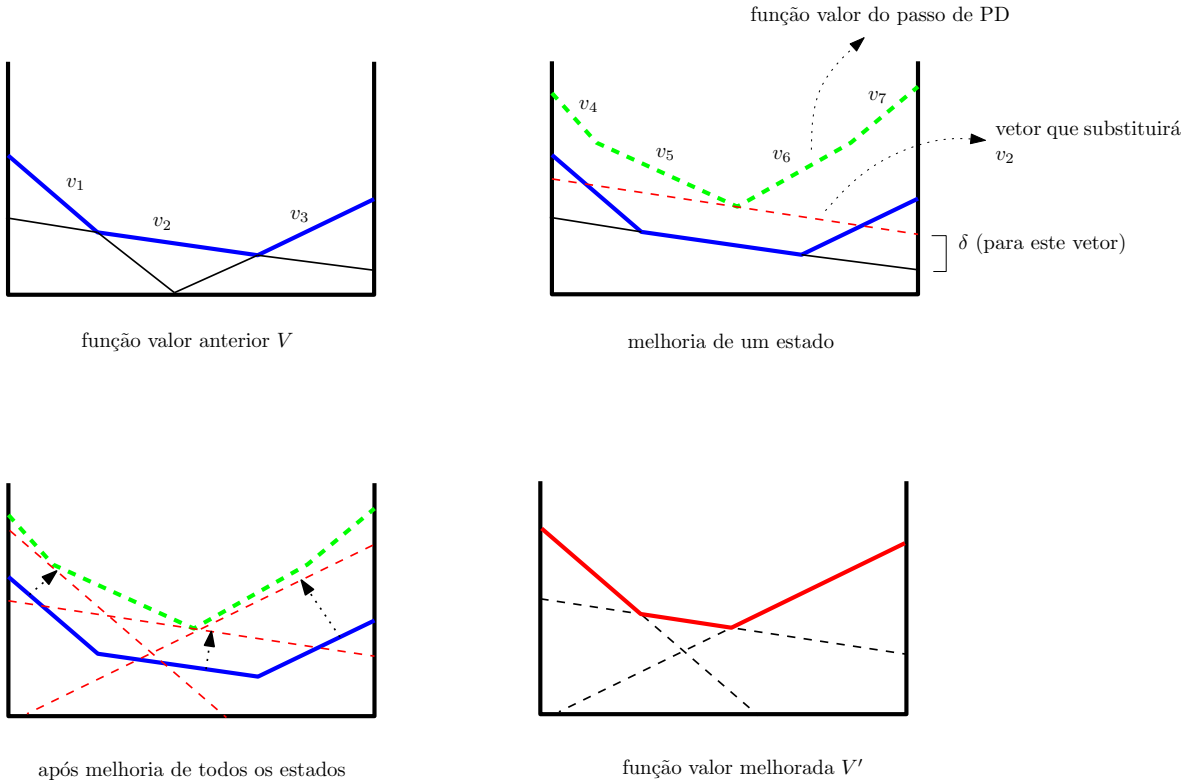
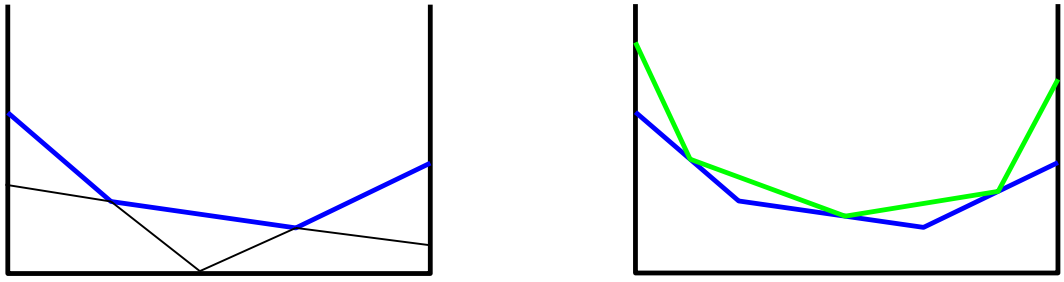


Figura 3.10: Melhoria da política no algoritmo BPI.

Os primeiros métodos baseados em grades de pontos de crença predatam os métodos exatos [26, 64]. De acordo com Cassandra, todos apresentavam performance muito ruim para problemas médios [19]. Há hoje algoritmos melhores nesta classe [82, 90], e são particularmente interessantes os métodos que aumentam a grade à medida que passos de programação dinâmica são executados, como o PBVI de Pineau [60, 61] e o HSVI de Smith e Simmons [74, 75]. O Perseus de Spaan e Vlassis é outro algoritmo baseado e pontos de crença, que executa o passo de programação dinâmica em apenas uma parte dos pontos [79].

As próximas subseções apresentam um método simples, usado nos experimentos do Capítulo 6, o RTDP-Bel, que simula a execução do POMDP enquanto procura pela política ótima, e dois métodos mais recentes, o PBVI e o HSVI. O primeiro método pode ser usado tanto para horizonte finito como para horizonte infinito, e os outros são apropriados apenas para horizonte infinito.



função valor anterior V

função valor do passo de PD toca cada vetor da função valor anterior

Figura 3.11: Ótimo local no algoritmo BPI.

Método básico de discretização

Este primeiro método descrito apresenta os conceitos básicos usados em outros métodos baseados em pontos de crença.

O estado de crença inicial pode ser definido de várias maneiras. Hauskrecht [39] menciona três formas de inicializar uma grade com pontos de crença:

- *Grade regular*: o espaço de crença é dividido em pontos equidistantes;
- *Grade aleatória*: pontos aleatórios são selecionados do espaço de crença;
- *Grade heurística*: pontos são adicionados de acordo com uma heurística que tenta maximizar a utilidade dos pontos ao mesmo tempo em que tenta diminuir a quantidade de pontos.

Há diversas heurísticas para a determinação da grade inicial. Normalmente, um conjunto de pontos é expandido usando uma simulação até gerar uma quantidade de pontos que possa ser usada na grade inicial.

O conjunto de pontos inicial pode ser, por exemplo, a crença uniforme ($b(s) = \frac{1}{|S|}$), ou os cantos do simplex de crença ($b(s) = 1, \quad \forall s \in S$). Um outro método é descrito no Algoritmo 3.6, que gera estados de crença usando observações para agrupar estados em classes de equivalência.

Após determinar este conjunto de pontos, uma busca em largura é feita nos possíveis estados de crença seguintes, usando o estimador de estados e simulando cada par de ação e observação, como mostra o Algoritmo 3.7. O resultado desta busca em largura é a grade inicial de pontos.

Depois da inicialização da grade, passos de melhoria são aplicados a cada ponto de crença.

Entrada: Um POMDP (S, A, T, R).
Saída: Um conjunto l de pontos de crença.

```
1  l ← ∅
2  para cada a ∈ A, o ∈ Ω faça
3  |  para cada s ∈ S faça
4  |  |  se O(o|s, a) > 0 então
5  |  |  |  S' ← S' ∪ {s}
6  |  |  fim
7  |  fim
8  |  para cada s ∈ S' faça
9  |  |  b(s) = 1/|S'|
10 |  fim
11 |  l ← l ∪ {b}
12 fim
13 Devolva l
```

Algoritmo 3.6: Inicialização dos estados de crença em uma grade.

O valor de uma ação em um estado de crença é calculado usando as Equações 3.1 e 3.2 (na página 31).

O passo de melhoria para pontos de crença é semelhante ao passo de programação dinâmica, exceto por ser calculado apenas para os pontos da grade:

$$V_{i+1}(b) = \max_{a \in A} \left[\rho(b, a) + \gamma \sum_{o \in \Omega} Pr(o|b, a) V_i(\tau(b, a, o)) \right].$$

Se $\tau(b, a, o)$ não estiver na grade, seu valor é determinado por interpolação. Esta equação pode ser calculada rapidamente, ao contrário de um passo exato de programação dinâmica para todo o espaço de crença.

RTDP

Geffner e Bonet desenvolveram uma variante do RTDP para POMDPs, chamado RTDP-Bel [28].

O Algoritmo 3.8 executa uma rodada do RTDP-Bel, dado um estado de crença inicial. A escolha do conjunto de estados iniciais (que se resume na discretização do espaço de estados) já foi discutida.

Para pontos de crença que ainda não tem valor definido, o algoritmo usa um limite inferior h. Geffner e Bonet sugerem usar o valor ótimo da política para o MDP subjacente:

$$h(b) = \sum_{s \in S} b(s) V^*_{MDP}(s).$$

Entrada: Um conjunto de estados de crença B, representado como fila.
Saída: Um conjunto expandido de estados de crença.

```
enquanto limite não é atingido faça
    b ← B.proximo
    para cada a ∈ A, o ∈ Ω faça
        b' = τ(b, a, o)
        B.enfilere(b')
    fim
fim
Devolva B
```

Algoritmo 3.7: Simulação de estados de crença para inicialização da grade.

Hauskrecht discute diversas outras heurísticas que podem ser usadas como limite inferior para funções valor de POMDPs [39].

PBVI

O algoritmo *Point-Based Value Iteration* desenvolvido por Joelle Pineau representa a função valor como um conjunto de pontos e um vetor α relacionado a cada ponto. O algoritmo intercala alguns passos de programação dinâmica modificados com passos de expansão do conjunto de pontos de crença usados.

A função valor é representada como um conjunto de pontos e vetores α associados a estes pontos, com mostra a Figura 3.12. Nesta figura, um conjunto de quatro pontos de crença, $B = \{b_0, b_1, b_2, b_3\}$. A partir destes quatro pontos, a função valor é construída com quatro vetores (um por ponto).

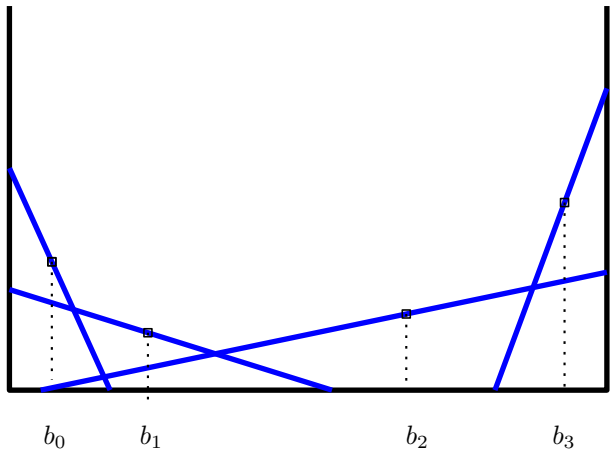


Figura 3.12: Representação de função valor no PBVI.

Entrada: Um POMDP (S, A, T, R, Ω, O);
Uma função valor V para o POMDP;
Um estado de crença inicial b_0.
Saída: Uma função valor V para o POMDP dado como entrada.

$b \leftarrow b_0$
enquanto *o critério de final da rodada não for satisfeito* **faça**
  para cada $o \in \Omega$ **faça**
    $b' \leftarrow \tau(b, a, o)$
    se b' *ainda não está na grade* **então**
      Inclua b' na grade
      $V(b') \leftarrow h(b')$
    fim
  fim
  Avalie o valor de cada ação no estado atual:
  $Q(b, a) \leftarrow \rho(b, a) + \gamma \sum_{o \in \Omega} b_a(o) V(b')$
  $a^* \leftarrow \arg\max\ Q(b, a)$
  $V(b) \leftarrow Q(b, a^*)$
  Simule a execução da ação a^* no estado de crença b
  Observe a observação resultante b'
  $b \leftarrow \tau(b, a, o)$
fim
Devolva V

Algoritmo 3.8: Algoritmo RTDP para POMDPs.

Dado um conjunto Γ_{k-1}, o passo de programação dinâmica é modificado de forma que apenas um vetor α é mantido por ponto de crença.

Para cada ação, os seguintes vetores dão a recompensa imediata:

$$\Gamma^{a,*} \leftarrow \alpha^{a,*}(s) = R(s,a).$$

Os vetores provenientes de outras funções valor são usados:

$$\Gamma^{a,o}(s) \leftarrow \alpha^{a,o}(s) = \gamma \sum_{s' \in S} T(s'|s,a) O(o|s',a) \alpha'(s'),\ \forall \alpha' \in \Gamma_{k-1}. \tag{3.4}$$

Enquanto o passo de programação dinâmica para POMDPs faz um produto de vetores, o PBVI usa apenas uma soma. O conjunto $\Gamma_b^a, \forall b \in B, \forall a \in A$ é construído de acordo com a fórmula:

$$\Gamma_b^a = \Gamma^{a,*} + \sum_{o \in \Omega} \arg\max_{\alpha \in \Gamma^{a,o}} \left[\sum_{s \in S} \alpha(s) b(s) \right].$$

Finalmente, a melhor ação para cada ponto é determinada:

$$\Gamma_i \leftarrow \arg\max_{\Gamma_b^a, \forall a \in A} \left[\sum_{s \in S} \Gamma_b^a(s) b(s), \forall b \in B \right].$$

Esta função valor pode ser usada para calcular o valor de qualquer ponto no espaço de crença, da mesma forma que as funções valor para POMDPs vistas no Capítulo 3:

$$V_k(b) = \max_{\alpha \in \Gamma_k} \left[\sum_{s \in S} \alpha(s) b(s) \right].$$

Apesar de ser possível extrair uma política não-estacionária do PBVI, ele não é uma boa escolha para problemas de horizonte finito. Como o conjunto de pontos de crença é aumentado a cada iteração do algoritmo, os passos iniciais geram uma política para poucos pontos de crença. A baixa resolução da grade de pontos de crença leva consequentemente a uma política de má qualidade.

HSVI

O HSVI de Smith e Simmons [75, 74] é outro método baseado em grade de pontos de crença que funciona especificamente para problemas de horizonte infinito.

Normalmente os algoritmos de programação dinâmica para POMDPs armazenam, no caso de horizonte infinito, um limite inferior para a recompensa esperada (representado pelo conjunto Γ de vetores). Este limite é representado pela função valor exata do caso finito, que converge para a do caso infinito à medida que vetores são adicionados ao conjunto.

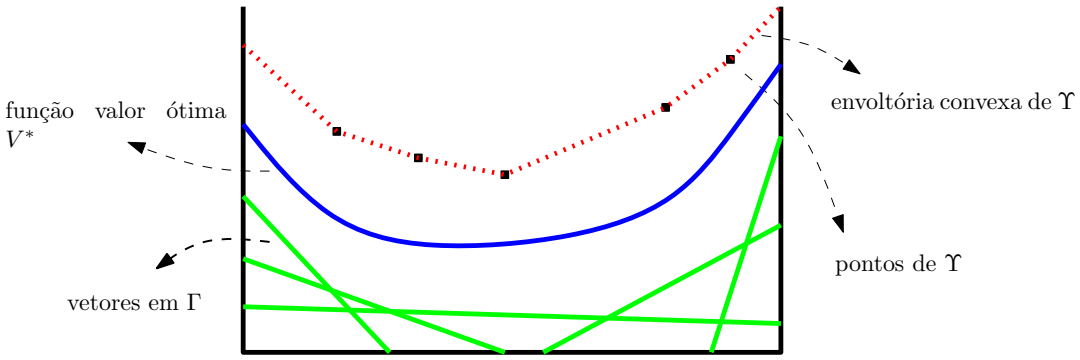


Figura 3.13: Limites superior e inferior da função valor no algoritmo HSVI.

O HSVI representa o limite mínimo da recompensa esperada desta forma, mas mantém também um limite máximo para a recompensa esperada. Este limite máximo é representado por um conjunto Υ de valores para pontos de crença, como mostra a Figura 3.13. A envoltória convexa desse conjunto de pontos dá o limite máximo da função valor.

Para inicializar os dois limites da função valor, o HSVI faz o seguinte:

- Γ_0 é inicializado com a "*política cega*": seja π_a a política que sempre seleciona a ação a. Um limite mínimo para a recompensa $\underline{R}_a$ pode ser calculado como a recompensa desta política.

$$\underline{R}_a = \sum_{t=0}^{\infty} \gamma^t \min_{s \in S} R(s,a) = \frac{\min_{s \in S} R(s,a)}{1-\gamma},$$

$$\underline{R} = \max_{a \in A} \underline{R}_a.$$

- Υ_0 é inicializado com a solução do MDP subjacente. Isso dá uma política otimista que sempre considera o simplex dos estados de crença em seus cantos.

A cada passo, o HSVI reduz a distância entre os dois limites, incluindo mais pontos em Υ e mais vetores em Γ, como mostra a Figura 3.14.

O operador H é usado para atualizar o conjunto Υ de pontos:

$$Q^V(b,a) = \sum_{s \in S} R(s,a)b(s) + \gamma \sum_{o \in \Omega} Pr(o|b,a)V(\tau(b,a,o)),$$

$$HV(b) = \max_{a \in A} Q^V(b,a).$$

E o operador β é usado para o conjunto Γ de vetores:

$$\beta_{a,o} = \arg\max_{\alpha \in \Gamma_V} \alpha\tau(b,a,o),$$

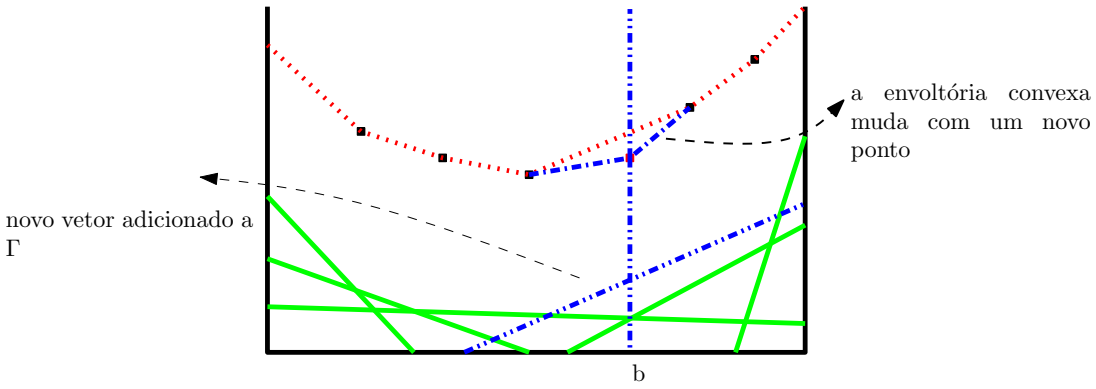


Figura 3.14: Passo do algoritmo HSVI.

$$\beta_a(s) = R(s,a) + \gamma \sum_{o \in \Omega, s' \in S} \beta_{a,o}(s') O(o|s',a) T(s'|s,a),$$

$$\beta(\Gamma, b) = \arg\max_{\beta_a} \beta_a b.$$

E finalmente, os dois operadores usados pelo HSVI para modificar os limites superior e inferior são:

- L_b, um operador que modifica o limite inferior $\underline{\Gamma}$;
- U_b, um operador que modifica o limite superior $\overline{\Upsilon}$.

Para cada ponto de crença b, estes operadores são definidos da seguinte forma:

$$L_b\Gamma = \Gamma \cup \beta(\Gamma, b),$$

$$U_b\Upsilon = \Upsilon \cup \{(b, H\Upsilon(b))\}.$$

Um número de passos de melhoria da aproximação da função valor é intercalado com um passo de expansão de pontos de crença, da mesma forma que para o PBVI. O HSVI não funciona para horizonte finito, porque o limite inferior é inicializado com uma política cega para horizonte infinito, e esta política não é útil para o caso de horizonte finito; além de haver a expansão da grade de pontos, como no PBVI.

3.4.4 Decomposição

Uma vez que a solução de POMDPs de maneira exata é um problema intratável, métodos de fatoração de POMDPs são especialmente importantes, já que permitem que pequenas partes de um POMDP sejam resolvidas rapidamente. O algoritmo apresentado daqui é a versão do PolCa para POMDPs, que é chamada de Polca+.

Polca+

O algoritmo PolCa+ divide as ações do POMDP original de forma a agrupar algumas delas em subtarefas, da mesma forma que o algoritmo PolCa para MDPs, descrito na página 17.

As subtarefas hierárquicas h são similares às subtarefas no algoritmo PolCa, e consistem de:

- S_h: um conjunto de clusters de estados. Os estados do POMDP original são agrupados em clusters;
- A_h: ações (tanto as primitivas como as abstratas);
- Ω_h: o conjunto de observações possíveis dentro da subtarefa h;
- $\overline{R}_h : S_h \times A_h \mapsto \mathbb{R}$: uma função local de recompensa.

Depois da decomposição do POMDP em subtarefas, os parâmetros das subtarefas são definidos pelo Algoritmo 3.9.

```
1 Reformule o espaço de estados H · S
2 para cada tarefa h ∈ H faça
3 |   Defina parâmetros T e S
4 |   Minimize estados
5 |   Minimize observações
6 |   Determine solução para h
7 fim
```

Algoritmo 3.9: Algoritmo PolCa+ para decomposição de POMDP em subtarefas.

Os passos do algoritmo são detalhados a seguir.

Reformulação do espaço de estados Este passo é idêntico à reformulação de estados no algoritmo Polca para MDPs, descrita na página 17.

Definição de T e R As funções T e R são definidas da mesma forma que no algoritmo Polca para MDPs (como descrito na página 18).

A função de probabilidade de observação é definida para as subtarefas para cada ação a:

Se a é uma ação primitiva,

$$O(o|h \cdot s, a_k) = O(o|s, a_k),$$

Se $\overline{a}_p$ é uma ação abstrata que contém a subtarefa h_p, então

$$O(o|h \cdot s, \overline{a}_p) = O(o|s, \pi^*_{h_p}(s)),$$

onde $\pi^*_{h_p}(s)$ é a política ótima para a subtarefa h_p no estado de crença que dá a probabilidade 1 para o estado s, e

$$b(r) = \begin{cases} 1 & \text{se } r = s \\ 0 & \text{se } r \neq s. \end{cases}$$

Isto faz da decomposição uma aproximação, já que a política ótima para a subtarefa h_p apenas é considerada nos extremos de seu espaço de crença, ignorando a possibilidade de alta incerteza (a modelagem leva em consideração a política para cada *estado do POMDP*, quando na verdade a política pode variar de acordo com o *estado de crença*).

Minimização de estados Os estados são divididos em clusters: Seja $z_h(s_i) = z_h(s_j)$ se

$$R(h \cdot s_i, a) = R(h \cdot s_j, a),\ \forall a \in A_h.$$

Em seguida, verifica-se a estabilidade de cada cluster c. Para todos os pares de estados originais s_i e s_j do cluster c, e cada ação de A_h, a soma das probabilidades de o sistema sair de s_i com a para outro cluster deve ser a mesma se o sistema sair de s_j. Ou seja, $c \in S_h$ é estável se e somente se

$$\begin{aligned} &\sum_{s' \in c'} T(h \cdot s'|h \cdot s_i, a) O(o|h \cdot s, a) \\ &= \sum_{s' \in c'} T(h \cdot s'|h \cdot s_j, a) O(o|h \cdot s, a), \\ &\forall (s_i, s_j) \in c,\ \forall c' \in S_h, \forall a \in A_h, \forall o \in \Omega. \end{aligned}$$

A noção de estabilidade de um cluster no algoritmo PolCa+ extende aquela do algoritmo PolCa, porque inclui as probabilidades de observação. Se algum cluster não é estável, ele é dividido ($c \rightarrow \{c_k, c_{k+1}, \ldots\}$), da mesma forma que no algoritmo PolCa.

Minimização de observações Este passo automaticamente determina o subconjunto de observações relevantes na subtarefa h, para cada ação $a \in A_h$. O efeito da minimização das observações é particularmente importante, uma vez que algoritmos exatos para solução de POMDPs são exponenciais no número de observações. Para cada subtarefa h, são determinados os clusters de observações $\Omega^a_h = \{\omega_0, \omega_1, \cdots\}$ primeiro colocando

cada observação em um cluster. Depois, os clusters cujas observações oferecem a mesma informação são agrupados de forma gulosa:

$$\exists k \text{ tal que } \sum_{s \in c} O(o_i|h \cdot s, a) = k \sum_{s \in c} O(o_j|h \cdot s, a), o_i \in \omega, o_j \in \omega', \forall a \in A_h$$

até que não haja mais um par de clusters ω e ω' que satisfaça estes critérios.

Redefinição do POMDP para clusters Neste ponto, o POMDP é redefinido em termos de clusters:

$$b_0(c) = \sum_{s \in c} b_0(s), \forall c \in S_h, \tag{3.5}$$

$$T(c'|c, a) = \sum_{s' \in c} T(h \cdot s'|h \cdot s, a), s \in c, \forall (c, c') \in S_h, \forall a \in A_h, \tag{3.6}$$

$$O(k|c, a) = \sum_{s' \in c} \sum_{o \in k} O(o|h \cdot s', a), \forall \omega \in \Omega_h^a, \forall c \in S_h, \forall a \in A_h, \tag{3.7}$$

$$R(c, a) = R(h \cdot s, a), s \in c, \forall c \in S_h, \forall a \in A_h. \tag{3.8}$$

A Equação 3.5 determina como o estado de crença inicial para um cluster de estados deve ser definido antes da política ótima ser encontrada. Os estados de crença para cada um dos estados s do cluster c são somados, e o resultado é o estado de crença para o cluster como um todo. A Equação 3.6 determina a função de transição: para cada par (c, c') de clusters, a probabilidade de uma ação a levar de c até c' é a soma das probabilidades de estados individuais dentro de c levarem a algum estado em c'. A Equação 3.7 determina a função de probabilidade para observações, da mesma forma que para as probabilidades de transição. A Equação 3.8 mostra a recompensa para (c, a) (que é a mesma para todos os estados dentro de um cluster).

Solução das subtarefas Ao contrário do algoritmo PolCa, que calcula a função valor quando define os clusters de estados, o algoritmo PolCa+ espera até que os clusters de estados e observações tenham sido definidos para resolver as subtarefas.

A política ótima para uma subtarefa h é determinada da mesma forma que para um POMDP comum. Pineau e Thrun usaram algoritmos exatos para solução de subtarefas (uma vez que em seus experimentos, as subtarefas eram todas pequenas).

Execução da política hierárquica

Quando à execução da política, o algoritmo PolCa+ para POMDPs é muito parecido com o PolCa. A única diferença é que para executar a política recursiva para POMDPs é

necessário calcular o novo estado de crença a cada passo. O Algoritmo 3.10 mostra como a política hierárquica é usada para determinar a próxima ação. Não é necessário manter estados de crença locais, já que estes podem ser calculados a partir do estado de crença global:

$$b_k^h(c) = \sum_{s \in c} b_k(s), \quad \forall c \in S_h.$$

O algoritmo determina a subtarefa corrente a cada nova iteração, não ficando "preso" em uma subtarefa entre iterações. O laço mostrado no algoritmo aprofunda-se na hierarquia até encontrar uma política local que recomende uma ação primitiva.

Entrada: h_k, a tarefa atual no tempo k, e b_k, o estado de crença global no tempo k.
Saída: a_k, a ação a ser executada no tempo k.

1 $b_k^h(c) = \sum_{s \in c} b_k(s), \quad \forall c \in S_h$
2 $a_k = \pi_h(b_k^h)$
3 **enquanto** a_k *é abstrata (ou seja,* $\overline{a}_k$*)* **faça**
4 Seja h_{sub} a tarefa indicada por $\overline{a}_k$
5 $b_k^h(c) \leftarrow \sum_{s \in c} b_k(s), \forall c \in S_h$
6 $a_k \leftarrow \pi_h(b_k^h)$
7 **fim**
8 Retorne a_k

Algoritmo 3.10: Algoritmo PolCa+ para execução de política.

Outras formas de fatorar e decompor POMDPs

Há outras formas de fatorar e decompor POMDPs. Boutilier e Poole [17] usam redes Bayesianas para representar POMDPs, de forma a representar a função de transição de forma compacta, e agregar estados que tenham a mesma soma total esperada de recompensa. A técnica tem sido refinada, de forma a diminuir ainda mais o tamanho da representação [27, 30]. Theocarous e Kaelbling [82] propuseram o uso de macro-ações com métodos de Monte Carlo para a solução de POMDPs genéricos. As técnicas de compressão *value-directed* desenvolvida por Poupart [65] e E-PCA de Roy e Gordon [71] permitem representar de forma compacta o estado de crença e o conjunto de estados. Todas estas técnicas estão, no entanto, fora do escopo deste trabalho.

3.4.5 Outros algoritmos

Há uma grande quantidade de otimizações para os algoritmos já mencionados que não incluímos neste trabalho. Uma visão mais ampla dos métodos para solução de POMDPs

pode ser obtida nos trabalhos de Hauskrecht [39], Murphy [53] e Cassandra [19]. Há também outros trabalhos, como algoritmos paralelos [70]; outros algoritmos, como as variantes do RTDP [15, 76], o Pegasus [54]; e variantes de POMDPs, como os POMDPs descentralizados [29, 13], por exemplo.

3.5 Processos de Decisão Semi-Markovianos Parcialmente Observáveis

Um processo de decisão semi-Markoviano parcialmente observável (POSMDP) é uma extensão da idéia de SMDPs, incorporando um modelo de observações.

3.5.1 POSMDPs

Formalmente, um POSMDP pode ser definido como uma tupla $(S, A, T, R, F, \Omega, O)$, onde os primeiros elementos são definidos como em um SMDP, e os dois últimos são um conjunto de observações e uma função de probabilidade de observação, como em um POMDP.

Em um SMDP, o estado corrente é completamente observável e não há manutenção de estado de crença. Em POSMDPs, o tempo desde a última época de decisão é usado na determinação do novo estado de crença em cada época de decisão. Se o estado de crença anterior era b, a ação executada foi a, a observação obtida foi o e o tempo decorrido foi t, o novo estado de crença $b^o_{a,t}$ é:

$$b^o_{a,t}(s') = \frac{O(o|s,a)\sum_{s\in S} b(s)Q(t,s|s,a)}{Pr(o,t|b,a)},$$

onde $Pr(o,t|b,a)$ é a probabilidade de a observação ser o e a próxima época de decisão acontecer no tempo t, sendo que o estado de crença anterior era b e a ação a foi executada:

$$Pr(o,t|b,a) = \sum_{s'\in S}\left[O(o|s',a)\sum_{s\in S} b(s)Q(t,s'|s,a)\right].$$

3.5.2 Política para um POSMDP

A política para um POSMDP é semelhante à política para um POMDP. O tempo decorrido desde a última época de decisão foi usado no cálculo do novo estado de crença, e portanto não precisa ser incluído como parâmetro da política.

3.5.3 Algoritmos

Um algoritmo de iteração de valores para POSMDPs teria que usar um operador

$$HV(b) = \max_{a \in A} \left[\rho(b,a) + \sum_{o \in \Omega} \sum_{t=0}^{\infty} e^{-\alpha t} Pr(o,t|b,a) V(b^o_{a,t}) \right].$$

No entanto, é impossível enumerar todos os valores $V(b^o_{a,t})$, para todo t que cada época de decisão pode ter levado para terminar, e este operador não pode ser calculado de forma exata para o caso geral.

Há alguns trabalhos sobre heurísticas e algoritmos aproximados para POSMDPs: Hansen [32] usou algoritmos para solução de POMDPs para a aproximação de políticas para POSMDPs. Mahadevan usou a política ótima do SMDP subjacente, usando uma heurística que assume que o processo está no estado mais provável (consultando-se o estado de crença), e a utilizou com sucesso em um controlador para um robô [48]. No entanto, o próprio Mahadevan comenta que esta heurística só funciona quando não há ações cujo principal objetivo é coletar mais informação e tornar o estado de crença mais preciso. Yu [86] propôs algoritmos aproximados para POSMDPs usando o critério de recompensa média. Além destes trabalhos, White [83] desenvolveu um algoritmo para a solução de POSMDPs com tempo discreto e horizonte finito (estas duas restrições permitem enumerar as possíveis durações entre épocas de decisão).

3.6 Sumário

Este capítulo apresentou os processos de Markov parcialmente observáveis e algoritmos para solução destes. Os algoritmos apresentados são mostrados na Tabela 3.2. A coluna "Exato" diz se o algoritmo obtém a solução exata para horizonte finito. As referências na tabela não indicam necessariamente os autores originais dos algoritmos, e sim para textos com descrições claras dos algoritmos.

POMDPs são descritos extensamente e com bastante clareza por Hauskrecht [36], e também por Kaelbling, Littman e Cassandra [44]. Algoritmos aproximados para POMDPs também são descritos por Aberdeen [2].

Algoritmo	Horizonte	Exato	Referência
Iteração de valores	F/I	S	Hauskrecht [36]
Grade finita (básico)	F/I	N	Pineau [60], Cassandra [19]
Polca+	F/I	N	Pineau [60]
Iteração de políticas (Hansen)	I	N	Hansen [34]
BPI	I	N	Poupart [65]
RTDP	I	N	Bonet e Geffner [8]
PBVI	I	N	Pineau [60]
HSVI	I	N	Smith e Simmons [75]

Tabela 3.2: Algoritmos para solução de POMDPs.

Capítulo 4

Processos de Decisão de Markov Limitados por Linguagem

Este capítulo apresenta uma nova forma de descrever processos de Markov e algoritmos que limitam a busca por políticas ótimas àquelas que respeitem certas restrições [58]. Estas restrições tem dupla utilidade: primeiro, permitem que ações obviamente desvantajosas para o agente não sejam consideradas durante o planejamento (por exemplo, um robô não deve considerar girar em torno de seu eixo mais que duas vezes em ângulo de 90^o); segundo, há situações onde podemos querer modelar tais restrições, por causa da própria natureza do problema.

4.1 Restrições para ações e observações

Em muitas situações, pode ser que determinada ação só deva ser executada apenas n vezes, ou que só possa ser executada depois de outra ação (ou depois de uma determinada observação). Por exemplo:

- **Modelagem de recursos limitados**: podemos imaginar que um agente (um robô, ou uma nave em um jogo de computador) pode executar uma ação (como andar ou atirar) por um certo número de vezes apenas. Isso pode ser descrito por uma linguagem regular;

- **Ações obrigatórias**: em algumas situações, pode ser necessário incluir forçosamente uma ação em um dado momento. Um determinado procedimento médico pode ser permitido apenas depois de outro, ainda que ele possa ser dispensável, do ponto de vista matemático;

- **Testes não repetíveis**: se a repetição de um teste não muda o estado de crença, o teste deve ser realizado apenas uma vez. Um exemplo disto é o de um robô que tem uma câmera fotográfica, usada para determinar sua posição. A câmera passa ao robô uma imagem para que o robô ajuste sua crença – mas não faz sentido que o robô tire outra foto sem antes mover a câmera (ou mover a si mesmo), porque de outra forma a imagem que receberá é a mesma, a menos de eventos externos. Durante o diagnóstico de doenças médicas, alguns testes também não devem ser repetidos. Por exemplo, perguntar dez vezes a mesma coisa ao paciente, só porque o custo da pergunta é baixo, não aumentará a certeza do médico a respeito de coisa alguma (quando muito, diminuirá a confiança do médico no sistema de apoio à decisão que venha a recomendar as perguntas repetidas). Assim, alguns testes deveriam ser repetidos apenas após ações que modifiquem o estado do sistema;

- **Ações e observações seletoras**: pode ser que queiramos que uma ação ou observação funcione como um "seletor". Uma ação ou observação seletora diz ao agente que a partir daquele momento, certas ações passam a ser permitidas. Um bom exemplo é a de diagnóstico e tratamento médico: determinada ação só pode ser executada após outra, ainda que seja mais vantajoso, do ponto de vista matemático, executar a segunda apenas. Outro exemplo é o do robô mencionado no item anterior. Se a imagem enviada da câmera for claramente incompatível com o esperado (todos os bits iguais a zero, por exemplo), a câmera está quebrada – e a partir deste momento não deveria mais ser usada;

- **Otimizações específicas em problemas**: por exemplo, um robô pode executar três ações: andar para a frente, virar noventa graus à esquerda e virar noventa graus à direita (em torno do próprio eixo). Se quisermos obter uma política de navegação para este robô, não faz sentido, durante o cálculo da política, que consideremos sequências de ações que claramente não são boas, como por exemplo fazer o robô girar em torno de seu eixo para o mesmo lado mais do que duas vezes.

Uma maneira de definir restrições na ordem de execução de ações é descrevendo tal ordem como uma linguagem. Agregamos a um MDP uma linguagem regular cujas sentenças representam todas as sequências possíveis de ações e observações que queremos permitir ao modelar o problema. Por exemplo, se quisermos que a ação "atirar" (a) possa ser executada apenas três vezes antes de cada ação "recarregar" (r), podemos usar a linguagem $((\varepsilon|a|aa|aaa)r^+)^*$.

As definições a seguir serão usadas neste trabalho.

Definição 4.1 (Evento em um MDP)
No contexto de MDPs, um evento é uma ação, o estado resultante após a ação ser executada, ou ambos (a, s' ou (a, s')). No desenvolvimento dos algoritmos, o alfabeto dos

autômatos será composto de todos os eventos possíveis.

Definição 4.2 (Autômato)
Neste trabalho, um autômato é definido como uma tupla $(\Sigma, E, F, I, \delta)$ *onde* Σ *é o alfabeto,* E *é um conjunto de estados,* F *é o conjunto de estados finais,* $\delta : E \times \Sigma \mapsto E$ *é a função de transição, e* I *é o conjunto de estados iniciais. O autômato pode ter mais de um estado inicial (o que é equivalente a dizer que o autômato pode ser não-determinístico). Quando apenas um estado inicial for relevante (ou quando houver apenas um estado inicial) a notação usada será a de elemento, e não de conjunto:* $(\Sigma, E, F, q_0, \delta)$*. Quando quisermos enfatizar o fato de um autômato* $\mathcal{M}$ *ser determinístico, usaremos um asterisco como índice* $(\mathcal{M}_*)$*.*

Definição 4.3 (Autômato reverso)
Dado um autômato finito $\mathcal{M} = (\Sigma, E, F, I, \delta)$*, o autômato reverso de* $\mathcal{M}$*, denotado* $\mathcal{M}^R$ *é aquele que tem suas transições invertidas, aceitando a linguagem reversa de* $L(\mathcal{M})$*. Formalmente,* $\mathcal{M}^R = (\Sigma, E, I, F, \delta')$ *(note a troca dos estados iniciais com os finais). Para quaisquer* $s, s' \in E, t \in \Sigma$*, se* $(s, t, s') \in \delta$ *então* $(s', t, s) \in \delta'$*, e se* $\mathcal{M}$ *tinha mais de um estado final,* $\mathcal{M}^R$ *terá mais de um estado inicial (e portanto será não-determinístico);*

Definição 4.4 (Entradas e saídas de um estado)
Seja um autômato $\mathcal{M} = (\Sigma, Q, F, s, \delta)$*. A função* $IN : E \mapsto P(E)$ *(onde* $P(X)$ *é o conjunto das partes de* X*) dá os estados de entrada de um estado (incluindo o próprio estado, se houver um laço):* $IN(q) = \{q' \in E | \exists e \in \Sigma, \delta(q', e) = q\}$*. Da mesma forma,* $OUT(q) = \{q' \in E | \exists e \in \Sigma, \delta(q, e) = q'\}$

As seções a seguir definem LL-MDPs e apresentam algoritmos para solução destes.

4.2 LL-MDPs

Um LL-MDP é definido como um MDP mais um autômato, cujo alfabeto é o conjunto de possíveis eventos do MDP. A linguagem do autômato consiste de todas as sequências de eventos que se quer permitir como solução para o LL-MDP. Por exemplo, a Figura 4.1 mostra um autômato que define as seguintes restrições:

- Se o primeiro evento for a, o último deve ser c;
- Se o primeiro evento for diferente de a, deve haver apenas dois eventos, e o último deve ser ou a ou b.

Na figura, A é o conjunto de todos os eventos possíveis. Uma solução para um LL-MDP que tenha este autômato como componente deve necessariamente obedecer estas restrições.

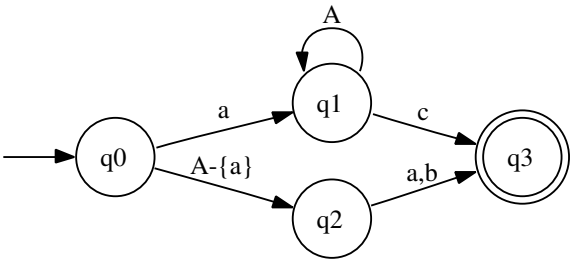


Figura 4.1: Autômato que restringe a ordem das ações em um MDP.

Definição 4.5 (LL-MDP)
Um processo de decisão de Markov limitado por linguagem (LL-MDP) é uma tupla $(S, A, T, R, \mathcal{M})$*, onde* $\mathcal{M} = (\Sigma, Q, F, s, \delta)$ *é um autômato finito que define em que ordem as ações podem ser executadas.*

Definição 4.6 (MDP subjacente a um LL-MDP)
Dado um LL-MDP $(S, A, T, R, \mathcal{M})$*, o* MDP subjacente *é* (S, A, T, R)*.*

Uma vez que os eventos (o alfabeto do autômato) incluem o estado em que o sistema termina após uma ação, é necessário ter certeza de que a modelagem não deixará de lado situações que podem ocorrer. Por exemplo, o autômato poderia determinar que a partir de um estado q os eventos possíveis são apenas $< a_1, s_2 >$ e $< a_2, s_3 >$. No entanto, se $\exists s_4 \in S,\ T(s_5|s_4, a_1) > 0$, há um problema: quando a ação a_1 for executada, é possível que o próximo estado seja s_5. Assim, sempre que houver um evento $< a, s >$ em uma transição saindo de um estado q do autômato, devem haver também transições saindo deste estado para todos os eventos $< a, s' >$ tais que $\exists s'',\ T(s'|s'', a) > 0$.

4.2.1 Política para um LL-MDP

A política para um LL-MDP deve mapear estados do autômato e estados do MDP em ações Assim, uma política para LL-MDP tem como componente um autômato.

Definição 4.7 (Política para um LL-MDP)
Dado um LL-MDP $L = (S, A, T, R, \mathcal{M})$*, uma política* $\mathcal{P}$ *para* L *é um par* $(\mathcal{W}, \dot{\pi})$*, onde*

- $\mathcal{W} = (\Sigma, \mathcal{E}, \mathcal{F}, \mathcal{N}, \Delta)$ *é um autômato com alfabeto* Σ*, estados* $\mathcal{E}$*, estados finais* $\mathcal{F}$*, estados iniciais* $\mathcal{N}$*, e função de transição* Δ*;*

- $\dot{\pi} = \{d_0, d_1, \cdots, d_{|\mathcal{E}|-1}\}$ *é um conjunto de regras de decisão, onde para cada estado* $q \in \mathcal{E}$ *há uma regra* $d_q : S \mapsto A$ *mapeando estados do LL-MDP em ações.*

Não há menção a épocas de decisão na definição de política para LL-MDPs, porque o autômato $\mathcal{W}$ usado na política é diferente do autômato original $\mathcal{M}$ do LL-MDP: para cada estado do autômato original $\mathcal{M}$ e cada época de decisão k, há um estado em $\mathcal{W}$. Assim, as diferentes épocas de decisão são representadas na política por diferentes estados do autômato.

Neste trabalho, políticas e funções valor para processos de Markov limitados por linguagem são denotadas por $\dot{\pi}$ e $\dot{V}$. No entanto, quando houver uma referência à função valor *em um estado do autômato*, a notação será a mesma que para funções valor de processos de decisão de Markov comuns (ou seja, V_q é uma das funções valor que compõem $\dot{V}$). O espaço de políticas e espaço de funções valor para um LL-MDP L são denotados por $\dot{\Pi}(L)$ e $\dot{\mathcal{V}}(L)$ (ou simplesmente $\dot{\Pi}$ e $\dot{\mathcal{V}}$ quando não houver ambiguidade).

Para horizonte infinito, $\mathcal{W} = \mathcal{M}$, já que as funções valor em cada estado do autômato serão as mesmas a cada época de decisão.

Para horizonte finito, o autômato $\mathcal{W}$ consiste de:

- Σ, o mesmo alfabeto de $\mathcal{M}$;
- $\mathcal{E}$, os estados. Para cada par de época de decisão e estado de $\mathcal{M}$ (por exemplo, k e q_m) considerado no algoritmo de iteração de valores, um estado $q'_{[k,m]}$ é incluído em $\mathcal{E}$;
- s, o estado inicial de $\mathcal{M}$ na primeira época de decisão;
- $\mathcal{F}$, os estados finais. Todos os estados em $\mathcal{E}$ que estejam na última época de decisão;
- Δ: a função de transição deve garantir que apenas as sequências permitidas por $\mathcal{M}$ aconteçam. Assim, para cada par $(q'_{[k,m]},\ q'_{[k+1,n]})$ de estados de $\mathcal{E}$, se $\delta(q_m, (a, s)) = q_n$, então $\Delta(q'_{[k,m]}) = q'_{[k+1,n]}$.

Os estados não alcançáveis a partir de s são descartados.

Definição 4.8 (Função valor para um LL-MDP)
Dado um LL-MDP $L = (S, A, T, R, \mathcal{M})$ e uma política $\mathcal{P} = (\mathcal{W}, \dot{\pi})$ para L, uma função valor $\dot{V}$ para $\mathcal{P}$ é o dada pelo conjunto $\dot{V} = \{V_0, V_1, \cdots, V_{|\mathcal{E}|-1}\}$ de funções valor, onde $V_q(s)$ é o valor esperado para a política $\mathcal{P}$ nos estados q e s de acordo com o critério de otimalidade escolhido.

A função valor para LL-MDP pode ser vista como uma matriz $|\mathcal{E}| \times |S|$ de números reais, tal que o elemento na posição q, s é o valor, na época de decisão k, para o estado q do autômato e s do MDP. Desta forma, é possível perceber que o espaço $\dot{\mathcal{V}}$ de funções valor para um LL-MDP é um espaço linear, da mesma forma que o espaço $\mathcal{V}$ de funções valor para MDPs.

Durante a execução da política é necessário guardar o estado corrente do autômato $\mathcal{W}$. O Algoritmo 4.1 deve ser usado para a execução de políticas para horizonte finito (o algoritmo usa mapeamento em ações, mas pode ser modificado para usar mapeamento em valores). O mesmo algoritmo pode ser trivialmente adaptado para horizonte infinito.

Entrada: $\mathcal{P} = (\mathcal{W}, \dot{\pi})$, uma política para LL-MDP, onde $\mathcal{W} = (\Sigma, \mathcal{E}, \mathcal{F}, \mathcal{N}, \Delta)$ e $\dot{\pi} : \mathcal{E} \times S \mapsto A$;
s, estado inicial do LL-MDP.

```
1 k ← 0
2 Escolha q ∈ N
3 enquanto k < h faça
4 |   a ← π̇([k, q], s)
5 |   Execute a
6 |   Verifique novo estado s
7 |   q ← Δ(q, < a, s >)
8 |   k ← k + 1
9 fim
```

Algoritmo 4.1: Execução de política para LL-MDP de horizonte finito.

Se uma política permite a execução de ações proibidas pelo autômato de um LL-MDP, ela não é admissível. Para formalizar a noção de política admissível, definimos o conjunto A_q de ações possíveis a partir de cada estado q do autômato $\mathcal{M}$:

$$A_q = \{a | \exists p \in E,\ \delta(q, < a, \cdot >) = p\}.$$

É possível definir este conjunto também para o autômato $\mathcal{W}$:

$$\begin{aligned} A_{[k,q]} &= \{a | \exists p \in \mathcal{E},\ \Delta([k, q], < a, \cdot >) = p\} \\ &= A_q. \end{aligned}$$

Como $A_{[k,q]} = A_q$ para todo k, usaremos apenas a notação A_q.

Definição 4.9 (Política admissível para um LL-MDP)
Seja $L = (S, A, T, R, \mathcal{M})$ um LL-MDP e $\mathcal{P} = (\mathcal{W}, \dot{\pi})$ uma política para L, tal que $\dot{\pi} = \{d_0, d_1, \cdots, d_{|\mathcal{E}|-1}\}$. $\mathcal{P}$ é admissível para L se:

- *A linguagem de $\mathcal{W}$ está contida na linguagem de $\mathcal{M}$;*
- *$\forall s \in S, \forall q \in \mathcal{E},\ d_q(s) \in A_q$, onde d_q é a regra de decisão para o estado s de $\mathcal{E}$ e o estado q de $\mathcal{M}$.*

A definição de política admissível vale tanto para horizonte infinito, onde $\mathcal{M} = \mathcal{W}$, como para horizonte finito. No caso de horizonte finito, tendo em mente o mapeamento dos estados de $\mathcal{W}$ nos estados de $\mathcal{M}$, suponha que um estado de $\mathcal{E}$ seja $q = [k, p]$. Então dizer que $d_q(s) \in A_q$ é o mesmo que dizer que $d_{[k,p]}(s) \in A_{[k,p]} = A_p$.

O conceito de admissibilidade pode ser extendido para funções valor. Uma função valor é admissível se $\forall s \in S, \forall q \in \mathcal{E},\ \arg\max_{a \in A} \dot{V}_q(s) \in A_q$ (além, é claro, de a linguagem de $\mathcal{W}$ estar contida na de $\mathcal{M}$).

O valor $Q^\pi(q, s, a)$ de uma ação em um estado é definido apenas quando $a \in A_q$, como

$$Q^\pi(q, s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s'|s, a) V^\pi_{\Delta(q, <a, s'>)}(s').$$

A política ótima para LL-MDPs é definida da mesma forma que para MDPs.

Definição 4.10 (Política ótima para um LL-MDP)
Seja $L = (S, A, T, R, \mathcal{M})$ um LL-MDP. Uma política $\mathcal{P} = (\mathcal{W}, \dot{\pi})$ para L é ótima para L se:

- *$\mathcal{P}$ é admissível para L;*
- *As ações recomendadas por $\dot{\pi}$ maximizam o critério de otimalidade:*

$$\forall \dot{\pi}' \in \dot{\Pi} \text{ tal que } \dot{\pi}' \text{ é admissível para } L,$$

$$\forall q \in \mathcal{E}, \forall s \in S,\ \ Q^{\dot{\pi}}(q, s, d_q(s)) \geq Q^{\dot{\pi}'}(q, s, d'_q(s)),$$

onde d_q é a regra de decisão de $\dot{\pi}$ para o estado q do autômato $\mathcal{W}$, e d'_q é a regra de decisão de $\dot{\pi}'$ para o estado q.

Definição 4.11 (Função valor ótima para um LL-MDP)
Uma função valor $\dot{V}$ é ótima para um LL-MDP $L = (S, A, T, R, \mathcal{M})$ se:

- *$\dot{V}$ é admissível para L;*
- *$\forall \dot{U} \in \dot{\mathcal{V}}$ tal que $\dot{U}$ é admissível para L, $\forall q \in \mathcal{E}, s \in S,\ \ V_q(s) \geq U_q(s)$.*

4.2.2 Recompensa para um LL-MDP

O espaço de busca de políticas admissíveis para um LL-MDP é menor que o do MDP subjacente, já que o número de ações possíveis em cada época de decisão pode ser menor para o LL-MDP. Como as ações possíveis em um LL-MDP são restritas, a recompensa esperada pode ser menor que a do MDP subjacente.

Seja um LL-MDP L e seu MDP subjacente M, com políticas $\dot{\pi}$ e π, e funções valor $\dot{V}$ e V. Supondo que em uma dada época de decisão o estado do autômato do LL-MDP seja

q, o conjunto de ações disponível para $\dot{\pi}$ é A_q, e para π o conjunto é A. Como $A_q \subseteq A$, para cada ação a tal que $a \in A$ e $a \notin A_q$, deve ser verdade que: ou a oferece melhor recompensa esperada do que todas as ações em A_q em algum estado s (e neste caso é possível que $\dot{V} < V$), ou não (e neste caso a não seria parte da solução ótima para este estado e esta época de decisão – pode ser então que $\dot{V} = V$).

Do fato descrito acima decorre que a solução do MDP subjacente oferece um limite superior para a função valor do LL-MDP.

4.3 Algoritmos

Esta seção apresenta quatro maneiras de resolver LL-MDPs. Uma delas é através da multiplicação do espaço de estados, e as outras três são modificações nos algoritmos exatos já existentes para MDPs.

4.3.1 Multiplicação de estados

Esta subseção mostra como transformar um LL-MDP em um MDP de forma que as políticas ótimas de ambos sempre recomendem as mesmas ações.

Dado um LL-MDP $L = (S, A, T, R, \mathcal{M})$, onde $\mathcal{M} = (\Sigma, Q, F, s, \delta)$, é possível criar o seguinte MDP $L' = (S', A, T', R')$, onde

- $S' = (S \times E)$. Para cada $s_m \in S$ e $q \in E$, o novo conjunto de estados S' terá um estado s_m^q.
- A função de transição $T' : S' \times A \times S' \mapsto [0, 1]$ é

$$T'(s_n^q | s_m^p, a) = \begin{cases} T(s_n | s_m, a) & \text{se } (p, a, q) \in \delta \\ 1 & \text{se } (p, a, q) \notin \delta, \quad s_m = s_n \text{ e } p = q \\ 0 & \text{se } (p, a, q) \notin \delta \text{ e } (s_m \neq s_n \text{ ou } p \neq q); \end{cases}$$

- $R' : S' \times A \mapsto \mathbb{R}$ é definida tal que para toda ação a,

$$R'(s_m^p, a) = \begin{cases} R(s_m, a) & \text{se } \exists q \in E \text{ tal que } (p, a, q) \in \delta \\ -\infty & \text{caso contrário.} \end{cases}$$

O valor $-\infty$ de recompensa para ações proibidas é usado para evitar que o algoritmo considere ações proibidas no cálculo das recompensas (tendo recompensa abaixo de qualquer outra, a ação não será escolhida, e seu valor não será computado).

Complexidade

Após a multiplicação, o espaço de estados passa a ser de tamanho $|S \times E| = |S||E|$. A complexidade do algoritmo para resolver o MDP resultante é, como já visto, $\Theta(z|A||S'|^2)$. Se os termos relacionados ao LL-MDP original forem usados, teremos $\Theta(z|A||S|^2|E|^2)$. Há, no entanto, algoritmos aproximados eficientes para a solução de MDPs com complexidade assintótica de tempo menor, e algoritmos que exploram a estrutura de MDPs para minimizar o tamanho de sua representação e acelerar a busca pela solução ótima.

4.3.2 Adaptação de algoritmos existentes

Os algoritmos mencionados no Capítulo 2 podem ser modificados diretamente para resolver LL-MDPs, sem a necessidade de multiplicar estados. As modificações podem ser resumidas em dois pontos:

- Na iteração i, ao invés de calcular um valor $V_i(s)$ ou uma ação $\pi(s)$ para cada estado s o algoritmo determina um valor $V_{[i,q]}(s)$ ou uma ação $\pi_{[i,q]}(s)$ para cada estado s do sistema e cada estado q do autômato;

- Para construir $V_{[i+1,p]}(s)$ ou $\pi_{[i+1,q]}(s)$, os $V_{[i,q]}(s')$ (ou $\pi_{[i,q]}(s)$) do passo anterior são usados.

4.3.3 Iteração de valores (LLVI)

Esta subseção descreve o algoritmo LLVI (*language-limited value iteration*). O LLVI é uma variante do algoritmo de iteração de valores desenvolvida para LL-MDPs.

Como descrito no Capítulo 2, a função valor para MDPs em uma dada época de decisão é um vetor de tamanho $|S|$. A função valor para um LL-MDP terá, para cada época de decisão, um conjunto de tamanho $|S||E|$ (uma função valor de tamanho $|S|$ para cada um dos $|E|$ estados do autômato. A Figura 4.2 mostra a função valor de um LL-MDP em uma época de decisão. Para cada estado do autômato, há um vetor diferente com os valores de cada estado.

O algoritmo de iteração de valores pode ser modificado para que produza todas estas funções valor em cada iteração. Isto é feito através da modificação do mapeamento h e o operador H de melhoria da política, que passarão a depender do estado do autômato.

O mapeamento $h : S \times A \times \dot{\mathcal{V}} \times \mathcal{E} \mapsto \mathbb{R}$ é definido como

$$h(s, a, \dot{V}, q) = R(s, a) + \gamma \sum_{s' \in S} T(s'|s, a) V_{\Delta(q,<a,s'>)}(s').$$

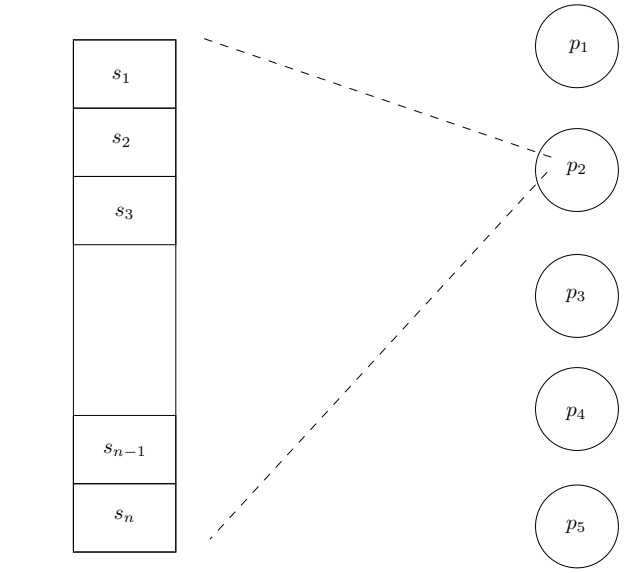


Figura 4.2: Função valor de um LL-MDP.

Para cada estado q do autômato $\mathcal{W}$, um operador de melhoria local $H : \mathcal{V} \mapsto \mathcal{V}$ modifica apenas uma das funções valor em $\dot{V}$. Para horizonte infinito $E = \mathcal{E}$, e o operador é:

$$HV_q(s) = \max_{a \in A_q} \left[h(s, a, \dot{V}, q) \right]. \tag{4.1}$$

Já para horizonte finito, usamos a notação $[k, q]$ para o estado:

$$HV_{[k,q]}(s) = \max_{a \in A_q} \left[h(s, a, \dot{V}, [k, q]) \right],$$

mas apesar da diferença na notação, o operador é exatamente o mesmo para ambos horizontes.

O operador de melhoria global de política $\dot{H} : \dot{\mathcal{V}} \mapsto \dot{\mathcal{V}}$ aplica todas as melhorias locais para cada estado do autômato:

$$\dot{H}\dot{V} = \left(HV_{q_0}, HV_{q_1}, \cdots, HV_{q_{|\mathcal{E}|-1}} \right).$$

Usando o operador $\dot{H}$, obtém-se a equação de otimalidade para LL-MDPs:

$$\begin{aligned} \forall q \in \mathcal{E}, s \in S, V_q(s) &= HV_q(s) \\ &= \max_{a \in A_q} h(s, a, \dot{V}, q) \\ &= \max_{a \in A_q} \left[R(s, a) + \gamma \sum_{s' \in S} T(s'|s, a) V_{\Delta(q, <a, s'>)}(s') \right]. \end{aligned}$$

A equação de otimalidade vale tanto para horizonte finito como para horizonte infinito (porque a informação sobre épocas de decisão está nos estados do autômato $\mathcal{W}$). Se o horizonte é finito, e o estado é $[k, p]$, as funções valor da época de decisão $k + 1$ serão selecionadas pela função de transição Δ do autômato ($\Delta([k, \cdot], \cdot)$ será necessariamente $[k + 1, \cdot]$, se existir).

O Algoritmo 4.2 mostra o passo de programação dinâmica para LL-MDPs, que também constrói gradualmente o autômato $\mathcal{W}$, evitando incluir estados que não sejam necessários.

Entrada: Um LL-MDP $(S, A, T, R, \mathcal{M})$ onde $\mathcal{M} = (\Sigma, E, F, q_0, \delta)$;
O conjunto $\mathcal{E}$ construído no passo anterior;
A função valor $\dot{V}$, parcialmente construída no passo anterior e definida para todo $[k, q] \in \mathcal{E}, k \leq i - 1$;
A relação Δ, parcialmente construída nos passos anteriores;
O número i do passo atual.
Saída: Um conjunto $\mathcal{E}$ de estados;
Uma função valor $\dot{V}$, definida para todos os estados $[k, q] \in \mathcal{E}, k \leq i$;
Uma relação $\Delta : \mathcal{E} \times \Sigma \to \mathcal{E}$.

para cada $p \in E$, *tal que* $\exists e \in \Sigma, \delta(q, e) = p$ **faça**
  $\mathcal{E} \leftarrow \mathcal{E} \cup \{[i, p]\}$
  $\Delta([i, p], e) \leftarrow [i - 1, q]$
fim
para cada $[i, p] \in \mathcal{E}$ *(onde i é a iteração sendo executada)* **faça**
  para cada $s \in S$ **faça**
    $V_{[i,p]}(s) \leftarrow \max_{a \in A_p} \left[R(s, a) + \gamma \sum_{s' \in S} T(s'|s, a) V_{\Delta([i,p],<s,a>)}(s')\right]$
  fim
fim
Devolva $\dot{V}, \mathcal{E}$ e Δ

Algoritmo 4.2: Passo de programação dinâmica para LL-MDPs.

Corretude e convergência

Esta seção demonstra a admissibilidade das políticas obtidas pelo algoritmo LLVI para horizonte infinito, além de uma garantia de convergência.

Definição 4.12 (Linguagem de um autômato a partir de um estado) *Seja $\mathcal{L}^k(X, y)$ o conjunto de palavras de tamanho k geradas pelo autômato X a partir do estado y (e não do estado inicial). A notação $\mathcal{L}^k(X)$ será usada para a linguagem de todas as palavras de tamanho k geradas a partir de estados iniciais de X. A notação $\mathcal{L}(X)$ é usada para denotar a linguagem de X a partir de seus estados iniciais (sem restrição quanto ao tamanho das palavras).*

Lema 4.1 (Admissibilidade da linguagem do autômato gerado pelo LLVI) *Sejam $L = (S, A, T, R, \mathcal{M})$ um LL-MDP e $\mathcal{P} = (\mathcal{W}, \dot{\pi})$ uma política gerada pelo algoritmo LLVI para L com horizonte z. A linguagem do autômato $\mathcal{W}$ está contida na linguagem do autômato $\mathcal{M}$.*

Prova *Para horizonte infinito, $\mathcal{M} = \mathcal{W}$, e portanto as linguagens são idênticas.*

Para horizonte finito, a prova é por indução no horizonte do LL-MDP.

A hipótese de indução é: para horizonte igual a z, $\mathcal{L}^z(\mathcal{W}, [z, q]) = \mathcal{L}^z(\mathcal{M}, q)$.

A base de indução é: para horizonte igual a zero, $\mathcal{L}^0(\mathcal{W}, [0, q]) = \mathcal{L}^0(\mathcal{M}, q) = \emptyset$).

O passo de indução segue: para horizonte i, o algoritmo calcula a solução para horizonte $i - 1$ (onde usamos a hipótese de indução) e aplica um passo de programação dinâmica. Neste passo, o autômato $\mathcal{W}$ é expandido com novos estados e transições, nas linhas 2 e 3 do Algoritmo 4.2. Seja $\mathcal{E}^{i-1}$ o conjunto de estados $[i-1, \cdot]$. Seja w o conjunto dos símbolos (eventos no LL-MDP) x tais que $\delta(p, x) = q$, e $[i-1, q] \in \mathcal{E}^{i-1}$. Então,

- $\mathcal{L}^i(\mathcal{M}, p) = w \cdot \mathcal{L}^{i-1}(\mathcal{M}, q)$;
- $\mathcal{L}^i(\mathcal{W}, [i, q]) = w \cdot \mathcal{L}^{i-1}(\mathcal{W}, [i-1, q])$.

Pela hipótese de indução, $\mathcal{L}^{i-1}(\mathcal{M}, q) = \mathcal{L}^{i-1}(\mathcal{W}, [i-1, q])$, e portanto $\mathcal{L}^i(\mathcal{M}, p) = \mathcal{L}^i(\mathcal{W}, [i, q])$.

Seja $\mathcal{E}_0$ o conjunto de estados $[i, q] \in \mathcal{E}$ tais que q é inicial e i é o número da iteração atual.

Para todo estado inicial $q_0 \in \mathcal{E}_0$, na última iteração as palavras a partir de q_0 são $\mathcal{L}^i(\mathcal{W}, [i, q_0]) = \mathcal{L}^i(\mathcal{M}, q_0)$.

Para horizonte z, a linguagem do autômato $\mathcal{W}$ tem apenas palavras de tamanho z, e está contida na linguagem de $\mathcal{M}$:

$$\mathcal{L}(\mathcal{W}) = \mathcal{L}^z(\mathcal{W}) = \bigcup_{p \in \mathcal{E}_0} \mathcal{L}^z(\mathcal{M}, p) \in \mathcal{L}(\mathcal{M}). \qquad \blacksquare$$

Teorema 4.1 (Admissibilidade da política gerada pelo LLVI) *Dado um LL-MDP L como entrada, o algoritmo LLVI gerará uma política admissível para L.*

Prova *O Lema 4.1 garante que a linguagem de $\mathcal{W}$ está contida na de $\mathcal{M}$. Da linha 7 do algoritmo, temos que $\forall s \in S, q \in E$, $\arg\max_{a \in A} V_q(s) \in A_q$.* $\blacksquare$

Quanto ao critério de convergência do algoritmo para horizonte infinito, deve-se esperar que todas as funções valor convirjam. Seja C um critério de convergência usado em

algoritmos para MDPs: $C(V, V')$ é verdadeiro se e somente se as funções valor V e V' satisfazem o critério. Então, para um LL-MDP L e iteração i:

$$\forall q \in E, \;\; C_q(L) = C(V_{[i,q]}, V_{[i-1,q]}), \tag{4.2}$$

$$C(L) = \bigwedge_{q \in E} C_q(L). \tag{4.3}$$

A convergência é garantida pelo Teorema 4.2. O Lema 4.2 pode ser trivialmente verificado.

Lema 4.2 ($\dot{\mathcal{V}}$ é um espaço de Banach)
O espaço $\dot{\mathcal{V}}$ das funções valor para LL-MDPs é um espaço de Banach.

Lema 4.3 (H para LL-MDPs é contração)
O operador de melhoria local H para LL-MDPs é uma contração.

Prova *Sejam $\dot{U}$ e $\dot{V}$ duas funções valor para um LL-MDP, e $||\cdot||$ uma norma definida em $\mathcal{V}$ tal que $||V|| = \max_{s \in S} V(s)$.*

Fixaremos os estados $q \in E$ e $s \in S$. Supondo que $HV_q(s) \geq HU_q(s)$,

$$0 \leq HV_q(s) - HU_q(s).$$

Uma das possíveis ações ótimas para os estados s e q é a_s^*:

$$a_s^* \in \arg\max_{a \in A_q} R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) V_q(s').$$

Para os estados q e s e para a ação a_s^*,

$$\begin{aligned}
HV_q(s) - HU_q(s) &= \max_{a \in A_q} \left[h(s,a,\dot{V},q) \right] - \max_{a \in A_q} \left[h(s,a,\dot{U},q) \right] \\
&= R(s,a_s^*) + \gamma \sum_{s' \in S} T(s'|s,a_s^*) V_{\delta(q,<a_s^*,s'>)}(s') - \\
&\quad \max_{a \in A_q} \left[R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) U_{\delta(q,<a,s'>)}(s') \right] \\
&\leq R(s,a_s^*) + \gamma \sum_{s' \in S} T(s'|s,a_s^*) V_{\delta(q,<a_s^*,s'>)}(s') - \\
&\quad R(s,a_s^*) + \gamma \sum_{s' \in S} T(s'|s,a_s^*) U_{\delta(q,<a_s^*,s'>)}(s') \\
&= \gamma \sum_{s' \in S} T(s'|s,a_s^*) \left[V_{\delta(q,<a_s^*,s'>)}(s') - U_{\delta(q,<a_s^*,s'>)}(s') \right] \\
&\leq \gamma \sum_{s' \in S} T(s'|s,a_s^*) ||V_q - U_q|| \\
&= \gamma ||V_q - U_q||.
\end{aligned}$$

O argumento é repetido para o caso em que $HV_q(s) < HU_q(s)$*, e portanto*

$$\forall s \in S, \;\; |HV_q(s) - HU_q(s)| \leq \gamma ||V_q - U_q||.$$

Disso segue imediatamente que

$$\max_{s \in S} |HV_q(s) - HU_q(s)| \leq \gamma \, ||V_q - U_q|| \,,$$

$$||HV_q(s) - HU_q(s)|| \leq \gamma \, ||V_q - U_q|| \,, \tag{4.4}$$

e portanto H *é uma contração.* ■

Teorema 4.2 (Convergência do LLVI para LL-MDPs)
Os valores gerados pelo algoritmo LLVI convergem para um ponto fixo.

Prova *Segue diretamente dos Lemas 4.2 e 4.3: o operador* $\dot{H}$ *é uma contração, uma vez que é composto de contrações: seja* $||\cdot||$ *uma norma definida no espaço* $\dot{\mathcal{V}}$ *tal que* $||\dot{V}|| = \max_{\substack{q \in E, \\ s \in S}} V_q(s)$*; usaremos também a norma* $||\cdot||$ *definida no Lema 4.3 para o espaço* $\mathcal{V}$*.*

$$\begin{aligned} ||\dot{H}\dot{V} - \dot{H}\dot{U}|| &= ||(\cdots, HV_q - HU_q, \cdots)|| \\ &\leq ||(\cdots, \gamma \, [V_q - U_q] \,, \cdots)|| \\ &= \gamma \, ||(\cdots, V_q - U_q, \cdots)|| \\ &= \gamma ||\dot{V} - \dot{U}||. \end{aligned}$$

■

Complexidade

O comportamento assintótico do algoritmo LLVI depende da estrutura do autômato $\mathcal{M}$, e quanto mais arestas forem consideradas nos passos de programação dinâmica, maior será o tempo de execução. Um único passo de programação dinâmica executa

$$\sum_{[i,q] \in \mathcal{E}} |A_q||S|^2$$

comparações de valores de estados. A somatória apenas considera os estados $[i, q]$ onde há a possibilidade de alguma ação ser executada.

O número de comparações para horizonte finito é

$$\sum_{i=0}^{z-1} \sum_{[i,q] \in \mathcal{E}} |A_q||S|^2,$$

e portanto o algoritmo LLVI é melhor que a multiplicação de estados seguida de iteração de valores (que tem complexidade $O(z|A||E|^2|S|^2)$), mesmo ignorando o tempo necessário para multiplicar os estados:

$$\begin{aligned}\sum_{i=0}^{z-1}\sum_{[i,q]\in\mathcal{E}}|A_q||S|^2 &\leq \sum_{i=0}^{z-1}\sum_{q\in E}|A_q||S|^2\\ &\leq z|A||E|^2|S|^2.\end{aligned}$$

No entanto, a complexidade de tempo do algoritmo de iteração de valores não leva em consideração o uso dos conjuntos A_s. Especialmente no caso de LL-MDPs multiplicados, haverá muitos estados para os quais $A_s < A$. Ainda assim, o algoritmo de iteração de valores usa todos os estados $s' \in S$ para melhorar o valor do estado s, como pode ser visto no somatório do passo de programação dinâmica:

$$V_i(s) \leftarrow \max_{a\in A_s}\left[R(s,a) + \gamma\sum_{s'\in S}T(s'|s,a)V_{i-1}(s')\right].$$

Com isto, o algoritmo VI percorre todos os $|S||E|$ estados no somatório. Já o algoritmo LLVI percorre apenas $|S||OUT(q)|$, e $|OUT(q)| \leq |E|$.

4.3.4 Iteração de Políticas (LLPI)

Esta subseção descreve o algoritmo LLPI (*language limited policy iteration*). O LLPI é uma variante do algoritmo de iteração de políticas desenvolvida para LL-MDPs.

O Algoritmo 2.2 (na página 13) para resolver MDPs usando iteração de políticas também pode ser modificado para manter funções valor para cada estado do autômato. No entanto, a política inicial deve ser admissível.

O valor da política no estado q pode ser avaliado através do seguinte sistema linear:

$$V_q^{\dot\pi}(s) = R(s,\dot\pi(s)) + \gamma\sum_{s'\in S}T(s'|s,\dot\pi(s))V^{\dot\pi}_{\delta(q,\dot\pi(s))}(s'). \tag{4.5}$$

A Equação 2.1 (na página 8) é modificada para

$$Q_q^{\dot\pi}(s,a) = R(s,a) + \gamma\sum_{s'\in S}T(s'|s,a)V^{\dot\pi}_{\delta(q,a)}(s'). \tag{4.6}$$

O passo de melhoria da política busca, para cada política $\dot\pi_{[i+1,q]}$ associada ao estado q do autômato, uma ação que pode melhorar esta política. No entanto, as ações procuradas serão sempre aquelas permitidas no estado q:

$$\forall q \in E, s \in S,\quad \dot\pi'_q(s) = \arg\max_{a\in A_q} Q_q^{\dot\pi}(s,a). \tag{4.7}$$

O princípio é o mesmo usado na alteração do algoritmo de iteração de valores. A política inicial respeita as restrições do autômato (por ter sido escolhida desta forma), e em cada passo de melhoria da política apenas as ações permitidas são consideradas.

O Algoritmo 4.3 mostra a iteração de políticas para LL-MDPs.

Entrada: Um LL-MDP $(S, A, T, R, \mathcal{M})$.
Saída: Uma política $\dot{\pi}'$ para o LL-MDP.

$\forall q \in E$, inicialize $\dot{\pi}_q$ aleatoriamente com uma política admissível
repita
  para cada $q \in E$ **faça**
    $\dot{\pi}'_q \leftarrow \dot{\pi}_q$
  fim
  Avalie a política atual resolvendo o sistema linear:
  $V_q(s) = R(s, \dot{\pi}'_q(s)) + \gamma \sum_{s' \in S} T(s'|s, \dot{\pi}'_q(s)) V_{\delta(q, \dot{\pi}'(s))}(s')$
  para cada $q \in E$ **faça**
    para cada $s \in S$ **faça**
      para cada $a \in A_q$ **faça**
        $Q_q^{\dot{\pi}'}(s, a) \leftarrow R(s, a) + \gamma \sum_{s' \in S} T(s'|s, a) V_{\delta(q,a)}(s')$
      fim
    fim
  fim
  para cada $q \in E$ **faça**
    para cada $s \in S$ **faça**
      Melhore a política:
      $\dot{\pi}_q(s) \leftarrow \arg\max_{a \in A_q} Q_q^{\dot{\pi}'}(s, a)$
    fim
  fim
até *que* $\forall q \in E, \ \dot{\pi}_q = \dot{\pi}'_q$
Devolva $\dot{\pi}$

Algoritmo 4.3: Iteração de políticas para LL-MDPs.

Corretude e convergência

As provas desta subseção usam um operador de melhoria para o algoritmo de iteração de políticas: para toda política $\dot{\pi}'$, seja $H^{\dot{\pi}'} : \mathcal{V} \mapsto \mathcal{V}$ um operador tal que

$$\forall q \in E, s \in S, \ \ H^{\dot{\pi}'} V_q^{\dot{\pi}}(s) = Q_q^{\dot{\pi}}(s, \dot{\pi}'_q(s)),$$

e $\dot{H}^{\dot{\pi}'} : \dot{\mathcal{V}} \mapsto \dot{\mathcal{V}}$ um operador tal que

$$\begin{aligned}\dot{H}^{\dot{\pi}'}\dot{V}^{\dot{\pi}}(s) &= \left(H^{\dot{\pi}'}V_0^{\dot{\pi}}(s), \cdots, H^{\dot{\pi}'}V_{|E|-1}^{\dot{\pi}}(s)\right) \\ &= \left(Q_0^{\dot{\pi}}(s, \dot{\pi}'(s)), \cdots, Q_{|E|-1}^{\dot{\pi}}(s, \dot{\pi}'(s))\right).\end{aligned}$$

Os seguintes lemas e teoremas demonstram que o algoritmo de iteração de políticas para LL-MDPs converge para a solução ótima do LL-MDP dado como entrada.

Teorema 4.3 (Admissibilidade da política gerada pelo LLPI)
Seja $\mathcal{P}_i = (\mathcal{M}, \dot{\pi}_i)$ uma política admissível para um LL-MDP L. Seja $\dot{\pi}^*$ a política obtida pelo Algoritmo 4.3 tendo L como entrada. Então, $\dot{\pi}^*$ é admissível para L.

Prova *A admissibilidade da política depende de dois fatos: primeiro, a linguagem do autômato da política deve estar contida na linguagem do autômato do LL-MDP (o que é trivialmente verdade, já que o autômato do LL-MDP é usado na política); segundo, $\forall s \in S, \forall q \in E, \dot{\pi}_q(s) \in A_q$ (o que também é verdade, já que o* $\arg\max$ *da linha 18 do algoritmo seleciona $a \in A_q$, e a política inicial é admissível).* ■

Lema 4.4 (Convergência do LLPI)
Sejam $\overline{M}$ a maior recompensa em alguma época de decisão e $\underline{M}$ a menor recompensa possível em alguma época de decisão. Se $-\infty < \underline{M} \leq \overline{M} < \infty$, então

$$\forall \dot{\pi} \in \dot{\Pi}, \forall \dot{U} \in \dot{\mathcal{V}}, \quad \lim_{N\to\infty}\left[\left(\dot{H}^{\dot{\pi}}\right)^N \dot{U}\right] = \dot{V}^{\dot{\pi}}$$

onde $(\dot{H}^{\dot{\pi}'})^N$ denota a aplicação do operador N vezes.

Prova *Se o operador $\dot{H}^{\dot{\pi}}$ é aplicado L vezes sobre $\dot{U}$ (resultando em $(\dot{H}^{\dot{\pi}})^L\dot{U}$), pode-se dividir a recompensa esperada da função valor resultante em duas partes: a recompensa da função valor $\dot{U}$, antes do operador ser aplicado, e a recompensa relativa às L primeiras épocas de decisão, dadas pela aplicação do operador $\dot{H}^{\dot{\pi}}$. Cada vez que o operador $\dot{H}^{\dot{\pi}}$ é aplicado, a componente relativa a $\dot{U}$ é multiplicada pelo fator de desconto γ. O valor desta componente quando L tende ao infinito é zero. A seguir esta idéia é apresentada formalmente.*

Dada uma função valor $\dot{V}$, pode-se dividir seus componentes em duas partes, uma com a recompensa até L épocas de decisão, e outra com a recompensa a partir da época de decisão número L:

$$\begin{aligned}\dot{V}^{\dot{\pi}}(s) &= \lim_{N\to\infty}\left[\sum_{k=0}^{N-1}\gamma^k r_k\right] \\ &= \sum_{k=0}^{L-1}\gamma^k r_k + \lim_{N\to\infty}\left[\sum_{k=L}^{N-1}\gamma^k r_k\right]\end{aligned}$$

A função valor resultante de $(\dot{H}^{\dot{\pi}})^L\dot{U}$ *pode ser decomposta da mesma forma, e o limite desta função quando* $L \to \infty$ *é:*

$$\begin{aligned}\lim_{N\to\infty}\left[\left(\dot{H}^{\dot{\pi}}\right)^N\dot{U}\right] &= \lim_{L\to\infty}\left\{\left[\sum_{k=0}^{L-1}\gamma^k r_k\right]+\lim_{N\to\infty}\left[\sum_{k=L}^{N-1}\gamma^k r_k^U\right]\right\}\\ &= \dot{V}^{\dot{\pi}}+\lim_{L\to\infty}\left\{\lim_{N\to\infty}\left[\sum_{k=L}^{N-1}\gamma^k r_k^U\right]\right\} \qquad (4.8)\end{aligned}$$

onde r_k^U *é a recompensa, na época de decisão* k*, relativa à política associada a* $\dot{U}$*. A soma das recompensas deve ser no mínimo igual à soma da recompensa mínima para todas as épocas de decisão:*

$$\lim_{N\to\infty}\left[\sum_{k=L}^{N-1}\gamma^k r_k^U\right] \geq \underline{M}\sum_{k=L}^{\infty}\gamma^k = \frac{\underline{M}\gamma^L}{1-\gamma}$$

e simetricamente, a soma das recompensas deve ser no máximo a soma da recompensa máxima:

$$\lim_{N\to\infty}\left[\sum_{k=L}^{N-1}\gamma^k r_k^U\right] \leq \overline{M}\sum_{k=L}^{\infty}\gamma^k = \frac{\overline{M}\gamma^L}{1-\gamma}.$$

No entanto, $\lim_{L\to\infty}\left[\frac{C\gamma^L}{1-\gamma}\right] = 0$ *para qualquer constante* C*.*

Concluímos então, usando a igualdade na Equação 4.8 que

$$0 \leq \dot{V}^{\dot{\pi}}+\lim_{L\to\infty}\left\{\lim_{N\to\infty}\left[\sum_{k=L}^{N-1}\gamma^k r_k^U\right]\right\} \leq 0,$$

e $\lim_{N\to\infty}\left[\left(\dot{H}^{\dot{\pi}}\right)^N\dot{U}\right] = \dot{V}^{\dot{\pi}}$*.* ■

Lema 4.5 (Melhoria da política no LLPI)
Sejam $\mathcal{P} = (\mathcal{M}, \dot{\pi})$ *e* $\mathcal{P}' = (\mathcal{M}, \dot{\pi}')$ *duas políticas para um LL-MDP* L*. Se* $\dot{H}V^{\dot{\pi}} = \dot{H}^{\dot{\pi}'}V^{\dot{\pi}}$*, então* $\dot{\pi}'$ *é uniformemente melhor do que* $\dot{\pi}$*.*

Prova *Como por hipótese* $\dot{H}^{\dot{\pi}'}V^{\dot{\pi}} = \dot{H}V^{\dot{\pi}}$ *e* $HV_q^{\dot{\pi}}(s) = \max_{a\in A_q}Q(s,a)$*, pode-se concluir que* $\dot{H}^{\dot{\pi}'}\dot{V}^{\dot{\pi}} \geq \dot{V}^{\dot{\pi}}$*.*

Repetidas aplicações do operador $\dot{H}^{\dot{\pi}'}$ *levam a:*

$$\dot{V}^{\dot{\pi}} \leq \dot{H}^{\dot{\pi}'}\left(\dot{V}^{\dot{\pi}}\right) \leq \cdots \leq \left(\dot{H}^{\dot{\pi}'}\right)^k\left(\dot{V}^{\dot{\pi}}\right) \leq \cdots \leq \lim_{k\to\infty}\left(\dot{H}^{\dot{\pi}'}\right)^k\left(\dot{V}^{\dot{\pi}}\right).$$

Usando o Lema 4.4, concluímos que $\dot{V}^{\dot{\pi}} \leq \dot{V}^{\dot{\pi}'}$*.* ■

Lema 4.6 (Corretude do critério de parada do LLPI)
Seja L um LL-MDP e $\dot{\pi}_i$ uma política para L. Se o valor esperado para $\dot{\pi}$ não pode ser melhorado pelo algoritmo LLPI, ou seja:

$$\forall q \in E, \forall s \in S, \;\; \dot{\pi}_q(s) = \arg\max_{a \in A_q} Q_q(s,a) \tag{4.9}$$

então $\dot{\pi}$ é ótima para L.

Prova *Uma política que não pode ser melhorada pelo LLPI obedece à Equação 4.9, que pode ser reescrita como*

$$\forall q \in E, \forall s \in S, \;\; \dot{\pi}_q(s) = \arg\max_{a \in A_q} \left[R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) V_{\delta(q,a)}(s) \right],$$

e sua função valor, portanto, é tal que

$$V_q(s) = \max_{a \in A_q} \left[R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) V_{\delta(q,a)}(s) \right],$$

o que significa que a função valor satisfaz o critério de otimalidade. Uma vez que pelo teorema do ponto fixo de Banach há um único ponto em $\dot{\mathcal{V}}$ que satisfaz esta equação, esta função valor e sua política então são ótimas. ■

Teorema 4.4 (Otimalidade da política gerada pelo LLPI)
Para qualquer LL-MDP $L = (S, A, T, R, \mathcal{M})$ dado como entrada, o algoritmo LLPI para após um número finito de iterações, e a política resultante é ótima.

Prova *O Teorema 4.3 garante a admissibilidade da política gerada pelo algoritmo. O Lema 4.5 garante que os valores das políticas geradas a cada iteração do algoritmo crescem monotonicamente. Como há um número finito de políticas possíveis para um LL-MDP, o algoritmo deverá parar. De acordo com o critério de parada, na última iteração $\dot{\pi}_{n+1,q} = \dot{\pi}_{n,q}$ (ou seja, a política não pode ser melhorada). Com isto o Lema 4.6 garante que a política gerada pelo LLPI é ótima.* ■

4.3.5 Programação Linear

Esta seção descreve a modelagem de LL-MDPs como programas lineares. Ao contrário dos algoritmos iterativos baseados em programação dinâmica, a abordagem da multiplicação de estados e a abordagem *ad-hoc* resultam no mesmo programa linear.

Assim como para MDPs, é possível criar um programa linear cuja solução é a política ótima para um LL-MDP. Como cada restrição descreve a dependência de v_s em outras

funções v'_s, basta que as restrições sejam criadas por ação e por estado do autômato, e que as funções usadas para construir cada função valor respeitem a linguagem do autômato. Nesta discussão, usaremos a notação $v_{[q,s]}$ para a função valor relacionada ao estado s do MDP e ao estado q do autômato. O programa linear para determinar a política de um LL-MDP é:

$$\text{minimizar: } \sum_{s \in S, q \in E} v_{[q,s]}$$

sujeito a:

$$\forall s \in S, a \in A, q' \in E \text{ tais que } \delta(q,a) = q',$$
$$v_{[q,s]} \geq \gamma \sum_{s' \in S} T(s'|s,a) v_{[q',s']}.$$

Por exemplo, o MDP modelado no Capítulo 2, na página 15 pode ser modificado de forma que a ação a_0 sempre tenha que ser a primeira a ser executada. O LL-MDP resultante usaria o autômato da Figura 4.3.

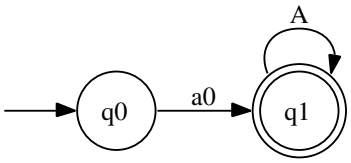


Figura 4.3: Autômato que limita as ações disponíveis no primeiro passo do MDP.

O programa linear para resolver este LL-MDP é:

$$\text{minimizar: } v_{[q0,s0]} + v_{[q1,s0]} + v_{[q0,s1]} + v_{[q1,s1]}$$

sujeito a:

$$\begin{aligned}
v_{[q0,s0]} &\geq 10 + \gamma 0.3 v_{[q1,s0]} + \gamma 0.7 v_{[q1,s1]}, \\
v_{[q0,s1]} &\geq 50 + \gamma 0.4 v_{[q1,s0]} + \gamma 0.5 v_{[q1,s1]}, \\
v_{[q1,s0]} &\geq 10 + \gamma 0.3 v_{[q1,s0]} + \gamma 0.7 v_{[q1,s1]}, \\
v_{[q1,s0]} &\geq 1 + \gamma 0.2 v_{[q1,s0]} + \gamma 0.4 v_{[q1,s1]}, \\
v_{[q1,s0]} &\geq 30 + \gamma 0.6 v_{[q1,s0]} + \gamma 0.4 v_{[q1,s1]}, \\
v_{[q1,s1]} &\geq 50 + \gamma 0.5 v_{[q1,s0]} + \gamma 0.5 v_{[q1,s1]}, \\
v_{[q1,s1]} &\geq 20 + \gamma 0.6 v_{[q1,s0]} + \gamma 0.4 v_{[q1,s1]}, \\
v_{[q1,s1]} &\geq 2 + \gamma 0.7 v_{[q1,s0]} + \gamma 0.3 v_{[q1,s1]}.
\end{aligned}$$

O programa linear obtido desta forma é o mesmo obtido a partir do MDP resultante da multiplicação de estados.

4.4 Sumário

Este capítulo descreve teoricamente algumas das contribuições centrais desta tese:

- A definição de MDPs limitados por linguagem;
- Algoritmos para solução dos novos problemas descritos: um multiplicador de estados e adaptação dos algoritmos exatos já existentes;
- Prova de convergência e corretude dos algoritmos modificados.

Capítulo 5

POMDPs Limitados por Linguagem

Este capítulo apresenta os processos de Markov parcialmente observáveis limitados por linguagem, análogos a LL-MDPs. Outra extensão descrita neste capítulo permite determinar para cada evento de um POMDP uma duração, de forma que esta seja levada em conta durante o cálculo da política ótima. A duração dos eventos deve ser discreta, e pode ser probabilística.

5.1 LL-POMDPs

A definição de LL-POMDPs extende a de LL-MDPs da mesma forma que os POMDPs extendem os MDPs, e um LL-POMDP pode ser visto como um LL-MDP sobre estados de crença.

Definição 5.1 (LL-POMDP)
Um POMDP limitado por linguagem (LL-POMDP) é uma tupla $(S, A, T, R, \Omega, O, \mathcal{M})$*, onde* $\mathcal{M}$ *é um autômato descrevendo a ordem em que eventos podem acontecer quando a política estiver sendo executada, da mesma forma que em um LL-MDP.*

Definição 5.2 (POMDP subjacente a um LL-POMDP)
Dado um LL-POMDP $L = (S, A, T, R, \Omega, O, \mathcal{M})$*, o* POMDP subjacente *a* L *é* (S, A, T, R, Ω, O).

Os eventos em um POMDP são pares de ação e observação (e não ação e estado resultante, como em MDPs). Assim, o alfabeto do autômato de um LL-POMDP é composto de pares (a, o), onde $a \in A$ e $o \in \Omega$.

Definição 5.3 (Evento em um POMDP)
No contexto de POMDPs, usaremos a palavra evento *para descrever uma ação ou uma observação ou um par ação/observação (ou seja, uma ação seguida de uma observação). Assim, a ação* a_4*, a observação* o_8 *e o par* (a_4, o_8) *são chamados de "eventos".*

Nem toda linguagem regular funcionaria para um LL-POMDP: é necessário garantir que durante a execução da política, todas as observações decorrentes de uma ação sejam levadas em consideração. Isso significa que no autômato $\mathcal{M}$, para cada estado q e cada ação a que possa resultar em q, devemos garantir que todas as transições *saindo de q* contenham todos os eventos $< a, o >$, para todas as observações;

5.1.1 Política para um LL-POMDP

Da mesma forma que para LL-MDPs, uma política para um LL-POMDP terá uma política para POMDP (uma função valor) para cada estado de um autômato.

Definição 5.4 (Política para LL-POMDP)
Uma política $\mathcal{P}$ para um LL-POMDP é um par $(\mathcal{W}, \dot{\pi})$ onde:

- $\mathcal{W} = (\Sigma, \mathcal{E}, \mathcal{F}, \mathcal{N}, \Delta)$, *um autômato com alfabeto Σ, estados $\mathcal{E}$, estados finais $\mathcal{F}$, estados iniciais $\mathcal{N}$, e função de transição Δ;*
- $\dot{\pi} = \{d_0, d_1, \cdots, d_{|\mathcal{E}|-1}\}$ *é um conjunto de regras de decisão, sendo que para cada $q \in \mathcal{E}$, há uma regra de decisão $d_q : \mathcal{B} \mapsto A$.*

O autômato $\mathcal{M}$ é usado para horizonte infinito, e o autômato $\mathcal{W}$ é construído para horizonte finito da mesma forma que para LL-MDPs.

Definição 5.5 (Função valor para um LL-POMDP)
Dado um LL-POMDP $L = (S, A, T, R, \Omega, O, \mathcal{M})$ e uma política $\mathcal{P} = (\mathcal{W}, \dot{\pi})$ para L, uma função valor $\dot{V}$ para L é dada pelo conjunto $\dot{V} = \{V_0, V_1, \cdots, V_{|\mathcal{E}|-1}\}$ de funções valor, onde $V_q(s)$ é o valor esperado para a política $\mathcal{P}$ nos estados q e s de acordo com o critério de otimalidade escolhido.

O Algoritmo 5.1 deve ser usado para a execução de políticas para horizonte finito, e é semelhante ao Algoritmo 4.1 para LL-MDPs.

Da mesma forma que para LL-MDPs, se uma política permite a execução de ações proibidas pelo autômato de um LL-POMDP L, ela não é admissível para L.

Definição 5.6 (Política admissível para um LL-POMDP)
Seja $L = (S, A, T, R, \Omega, O, \mathcal{M})$ um LL-POMDP. Uma política $\mathcal{P} = (\mathcal{W}, \dot{V})$ é admissível para L se:

- *A linguagem de $\mathcal{W}$ está contida na linguagem de $\mathcal{M}$;*
- *Para toda época de decisão k, $\forall b \in \mathcal{B}, \forall q \in \mathcal{E}, d_q(b) \in A_q$.*

Entrada: $\mathcal{P} = (\mathcal{W}, \dot{\pi})$, uma política para LL-POMDP, onde $\mathcal{W} = (\Sigma, \mathcal{E}, \mathcal{F}, \mathcal{N}, \Delta)$ e $\dot{\pi} : \mathcal{E} \times \mathcal{B} \mapsto A$;
b, o estado inicial de crença do LL-POMDP.

```
k ← 0
Escolha q ∈ 𝒩
enquanto k < h faça
    a ← π̇([k, q], b)
    Execute a
    Receba nova observação o
    q ← Δ(q, < a, o >)
    b ← τ(b, a, o)
    t ← k + 1
fim
```

Algoritmo 5.1: Execução de política para LL-POMDP de horizonte finito.

Assim como a definição de política admissível para LL-MDPs, a definição de política admissível para LL-POMDPs vale tanto para horizonte finito como para horizonte infinito.

Uma função valor $\dot{V}$ para LL-POMDPs é admissível se a política induzida por ela também o for ($\forall t < z, \forall b \in \mathcal{B}, \forall q \in \mathcal{E},\ \arg\max_{a \in A} \dot{V}_{[k,q]}(b) \in A_q$, e a linguagem de $\mathcal{W}$ deve estar contida na de $\mathcal{M}$).

A recompensa para uma ação em um estado de crença é:

$$Q^{\pi}(q, b, a) = \rho(b, a) + \gamma \sum_{o \in \Omega} Pr(o|b, a) V^{\pi}_{\Delta(q, <a,o>)}(\tau(b, a, o)).$$

A definição de política ótima para LL-POMDPs é a mesma usada para LL-MDPs, trocando S por $\mathcal{B}$:

Definição 5.7 (Política ótima para um LL-POMDP)
Seja $L = (S, A, T, R, \Omega, O, \mathcal{M})$ um LL-POMDP. Seja $\mathcal{P} = (\mathcal{W}, \dot{\pi})$ uma política para L.

- *$\mathcal{P}$ é admissível para L;*
- *Para toda época de decisão k e política π, seja $d_{[k,q]}$ a regra de decisão de π para o estado q do autômato em k.*

$$\forall \dot{\pi}' \in \dot{\Pi}, \forall q \in \mathcal{E}, \forall b \in \mathcal{B},\ \ Q^{\dot{\pi}}(q, b, d_q(b)) \geq Q^{\dot{\pi}'}(q, b, d'_q(b)).$$

Definição 5.8 (Função valor ótima para um LL-POMDP)
Seja $L = (S, A, T, R, \Omega, O, \mathcal{M})$ um LL-POMDP. Uma função valor $\dot{V}$ é ótima para L se $\forall \dot{U} \in \dot{\mathcal{V}}, \forall q \in \mathcal{E}, \forall b \in \mathcal{B},\ \ V_q(b) \geq U_q(b)$.

5.1.2 Recompensa para um LL-POMDP

De forma semelhante aos LL-MDPs, a função valor do POMDP subjacente é um limite superior para a função valor do LL-POMDP (o mesmo argumento usado na subseção 4.2.2 na página 69 se aplica aos LL-POMDPs).

5.2 Algoritmos

As subseções a seguir apresentam duas maneiras de resolver LL-POMDPs. Assim como LL-MDPs, LL-POMDPs podem ser resolvidos pela multiplicação de seus estados ou por algoritmos para POMDPs modificados para trabalhar com funções valor de LL-POMDPs.

5.2.1 Multiplicação estados

Da mesma forma que fizemos com LL-MDPs, podemos usar o produto cartesiano dos estados do autômato e do LL-POMDP para gerar um novo POMDP. Para cada estado p no autômato $\mathcal{M}$ do LL-POMDP e cada $s_k \in S$, criamos um estado s_k^p no novo POMDP. Como o conjunto de estados é aumentado, as T', R', e O' são modificadas:

- $S' = (S \times E)$. Para cada $s_m \in S$ e $q \in E$, o novo conjunto de estados S' terá um estado s_m^q.
- A nova função de transição é

$$T'(s_n^q | s_m^p, a) = \begin{cases} T(s_n | s_m, a) & \text{se } (p, a, q) \in \delta \\ 1 & \text{se } (p, a, q) \notin \delta, \quad s_m = s_n \text{ e } p = q \\ 0 & \text{se } (p, a, q) \notin \delta \text{ e } (s_m \neq s_n \text{ ou } p \neq q); \end{cases}$$

- A nova função de probabilidade de observação é

$$O'(o | s_m^p, a) = O(o | s_m, a);$$

- E finalmente, a nova função de recompensa é

$$R'(s_m^p, a) = \begin{cases} R(s_m, a) & \text{se } \exists q \in E \text{ tal que } (p, a, q) \in \delta \\ -\infty & \text{caso contrário.} \end{cases}$$

O novo POMDP tem mais estados que o LL-POMDP original. Por isso, o estado de crença inicial, que diz respeito a apenas $|S|$ estados, deve ser mapeado no novo conjunto de estados. Isto é feito trivialmente, da seguinte forma: seja b o estado de crença inicial

do LL-POMDP e p o estado inicial do autômato $\mathcal{M}$. Então calculamos o novo estado de crença b':

$$\forall s_m \in S,\ b'(s_m^p) = b(k),$$

$$\forall s_m \in S,\ \forall q \neq p,\ b'(s_m^q) = 0.$$

À medida que a política é usada e ações são executadas, o estado de crença mudará e ficará não-negativo em no máximo $|S|$ estados de cada vez, uma vez que o estado corrente do autômato é perfeitamente observável.

Este método funciona somente quando os eventos são apenas ações (não podemos usar observações como seletores). Se este algoritmo for aplicado diretamente em POMDPs com observações seletoras, o resultado será um POMDP que não é consistente. Por exemplo, se o POMDP original tiver:

- $S = \{s_1, s_2\}$
- $\Omega = \{o_1, o_2\}$
- $A = \{a_2, a_2\}$

e o autômato da Figura 5.1 for usado na multiplicação, o POMDP da figura 5.2 seria gerado.

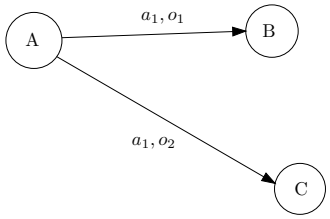


Figura 5.1: Autômato $\mathcal{M}$

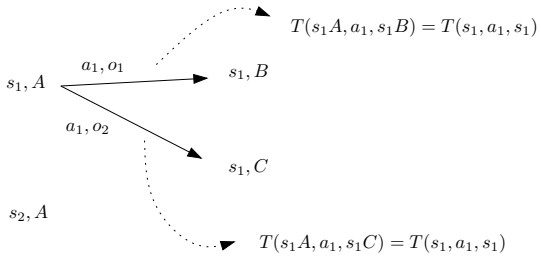


Figura 5.2: POMDP resultante da multiplicação de estados.

No entanto, as probabilidades de transição para (As_1, a_1) não somam um. Este problema pode ser resolvido multiplicando novamente o espaço de estados: o POMDP original tinha $|S|$ estados. O POMDP que leva em consideração os estados do autômato tem $|S||E|$ estados. Pode-se criar, ainda, um novo POMDP, que usa estados para representar a última observação que foi percebida pelo agente. Este novo POMDP terá $|S||E||\Omega|$ estados. Assim, uma transição passa a ter sua probabilidade multiplicada pela probabilidade de acontecimento da observação referente ao estado destino, como a Figura 5.1 mostra.

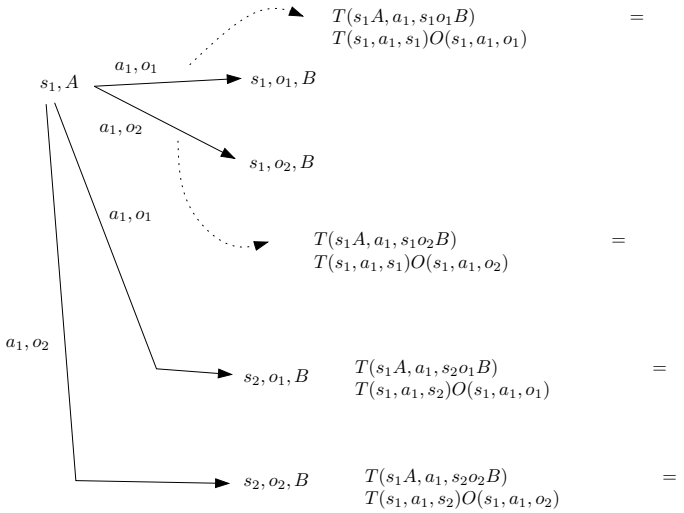


Figura 5.3: POMDP resultante da multiplicação de estados (com probabilidades corretas).

Durante o uso da política, o tomador de decisões sempre saberá em que estado do autômato ele está, e portanto o estado de crença ficará sempre restrito a no máximo $|S|$ valores diferentes de zero. Por exemplo, para um autômato com três estados e POMDP com 5 estados, o estado de crença poderia ter os seguintes valores (as partes dos estados de crença equivalentes a estados do autômato foram destacadas):

$$\underline{0.0, 0.2, 0.4, 0.1, 0.3}, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0$$
$$0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, \underline{0.4, 0.0, 0.0, 0.5, 0.1}$$
$$0.0, 0.0, 0.0, 0.0, 0.0, \underline{0.3, 0.0, 0.1, 0.1, 0.5}, 0.0, 0.0, 0.0, 0.0, 0.0$$

Hauskrecht definiu *MDPs híbridos*, que combinam MDPs e POMDPs [37, 36], e os usou na modelagem de diagnóstico e planejamento de terapia para doenças coronárias. Hauskrecht demonstrou que é possível usar uma função valor PWLC para cada variável perfeitamente observável, onde cada uma das funções valor é definida sobre o conjunto

das variáveis parcialmente observáveis. Esta abordagem leva o algoritmo LLVI para POMDPs, descrito na subseção 5.2.2. De acordo com Zhang e Zhang [88, 89], um POMDP onde uma observação pode restringir o estado de crença a poucos estados é um POMDP *informativo*, e o algoritmo de *iteração de valores restrita* explora esta estrutura, reduzindo a dimensão do espaço de crença.

Os algoritmos baseados em pontos de crença fazem uma busca inicial por todos os estados de crença antes de começar o laço onde a função valor é melhorada. Para um POMDP multiplicado, nenhum estado de crença que considere mais de um estado do autômato ao mesmo tempo será incluído na grade, porque as probabilidades de transição não levam a estes estados. Isto resulta em uma grade menor do que o esperado para um espaço de estados grande, mas ainda assim eficiente.

Complexidade

Usando os termos do POMDP já multiplicado, a complexidade do algoritmo de iteração de valores é $\mathcal{O}(|S||A|^{|\Omega|^z})$. Usando os termos do LL-POMDP original, a complexidade é $\mathcal{O}(|S||E||A|^{|\Omega|^z})$. Com observações seletoras, $\mathcal{O}(|S||E||\Omega||A|^{|\Omega|^z})$.

No entanto, a multiplicação de estados permite que qualquer algoritmo seja usado para a solução do POMDP resultante, e especialmente para horizonte infinito, há algoritmos mais eficientes que o de iteração de valores.

5.2.2 Adaptações dos algoritmos existentes

Da mesma forma que com MDPs, diversos algoritmos podem ser adaptados para resolver LL-POMDPs para que trabalhem com a linguagem do autômato $\mathcal{M}$. Alguns dos algoritmos que poderiam ser adaptados são os de iteração de valores (que seleciona ações e observações a cada iteração), o de iteração de políticas de Hansen (que seleciona também ações e observações, da mesma forma que um algoritmo de iteração de valores), embora alguns destes possam se beneficiar mais da multiplicação de estados.

As modificações são:

- Uma função valor (ou controlador) é mantida para cada estado do autômato $\mathcal{M}$;
- Ao gerar uma nova função valor para o estado q, são consideradas as funções valor dos estados $OUT(q)$.

5.2.3 Iteração de valores (LLVI)

Esta subseção descreve o algoritmo de iteração de valores para POMDPs limitados por linguagens.

Os algoritmos de iteração de valores para a solução de POMDPs vistos no Capítulo 3 podem ser adaptados para, a cada passo, consultar o autômato $\mathcal{M}$ e levar em conta a ordem desejada para os eventos, da mesma forma como os algoritmos para MDPs foram adaptados. A cada passo, ao invés de construir um novo conjunto de vetores (uma função valor) Γ_i, o algoritmo construirá uma função valor por estado onde o autômato possa estar. Ou seja, no i-ésimo passo de programação dinâmica, o algoritmo construirá as funções-valor $\Gamma_{[i,p]}$, onde $p \in E$. A cada passo, o algoritmo deverá se lembrar de quais são os estados do autômato onde o agente poderá estar naquela época de decisão. A Figura 5.4 mostra a função valor do LL-POMDP sendo construída.

O mapeamento $h : \mathcal{B} \times A \times \dot{\mathcal{V}} \times \mathcal{E} \mapsto \mathbb{R}$ é

$$h(b, a, \dot{V}, q) = \rho(b, a) + \gamma \sum_{o \in \Omega} Pr(o|b, a) V_{\Delta(q,<a,o>)}(\tau(b, a, o)).$$

O operador de melhoria local é semelhante àquele definido para LL-MDPs na Equação 4.1; a única diferença é o uso de um estado de crença no lugar do estado do MDP:

$$HV_q(b) = \max_{a \in A_q} \left[h(b, a, \dot{V}, q) \right].$$

E operador $\dot{H} : \dot{\mathcal{V}} \mapsto \dot{\mathcal{V}}$ de melhoria global é

$$\dot{H}\dot{V} = \left(HV_{q_0}, HV_{q_1}, \cdots HV_{q_{|\mathcal{E}|-1}} \right).$$

A equação de otimalidade para LL-POMDPs é

$$\begin{aligned} \forall q \in E, b \in \mathcal{B}, V_q(b) &= HV_q(b) \\ &= \max_{a \in A_q} h(b, a, \dot{V}, q) \\ &= \max_{a \in A_q} \sum_{s \in S} R(s, a) b(s) + \gamma \sum_{o \in \Omega} Pr(o|b, a) V_{\Delta(q,<a,o>)}(\tau(b, a, o)). \end{aligned}$$

O Algoritmo 5.2 mostra um passo de programação dinâmica, modificado para levar em consideração apenas as ações e observações definidas por um autômato $\mathcal{M}$. O algoritmo implementa a variante variante de Monahan para construir a função valor. A linha 8 do algoritmo calcula o conjunto de vetores $\Gamma^a_{[i,p]}$ para cada ação. Apenas as ações que resultam em alguma observação são selecionadas na linha 7.

Usando a política

O algoritmo constrói $\mathcal{E}$ e Δ. Durante a execução da política, o autômato $\mathcal{W} = (\Sigma, \mathcal{E}, \mathcal{F}, \mathcal{N}, \Delta)$ é usado, onde:

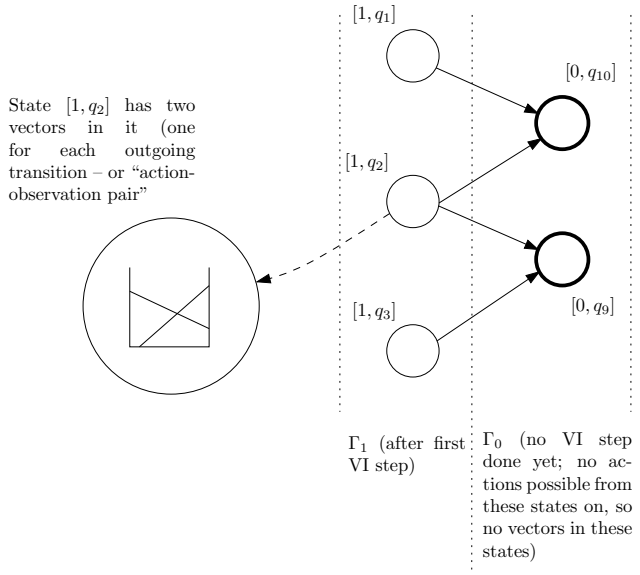


Figura 5.4: Estados finais do autômato construído.

- $\mathcal{N} = \{[z-1, q] | q \in I\}$ é o conjunto de estados iniciais, determinado pelo horizonte z e pelo conjunto I de estados iniciais de $\mathcal{M}$;

- $\mathcal{F} = \cup_{q \in F}\{[0, q]\}$ é o conjunto de estados finais.

Além deste autômato, há também o conjunto funções valor $\Gamma_{[i,q]}$, onde $0 \leq i < z$ e $q \in \mathcal{E}$. Estas duas estruturas formam a política para o LL-POMDP. O algoritmo 5.3 mostra como a política poderia ser usada em um LL-POMDP de horizonte finito.

Horizonte infinito

A convergência para horizonte infinito pode ser verificada usando a conjunção das convergências das funções valor de cada estado do autômato, da mesma forma que no caso dos LL-MDPs.

Corretude e convergência

O algoritmo modificado de iteração de valores produzirá uma política ótima dentro dos limites da linguagem do autômato $\mathcal{M}$. As provas apresentadas para o algoritmo LLVI para LL-MDPs valem também para LL-POMDPs, bastando que os estados sejam entendidos como "estados de crença".

Entrada: Um LL-POMDP $(S, A, T, R, \Omega, O, \mathcal{M})$ onde $\mathcal{M} = (\Sigma, E, F, q_0, \delta)$;
O conjunto $\mathcal{E}$ construído no passo anterior;
A função valor Γ, parcialmente construída nos passos anteriores e definida para todos os estados $[k, q] \in \mathcal{E}, k \leq i-1$;
A relação Δ, parcialmente construída nos passos anteriores;
O número i do passo atual.

Saída: Uma função valor Γ, definida para todos os estados $[k, q] \in \mathcal{E}, k \leq i$;
Um conjunto $\mathcal{E}$ estados;
Uma relação $\Delta : \mathcal{E} \times \Sigma \to \mathcal{E}$.

para cada $p \in E$, *tal que* $\exists e \in \Sigma, \delta(q, e) = p$ **faça**
  $\mathcal{E} \leftarrow \mathcal{E} \cup \{[i, p]\}$
  $\Delta([i, p], e) \leftarrow [i-1, q]$
fim
para cada $[i, p] \in \mathcal{E}$ *(onde i é a iteração sendo executada)* **faça**
  para cada $[i-1, q] \in \mathcal{E}$ **faça**
    para cada $a \in A$ *tal que* $\exists o \in \Omega,\ \delta(p, < a, o >) = q$ **faça**
      $\Gamma^a_{[i,p]} \leftarrow \Gamma^a_{[i,p]} \cup \left\{ r_a + \gamma \left(\sum_{\substack{o \in \Omega, \\ \delta(p, <a,o>) = q}} M^{a,o} \alpha \right) \;\middle|\; \alpha \in \Gamma_{[i-1,q]} \right\}$
    fim
  fim
  $\Gamma_{[i,p]} \leftarrow \bigcup_{a \in A} \Gamma^a_{[i,p]}$
fim
Devolva Γ, $\mathcal{E}$ e Δ

Algoritmo 5.2: Passo de programação dinâmica para LL-POMDPs.

Complexidade

Sejam q e q' dois estados do autômato $\mathcal{W}$, $\Gamma_{q'}$ o conjunto de vetores em q', e $\Omega_{q,a,q'}$ o conjunto de todas as observações que podem resultar da ação a no estado q, levando o autômato para o estado q':

$$\Omega_{q,a,q'} = \{o \in \Omega | \delta(q, < a, o >) = q'\}.$$

O número máximo de vetores $\Gamma_{q,q'}$ calculados entre q e q' no algoritmo LLVI para LL-POMDPs pode ser determinado da seguinte forma: para cada ação em A_q deve-se considerar o conjunto de observações possíveis $\Omega_{q,a,q'}$ e as possíveis ações subsequentes $A_{q'}$. Com isto, para cada ação em A_q são gerados $\Gamma_{q'}^{|\Omega_{q,a,q'}|}$ vetores. O número total de vetores gerados entre q e q' é:

$$\Gamma_{q,q'} = \sum_{a \in A_q} |\Gamma_{q'}|^{|\Omega_{q,a,q'}|},$$

```
1 q ← q_0
2 b ← estado inicial
3 repita
4 |   a ← melhor_acao (Γ_q, b)
5 |   o ← execute (a)
6 |   q ← Δ(q, a, o)
7 |   b ← τ(b, a, o)
8 até completar z iterações
```

Algoritmo 5.3: Algoritmo de execução de política para LL-POMDPs.

e o total de vetores em q é $\sum_{q' \in OUT(q)} \Gamma_{q,q'}$.

O grafo que representa o autômato $\mathcal{W}$ também representa uma rede de fluxo com múltiplas fontes e sorvedouros. Os estados finais de $\mathcal{W}$ são as fontes, e os estados iniciais são os sorvedouros. Para cada transição do autômato há um arco na rede. O número de vetores levados de um estado de $\mathcal{W}$ a outro é o fluxo de um nó da rede a outro. Seja q um nó da rede que representa um estado do autômato $\mathcal{W}$, em algum passo do algoritmo. A função que dá o fluxo saindo de q é

$$f(q) = \sum_{q' \in OUT(q)} \sum_{a \in A_q} f(q')^{|\Omega_{q,a,q'}|},$$

Em toda fonte $[0, q]$,

$$f([0, q]) = 1$$

porque da fonte, só é possível obter um único vetor para cada ação.

O fluxo máximo total na rede dá a complexidade de tempo do algoritmo para o pior caso (ou seja, o caso em que nenhum vetor dominado é encontrado).

O número máximo de vetores gerado por iteração do LLVI pode ser menor que o gerado pelo algoritmo de iteração de valores sobre o POMDP multiplicado, porque

$$\begin{aligned}
\Gamma_{[i,q]} &= \sum_{q' \in OUT(q)} \sum_{a \in A_q} |\Gamma_{[i-1,q']}|^{|\Omega_{q,a,q'}|} \\
&\leq \sum_{q' \in E} \sum_{a \in A} |\Gamma_{[i-1,q']}|^{|\Omega_{q,a,q'}|} \\
&\leq |E||A||\Gamma_{i-1}|^{|\Omega|} \qquad (5.1) \\
&\leq |\Omega||E||A||\Gamma_{i-1}|^{|\Omega|}. \qquad (5.2)
\end{aligned}$$

As linhas 5.1 e 5.2 são os números de vetores gerados para o POMDP multiplicado sem e com observações seletoras, respectivamente. No entanto, como vetores dominados são eliminados, o algoritmo de iteração de valores poderia ter melhor desempenho. Em

nossos experimentos o LLVI se mostrou sempre melhor que a iteração de valores sobre o POMDP multiplicado.

Uma melhoria

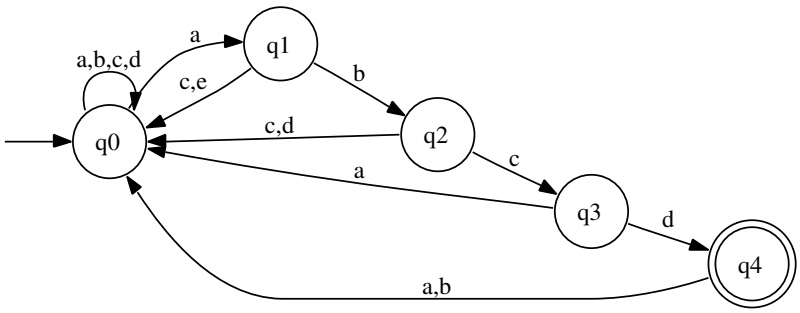


Figura 5.5: Autômato que causa operações desnecessárias de verificação de dominância.

O algoritmo LLVI pode computar a mesma função valor mais de uma vez, fazendo mais verificações de dominância que o necessário. Por exemplo, um LL-POMDP usa o autômato $\mathcal{M}$ da Figura 5.5. Na época i, ao calcular as funções valor para a época $i+1$, novas transições serão adicionadas ao novo autômato $\mathcal{W}$. A Figura 5.6 mostra as transições que seriam adicionadas a Δ terminando em $[i, q_0]$: as funções para $\{a, b\}$ e $\{c, d\}$ seriam calculadas, e a nova função $\{a, b, c, d\}$ seria calculada também. No entanto, com isto alguns testes de dominância seriam repetidos (se a domina b, este teste seria feito duas vezes). Poderíamos ter usado $\{a, b\}$ e $\{c, d\}$ (que já passaram por testes de dominância) para construir $\{a, b, c, d\}$. Como o teste de dominância é caro, seria melhor se o cálculo das funções valor sempre fosse feito aproveitando aquelas já calculadas, quando possível.

Uma maneira de resolver este problema usa o conceito de autômato reverso, definido neste capítulo. Denotamos um autômato determinístico com uma estrela (Z_* é determinístico), e o autômato reverso com um R: Z^R é Z reverso.

Os testes de dominância adicionais mostrados na Figura 5.6 só acontecem quando transições com o mesmo evento terminam no mesmo estado – o que é o mesmo que dizer que o problema acontece quando o autômato reverso $\mathcal{M}^R$ é não-determinístico. Para resolver o problema, basta reverter o autômato $\mathcal{M}$, transformá-lo em um autômato determinístico, e revertê-lo novamente. O autômato resultante $((\mathcal{M}^R)_*)^R$ tem a mesma linguagem de $\mathcal{M}$, e não terá transições com o mesmo evento terminando no mesmo estado. Isto evitará os testes de dominância desnecessários.

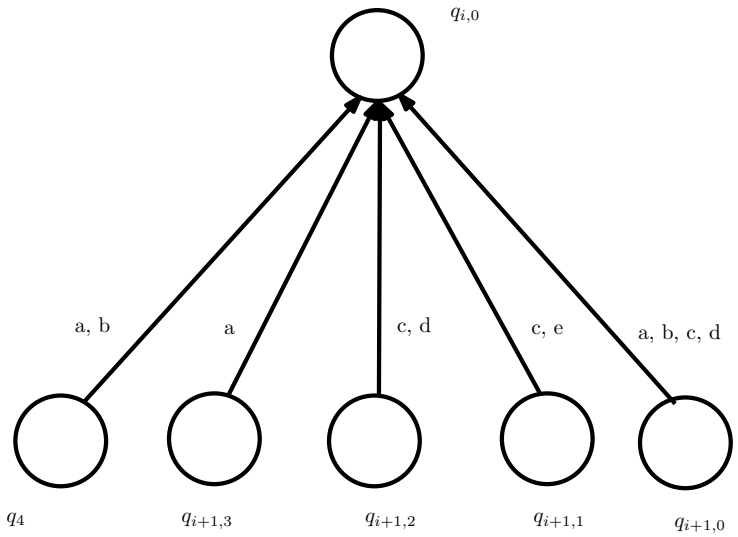


Figura 5.6: Operações desnecessárias realizadas pelo LL-POMDP-DP.

5.2.4 Algoritmos baseados em discretização

Os algoritmos baseados em discretização do espaço de crença também podem ser adaptados para LL-POMDPs. Há duas mudanças a serem feitas nestes algoritmos:

- A grade inicial pode ser determinada simulando os pontos de crença que poderiam ser alcançados a partir de um ponto de crença inicial. Isto permite que o algoritmo trabalhe com pontos de crença relevantes. A linguagem do autômato $\mathcal{M}$ pode restringir a busca por pontos de crença ao determinar a grade inicial, incluindo, para cada estado do autômato, os pontos relevantes para aquele estado;

- Novamente, há a necessidade de manter diferentes funções valor para cada estado do autômato $\mathcal{M}$. O passo de melhoria da função valor deve ser modificado de forma a fazer a melhoria em cada uma destas funções valor, da mesma forma já descrita para os algoritmos de iteração de valores.

Descreveremos estas modificações para o PBVI. A primeira modificação é simples; para a segunda, a Equação 3.4 (na página 53) passa a ser usada apenas para as ações e observações permitidas no estado q.

Primeiro o conjunto de vetores α a ser gerado é só o das ações permitidas em q:

$$\Gamma_q^{a,*} \leftarrow {\alpha_q}^{a,*}(s) = R(s,a).$$

Em seguida, os vetores provenientes de outras funções valor são usados, de acordo com as ações e observações possíveis:

$$\forall p \in E, a \in A, o \in \Omega \text{ tais que } \delta(q, < a, o >) = p,$$

$$\forall \alpha' \in \Gamma_{[i-1,p]},$$

$$\Gamma_q^{a,o}(s) \leftarrow \alpha_p^{a,o}(a) = \gamma \sum_{s' \in S} T(s'|s,a) O(o|s',a) \alpha'(s').$$

E cada ponto de crença tem seu vetor modificado:

$$\forall b,$$

$$\Gamma_{[b,q]}^{a} = \Gamma_q^{a,*} + \sum_{o \in \Omega_q} \arg\max_{\alpha \in \Gamma_q^{a,o}} \left(\sum_{s \in S} \alpha(s) b(s) \right).$$

Complexidade

Há uma diferença importante entre os algoritmos exatos de iteração de valores e os algoritmos baseados em pontos de crença: para estes últimos, o passo de melhoria é polinomial. Isto invalida, para estes algoritmos, o argumento a favor da divisão da função valor. Para o algoritmo simples com grade fixa, a análise de complexidade é semelhante àquela para iteração de valores em LL-MDPs.

5.3 Duração Probabilística Discreta para Eventos

Há situações nas quais é necessário levar em conta o fato de diferentes ações terem durações diferentes (um tratamento médico pode durar mais que outro, por exemplo). O mesmo acontece com observações: uma ação que dispara uma busca por um elemento em uma base de dados distribuída pode resultar em uma observação em um curto tempo (se a informação foi achada logo no início da busca) ou depois de um longo tempo (se a busca exaustiva não encontrou a informação, ou se um limite de tempo foi excedido).

A solução de POSMDPs, como já visto, é difícil – mas POMDPs e LL-POMDPs podem ser estendidos para que seja possível especificar um conjunto de durações para cada ação e cada observação, além da probabilidade de que cada uma das ações e observações tenha cada uma das durações definidas.

Definição 5.9 (LL-POMDP com durações)
Um LL-POMDP com durações para eventos é a tupla $(S, A, T, R, \Omega, O, \mathcal{M}, D)$. *A função* $D : A \times \Omega \times \mathbb{N} \mapsto [0,1]$ *determina a probabilidade de que cada evento tenha uma certa duração:* $D(a, o, j)$ *dá a probabilidade de que o evento* $< a, o >$ *tenha a duração igual a* j *unidades de tempo. A noção de horizonte passa a significar "unidades de tempo" ao invés do significado usual, "número de decisões". Cada decisão pode consumir uma ou mais unidades de tempo.*

Em um LL-POMDP, a duração da última ação é usada na determinação do próximo estado do autômato). O alfabeto do autômato é $\Sigma = \{< a, o, j > | a \in A, o \in \Omega, j \in \mathbb{N}\}$.

É possível também modificar a definição de LL-POMDPs com duração para que as durações dos eventos dependam também do estado do MDP subjacente (ou seja, $D : S \times A \times \Omega \times \mathbb{N} \mapsto [0, 1]$. Para isto seria necessário modificar o estimador de estados τ de forma que este passe a usar esta informação. Esta modificação, no entanto, fica fora do escopo deste trabalho.

5.3.1 Política

A política ótima para um LL-POMDP com durações é semelhante à de um LL-POMDP, determinando a ação a ser tomada a partir do estado de crença. As durações de eventos são usadas apenas nos algoritmos para determinar a política e para executá-la, e são implicitamente representadas nas transições do autômato $\mathcal{W}$ (o alfabeto do autômato foi extendido para incluir a duração de cada evento).

5.3.2 Algoritmos

O algoritmo de iteração de valores para LL-POMDPs pode ser modificado para levar em conta durações probabilísticas para eventos. O algoritmo de iteração de valores é alterado de forma que, após k iterações (ou seja, construindo a política para o momento em que há k unidades de tempo disponíveis para o tomador de decisões), considere as políticas de diversos momentos à frente, dependendo das durações das ações.

A Figura 5.7 mostra três eventos, e_0, e_1 e e_2, com durações 1, 2 e 4, respectivamente. As arestas mostram quais funções valor das iterações anteriores são usadas para construir a função valor $\Gamma_{[4,0]}$: as funções valor $\Gamma_{[1,0]}$ e $\Gamma_{[1,1]}$ não são usadas, porque não há evento com duração igual a três.

Iteração de valores

Apresentaremos apenas o algoritmo de programação dinâmica para LL-POMDPs, uma vez que a versão para LL-MDPs pode ser intuída trivialmente deste. O operador H modificado é:

$$h(b, a, V, q, i) = \sum_{s \in S} R(s, a) b(s) +$$

$$\gamma \sum_{j \in \mathbb{N}^+} \left\{ D(a, o, j) \sum_{o \in \Omega} Pr(o|b, a) V\left[\tau(b, a, o), \delta(q, < a, o, j >), i - j\right] \right\}$$

$$HV(b, q, i) = \max_{a \in A_q} h(b, a, V, q, i).$$

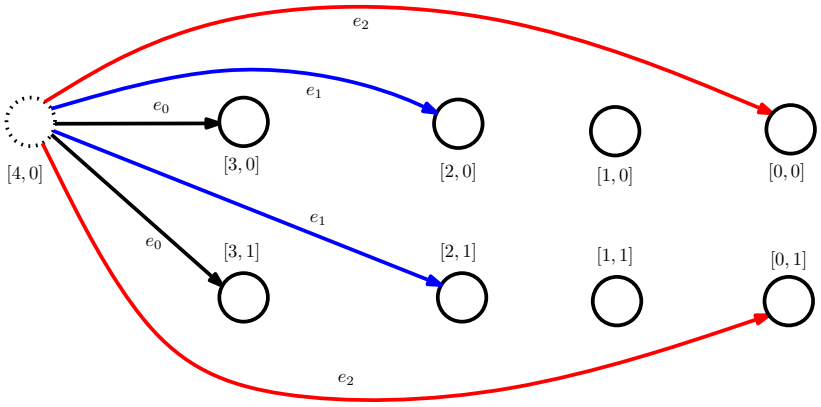


Figura 5.7: Construindo uma política para LL-POMDPs com durações nas ações.

O algoritmo de iteração de valores passa a usar a seguinte fórmula no passo de programação dinâmica:

$$\forall q \in E,\ p \in IN(\mathcal{M}, q),$$

$$\forall a \in A \ \text{ tal que } \ \exists o \in \Omega, \exists \in \mathbb{N},\ \delta(p, < a, o, t >) = q,$$

$$\Gamma^{a}_{[i,p]} = \left\{ r_a + \gamma \sum_{j \in \mathbb{N}^+} \left(D(a,o,j) \sum_{\substack{o \in \Omega \\ \delta(p,<a,o>)=q}} M^{a,o}\alpha \right) \,\middle|\, \alpha \in \Gamma_{[i-j,q]} \right\}. \tag{5.3}$$

$$\Gamma_{[i,p]} = \bigcup_{a \in A} \Gamma^{a}_{[i,p]}.$$

As funções valor das iterações anteriores são usadas e ponderadas de acordo com as probabilidades das durações de eventos.

Para horizonte infinito, um possível critério de convergência para o algoritmo é similar àquele para LL-POMDPs, mas com o cuidado extra de se comparar a função valor do estado q com todas as outras usadas para construí-la (em todas as iterações anteriores):

$$\forall q \in E, j \in \mathbb{N} \ \ C_{q,j}(L) = C(V_{[i,q]}, V_{[i-j,q]}), \tag{5.4}$$

$$C(L) = \bigwedge_{q \in E, j \in \mathbb{N}} C_{q,j}(L). \tag{5.5}$$

Multiplicação de estados

Pode-se multiplicar os estados do LL-POMDP com durações da mesma forma que LL-POMDPs. No entanto, como a duração de um evento pode determinar o próximo estado do autômato, é necessário multiplicar o espaço de estados também pela quantidade de possíveis durações de eventos. O espaço de estados resultante é $S \times E \times \Omega \times \mathcal{T}$, onde $\mathcal{T}$ é o conjunto de todas as possíveis durações de eventos.

5.3.3 Determinação das unidades de tempo

Seja $\mathcal{T}$ o conjunto de todas as durações de eventos, usando a mesma unidade de medida (ou seja, $\mathcal{T} = \{u | \exists a \in A, o \in \Omega, D(a, o, u) \neq 0\}$. Há dois problemas a serem resolvidos:

- As durações, do ponto de vista do algoritmo, devem ser naturais, e não reais;
- O uso de uma unidade de tempo inadequada pode causar iterações desnecessárias e produzir uma política muito grande. Por exemplo, se $\mathcal{T} = \{20000s, 40000s, 60000s\}$, seria inconveniente executar o algoritmo para horizonte acima de 20000, quando é possível dividir os tempos por 20000 e usar $\{1, 2, 3\}$.

Pode-se determinar um conjunto de durações mais adequado a partir de $\mathcal{T}$, da seguinte forma:

$$v = \max_{w \in \mathbb{R}} \{w | \forall t \in \mathcal{T},\ \exists i \in \mathbb{N},\ t = iw\},$$
$$\forall a \in A,\ o \in \Omega,\ \forall u \in \mathcal{T},$$
$$D'\left(a, o, \frac{u}{v}\right) = D(a, o, u).$$

A unidade de tempo é v, e a nova função que dá as durações é D'.

5.3.4 Usando a política

O Algoritmo 5.4 mostra como a política para um LL-POMDP deve ser usada: basta que o tomador de decisões se lembre do estado do autômato, como na política para LL-POMDPs. Neste caso, pode ser que ele vá de um estado para outro que está várias unidades de tempo à frente.

5.4 Sumário

Este capítulo descreve teoricamente algumas das contribuições centrais desta tese:

- A definição de POMDPs limitados por linguagem;
- Dois algoritmos para solução dos novos problemas descritos: um multiplicador de estados e um passo modificado de programação dinâmica;
- Prova de corretude do passo modificado de programação dinâmica;
- Adaptação dos algoritmos para que aceitem a especificação de duração probabilística discreta para eventos.

Entrada: Uma política $(\mathcal{W}, \Gamma)$ onde $\mathcal{W} = (\mathcal{E}, \mathcal{N}, \mathcal{F}, \Delta)$ é um autômato e
$\Gamma = \{\Gamma_q | q \in \mathcal{E}\}$ é um conjunto de funções valor;
Um estado inicial de crença b;
Um horizonte z.

Escolha $q \in \mathcal{N}$
enquanto $q = [k, \cdot],\ 0 \leq k < z$ **faça**
  $a \leftarrow \text{best\_action}(\Gamma_q, b)$
  Execute a, obtendo a próxima observação o após u unidades de tempo
  $q \leftarrow \Delta(q, < a, o, u >)$
  $b \leftarrow \tau(b, a, o)$
fim

Algoritmo 5.4: Usando uma política para LL-POMDPs com duração probabilística.

Capítulo 6

Resultados experimentais

Este capítulo descreve experimentos realizados com algoritmos para solução de LL-MDPs e LL-POMDPs. Os algoritmos implementados são:

- **LL-MDPs**
  - **Multiplicador de estados**: implementamos um programa que multiplica os estados do POMDP usando o autômato $\mathcal{M}$, como descrito no Capítulo 4;
  - **Iteração de valores**: implementamos, em C++, o algoritmo de iteração de valores e sua versão modificada para LL-MDPs, como descrito no Capítulo 4;
- **LL-POMDPs**
  - **Multiplicador de estados**: implementamos um programa que multiplica os estados do POMDP usando o autômato $\mathcal{M}$, como descrito anteriormente. Este programa permite especificar se há ou não observações seletoras no LL-POMDP, e usa um algoritmo simples e mais rápido quando estas não existem;
  - **Iteração de valores**: modificamos o resolvedor de POMDPs de Anthony Cassandra [18] (o `pomdp-solve`, implementado em C) de forma a usar o passo modificado de programação dinâmica descrito no Capítulo 5;
  - **Iteração de valores com durações**: similar ao anterior, mas com as modificações necessárias para que seja possível especificar durações para cada evento, como descrito na Seção 5.3.

Os resultados descritos neste capítulo foram obtidos da seguinte maneira: para cada LL-MDP,

- Usamos o multiplicador de estados e em seguida usamos tanto programação linear como iteração de valores para resolver o problema resultante;

- Usamos o algoritmo de iteração de valores para LL-MDPs (LLVI, cujo passo de programação dinâmica é descrito no Algoritmo 4.2);

e para cada LL-POMDP,

- Testamos diversos algoritmos com o POMDP subjacente;
- Usamos o multiplicador de estados e em seguida usamos diversos algoritmos para resolver o problema resultante;
- Usamos o `pomdp-solve` modificado para LL-POMDPs (LLVI, para POMDPs, cujo passo de programação dinâmica é descrito no Algoritmo 5.2).

Além disso, para cada problema (tanto LL-MDPs como LL-POMDPs) verificamos que:

- Para entradas aleatórias a política gerada usando o multiplicador de estados recomendava as mesmas ações que a política gerada pelo algoritmo de programação dinâmica modificado para LL-POMDPs;
- Os estados de crença intermediários nos algoritmos MS tinham no máximo $|S|$ elementos não zero (como descrito na página 90);
- Os valores não zero mencionados eram os mesmos produzidos ao usar as políticas geradas por algoritmos LL (apenas quando as ações recomendadas eram as mesmas).

Também rodamos nossa implementação de iteração de valores para LL-POMDPs com durações para alguns problemas e verificamos que a política gerada estava correta.

6.1 Problemas modelados como LL-MDPs

Esta seção descreve problemas modelados como LL-MDPs. O problema do robô coletor de rochas é uma variação de um POMDP usado como benchmark por Smith e Simmons [75].

6.1.1 Administrador de sistemas

Esta é uma variante de um problema descrito por Guestrin e outros [31]: um administrador de sistemas deve monitorar n máquinas, e reiniciá-las quando elas falharem. Cada máquina pode falhar espontaneamente com uma pequena probabilidade, e quando uma máquina falha, ela aumenta a probabilidade de as máquinas vizinhas falharem. O administrador de sistemas pode reiniciar apenas uma máquina de cada vez. O objetivo do administrador é maximizar a quantidade de máquinas funcionando ao longo do tempo.

A versão apresentada neste trabalho tem uma restrição adicional: uma máquina só pode ser reiniciada k vezes, e depois deve ser trocada por outra. Para cada n e k um autômato diferente precisa ser criado. O autômato tem número de estados exponencial (mais precisamente, k^n); o MDP tem 2^n estados (um bit para cada máquina, indicando se ela está ligada ou desligada).

6.1.2 Robô coletor de rochas (I)

Um robô está em uma área quadrada de tamanho $L \times L$. Há também r rochas que ele deve coletar e analisar. Quando a análise estiver pronta, ele deve enviar os dados por rádio para uma estação de pesquisa. Como a memória do robô é limitada, ele pode armazenar apenas os dados de k rochas de cada vez, e portanto deve alternar coleta e envio de dados.

As ações disponíveis para o robô são:

- Mover-se nas quatro direções. Estas ações terão sucesso com probabilidade 0.9, e manterão o robô no mesmo estado com probabilidade 0.1;
- Coletar: esta ação coleta a rocha no local onde o robô está;
- Enviar: esta ação envia os dados sobre as últimas k rochas coletadas.

O espaço de estados tem tamanho $L \times L \times 2^r$ (cada combinação de rochas já coletadas e cada posição são representadas). Para forçar a restrição do envio dos dados de k rochas de cada vez, seria necessário multiplicar este espaço de estados por k, para que o estado represente também o número de rochas coletadas e não processadas – ou pode-se modelar o problema como um LL-MDP, usando o autômato da Figura 6.1 (o autômato representado funciona para $k = 3$.

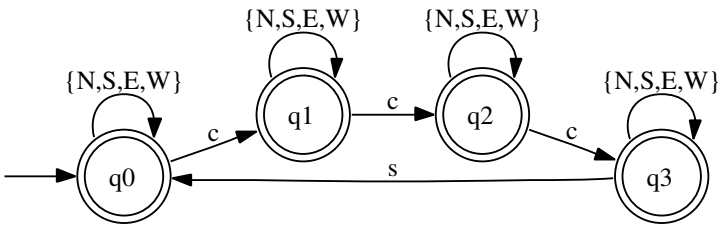


Figura 6.1: Autômato que restringe o número de coletas antes do envio de dados por um robô.

6.2 Problemas modelados como LL-POMDPs

Esta seção descreve os problemas que modelamos como LL-POMDPs. Os problemas do tigre e do robô coletor de rochas foram obtidos na literatura sobre POMDPs [74, 75, 44], e os problemas do labirinto e do tratamento médico foram desenvolvidos para este trabalho.

6.2.1 O problema do Tigre

Modelamos duas versões do problema do tigre, já descrito no Capítulo 3:

- Na primeira variação, exigimos que o agente ouça pelo menos duas vezes antes de abrir uma das portas. Isso é naturalmente modelado como um LL-POMDP: basta usar o POMDP já descrito no Capítulo 3, com restrições adicionais. Para este problema, o autômato usado é o da Figura 6.2;

- Na segunda variação, o agente está em um corredor, e deve dar passos à frente antes de ouvir ou abrir a porta. Neste caso há uma ação a mais (andar para a frente). O autômato da Figura 6.3 foi usado neste problema.

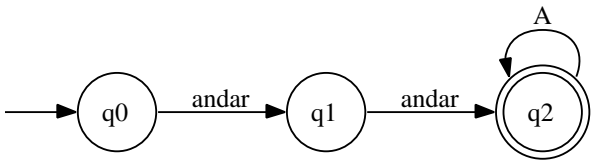


Figura 6.2: Autômato para o problema do tigre, com três estados.

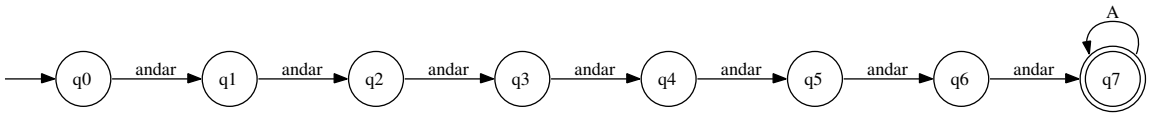


Figura 6.3: Autômato para o problema do Tigre, com oito estados.

6.2.2 Navegação de robô

Um robô está em uma sala da qual conhece o mapa, que a Figura 6.4 mostra. O "R" marca uma recompensa (este é o objetivo do robô), e os quadros escuros são obstáculos. O robô tem cinco sensores:

- Quatro sensores que dizem ao robô se há obstáculos nas quatro direções. Estes sensores podem deixar de funcionar, com probabilidade 0.05, por algum motivo (podem ficar cobertos de líquido, as baterias dos sensores podem descarregar, ou os sensores podem quebrar). Quando um sensor quebrar, o robô saberá que isto aconteceu. Usar os sensores tem um custo para o robô;

- Um sensor que sempre diz ao robô se o último movimento foi bem-sucedido.

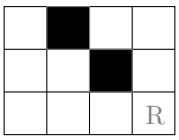


Figura 6.4: O labirinto do exemplo de navegação de robô.

Além de "sentir" nas quatro direções, o robô pode se mover nas mesmas quatro direções.

Modelamos este problema como um LL-POMDP da seguinte forma:

- S: cada posição do labirinto é um estado, como podemos ver na Figura 6.4;

- A: $\{move\_N, move\_S, move\_E, move\_W, sense\_N, sense\_S, sense\_E, sense\_W\}$

- T: Mover na direção de um obstáculo nunca funciona; mover em uma direção livre sempre funciona;

- R: independentemente da direção, mover tem um custo igual a 10, e usar um sensor tem custo igual a 2;

- Ω: $\{obstacle; nothing, broken\_sensor\}$;

- O: qualquer sensor que esteja funcionando dará a resposta certa com probabilidade 0.9; o resultado errado com probabilidade 0.5; e responderá "estou quebrado" com probabilidade 0.5. Depois da primeira vez que anunciar que está quebrado, um sensor sempre dará a mesma resposta. Depois de mover-se para uma direção livre, o robô sempre sabe que obteve sucesso; depois de bater em um obstáculo, o robô recebe a observação "obstáculo" com probabilidade 0.91, e "OK" com probabilidade 0.9.

- $\mathcal{M}$: como na Figura 6.5. Os estados representam sensores quebrados, e os rótulos das transições são omitidos para manter a clareza.

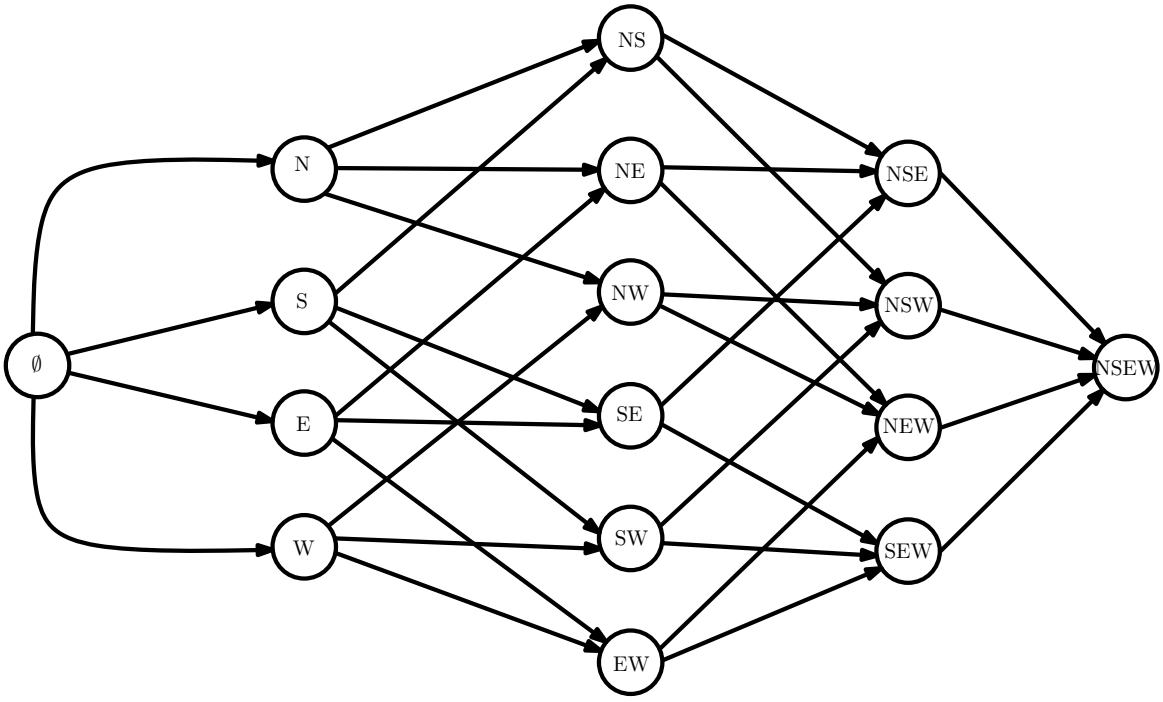


Figura 6.5: Autômato $\mathcal{M}$ usado no exemplo de navegação de robô.

O autômato $\mathcal{M}$ só permitirá ao robô usar o sensor de uma dada direção até que o sensor quebre. Isto significa que a observação "estou quebrado" será seletora (mudará o estado do autômato). O robô poderá, então, usar os sensores até que todos quebrem, e depois disso andar às escuras, percebendo (com alta probabilidade, mas não certeza) quando esbarra em um obstáculo.

6.2.3 Diagnóstico e Tratamento de uma Doença

Um sistema de apoio à decisão deve ajudar no diagnóstico e tratamento de duas doenças similares (doençaA e doençaB), ambas com os mesmos sintomas.

Há três testes, dois para doençaA e um para doençaB. O TesteA não deve ser repetido, uma vez que isto não mudaria o estado de crença (e, sendo um teste, também não afetaria o estado do sistema). Os outros testes podem ser repetidos, já que quanto mais vezes forem feitos, mais preciso o diagnóstico será.

Há três tratamentos, TrataA_1, TrataA_2, e TrataB. Para a doençaA, TrataA_1 demora de quatro a seis meses para completar, custa \$20 e tem eficiência igual a 0.98. TrataA_2 leva de dois a três meses para terminar, custa \$30 e tem eficiência 0.95.

TrataB é o único tratamento para doençaB, com eficácia 0.9 e duração de 3 meses.

O tratamento TrataA_2 não pode ser usado após TrataB, porque poderia levar à morte

imediata do paciente.

Os estados possíveis são Saudável, doençaA e doençaB. As ações (e função de recompensa) estão na Tabela 6.1. As probabilidades de duração são mostradas no formato "$d(p)$", onde d é a duração e p é a probabilidade de que a ação leve aquele tempo para terminar.

Ação	Custo	Durações e probabilidades
TrataA_1	\$20	24(0.7) 16(0.3)
TrataA_2	\$30	8(0.9) 12(0.1)
TrataB	\$25	3(1)
TesteA	\$4	1(1)
TesteA'	\$6	1(1)
TesteB	\$5	1(1)

Tabela 6.1: Ações e recompensas para o exemplo de diagnóstico e tratamento médico.

A Tabela 6.2 mostra a função de transição. Transições que não mudam o estado corrente são omitidas na tabela.

	TrataA_1	TrataA_2	TrataB	TesteA	TesteA'	TesteB
Saudável						
doençaA	Saudável(0.98)	Saudável(0.95)				
doençaB			Saudável(0.9)			

Tabela 6.2: Função de transição para o exemplo de diagnóstico e tratamento médico.

As observações são Positivo, Negativo, Sintomas e Nada, e a função de probabilidade de observação está na Tabela 6.3. Neste exemplo, todas as observações levam o mesmo tempo para serem obtidas, por isso as durações não são mostradas aqui. Entradas vazias na tabela significam que a probabilidade da observação é um.

	TrataA_1	TrataA_2	TrataB
Saudável	Nada	Nada	Nada
diseaseA	Sintomas	Sintomas	Sintomas
diseaseB	Sintomas	Sintomas	Sintomas

	TesteA	TesteA'	TesteB
Saudável	Positivo(0.02)	Positivo(0.001)	Positivo(0.009)
diseaseA	Positivo(0.999)	Positivo(0.91)	Positivo(0.05)
diseaseB	Positivo(0.07)	Positivo(0.001)	Positivo(0.97)

Tabela 6.3: Função observação para o exemplo de diagnóstico e tratamento médico.

O autômato para este LL-POMDP, na Figura 6.6, não permite sequências de ações onde TrataA_2 segue TrataB, e só permite que TesteA seja feito uma única vez.

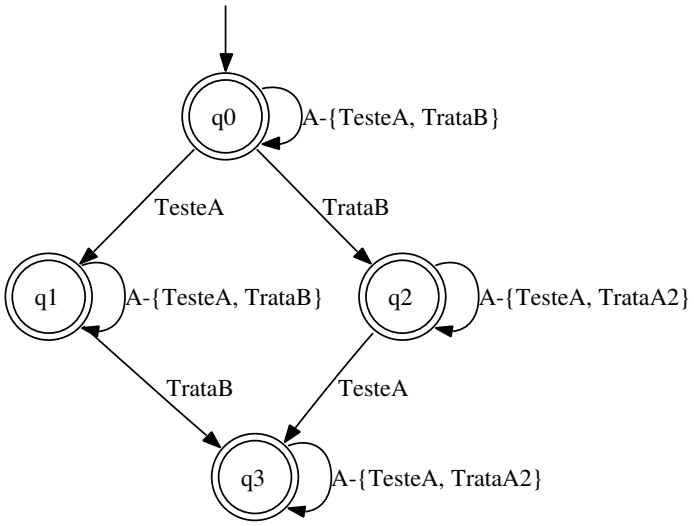


Figura 6.6: O autômato $\mathcal{M}$ para o exemplo de diagnóstico e tratamento médico.

6.2.4 Robô coletor de rochas (II)

Este problema é apresentado por Trey Smith em sua exposição sobre o HSVI [75].

Um robô é mandado a um planeta para coletar amostras de minério em uma determinada região. A posição das amostras é conhecida, porque foi fotografada. No entanto, não há como saber quais amostras tem algum valor científico, e o robô precisa testar várias delas, uma a uma.

Além de se mover nas quatro direções, o robô pode:

- Tentar identificar o valor de uma amostra usando um sensor. A precisão do sensor decai com a distância entre o robô e a amostra, de acordo com $2^{-d/d_0}$ (d_0 é uma constante ajustável);
- Coletar uma amostra.

Há apenas duas observações, *Good* e *Bad*. Após um movimento, a observação diz se o movimento foi bem sucedido ou não (a movimentação do robô é completamente observável). Após usar o sensor, as observações dizem se a amostra tem valor ou não (com ruído).

O robô deve andar por uma área de 4×4 posições, e há quatro rochas a serem coletadas (das quais o robô conhece as posições). O espaço de estados representa as 16 posições

do robô e também as 2^4 possibilidades diferentes para as rochas já coletadas (quando o robô coleta uma rocha, ele passa para os estados que representam que esta rocha já foi coletada).

Modelamos este problema com uma restrição adicional: o robô só pode coletar 2 das 4 rochas. Esta restrição é feita como mostra o autômato 6.7 (obviamente, após a última coleta não há mais necessidade de verificar minério, então as ações de "*sampling*" também são excluídas no último estado). Este autômato não muda o problema original, e funciona apenas como um acelerador para o algoritmo de programação dinâmica.

Este problema é interessante tanto com horizonte infinito (o robô termina ao chegar a um estado final) como com horizonte limitado (o robô analisa a rocha e envia dados de volta por rádio, mas quando sua bateria acabar, ele para de funcionar).

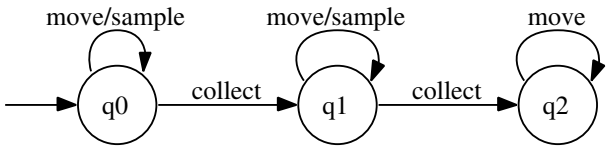


Figura 6.7: Autômato definindo número máximo de rochas coletadas pelo robô exploratório.

6.3 Representação dos LL-MDPs

A Tabela 6.4 mostra o número de estados dos MDPs modelados, incluindo os MDPs resultantes de multiplicação de estados. S^M é o conjunto de estados do POMDP resultante da multiplicação. Nesta tabela, "Robô X,Y,Z" é o problema do robô coletor de rochas onde a área é um quadrado de lado X, há Y rochas e apenas Z delas podem ser coletadas de cada vez. "AdmSis X,Y" é o problema do administrador de sistemas com X máquinas que podem ser reiniciadas no máximo Y vezes.

Problema	$\|S\|$	$\|E\|$	$\|\delta\|$	$\|S^M\|$
Robô 4,4,2	256	5	25	1281
Robô 5,5,2	800	5	25	4001
AdmSis 3,4	8	64	333	513
AdmSis 4,3	16	81	540	1297
AdmSis 4,4	16	256	1788	4097
AdmSis 5,2	32	32	235	1025
AdmSis 6,2	64	64	570	4097

Tabela 6.4: Número de estados nos LL-MDPs dos experimentos.

Para os problemas AdmSis com n máquinas e máximo de k reinicializações, o número de estados é 2^n no LL-MDP e k^n no autômato (2^n+k^n). O MDP com estados multiplicados tem $2^n k^n$ estados.

6.4 Tempo de execução para LL-MDPs

Todos os tempos foram tomados em um Athlon 1.3 GHz com 1.1Gb de memória RAM; o compilador usado foi o mesmo e os mesmos parâmetros de otimização foram usados em todas as compilações. Todos os números mostrados são médias de cinco tomadas de tempo para o mesmo experimento, exceto para os casos em que o processo teve que ser interrompido por exceder o limite estabelecido (43200 segundos). A medida de tempo foi realizada usando o tempo reportado pelos próprios programas, que medem o tempo de CPU que usam em modo de usuário.

Os algoritmos usados foram:

- **MS+LIN**: Multiplicação de estados seguida da transformação em programa linear e solução deste;
- **MS+VI**: Multiplicação de estados seguida de iteração de valores;
- **LLVI**: Iteração de valores modificada para funcionar com LL-MDPs.

Em ambos os casos de "VI", a iteração de valores foi implementada de forma "ingênua": o passo de programação dinâmica percorre, para cada estado, todos os estados, sem qualquer tentativa de mudar a ordem destes. Já o resolvedor de programas lineares é o `lp-solve` [12], versão 5.5.

6.4.1 Noção de convergência

Em todos os experimentos com MDPs e LL-MDPs, a convergência dos algoritmos de iteração de valores foi verificada usando a equação 2.3, com ϵ igual a 10^{-9}. O algoritmo adaptado para LL-MDPs só termina após todas as funções valor terem convergido (conforme as equações 4.2 e 4.3).

6.4.2 Horizonte finito

A Tabela 6.5 mostra o tempo de execução dos algoritmos para LL-MDPs com horizonte finito (todos os tempos são em segundos).

Problema	Horizonte	MS+VI	LLVI
Robô 4,4,2	10	25.94	5.18
Robô 4,4,2	50	131.34	26.07
Robô 5,5,2	10	271.58	53.06
Robô 5,5,2	50	1331.90	266.37
AdmSis 3,4	10	4.95	0.09
AdmSis 3,4	50	24.48	0.46
AdmSis 4,3	10	42.37	0.5
AdmSis 4,3	50	211.74	2.6
AdmSis 4,4	10	454.05	2.31
AdmSis 4,4	50	1893.32	11.61
AdmSis 5,2	10	32.99	0.78
AdmSis 5,2	50	163.42	3.93
AdmSis 6,2	10	707.52	7.83
AdmSis 6,2	50	2740.43	38.91

Tabela 6.5: Desempenho dos algoritmos para a solução de LL-MDPs com horizonte finito.

6.4.3 Horizonte infinito

A Tabela 6.6 mostra o tempo de execução dos algoritmos de iteração de valores, puro e modificado, para LL-MDPs com horizonte infinito. O fator de desconto γ para o problema do robô coletor de rochas é o mesmo usado por Trey Smith na descrição original do problema como POMDP. O fator de desconto para o problema do administrador de sistemas é o mesmo usado por Wingate e Seppi [85].

Problema	γ	MS+LIN	MS+VI	LLVI
Robô 4,4,2	0.95	1.73	327.91	67.87
Robô 5,5,2	0.95	14.05	2979.95	681.25
AdmSis 3,4	0.95	3.01	261.13	4.91
AdmSis 4,3	0.95	47.33	2299.2	26.94
AdmSis 4,4	0.95	850.77	2942.62	125.07
AdmSis 5,2	0.95	33.13	1686.87	40.37
AdmSis 6,2	0.95	1722.25	25537.8	411.89

Tabela 6.6: Desempenho dos algoritmos para a solução de LL-MDPs com horizonte infinito.

6.5 Representação dos LL-POMDPs

A Tabela 6.7 mostra o número de estados dos POMDPs modelados, incluindo os POMDPs resultantes de multiplicação de estados. S^M é o conjunto de estados do POMDP resultante da multiplicação.

Problema	$\|S\|$	$\|\Omega\|$	$\|E\|$	$\|\delta\|$	$\|S^M\|$
Robô	10	3	15	276	451
Tigre	2	2	3	10	7
Tigre8	2	2	8	21	17
Médico	3	4	4	81	13
Rochas	257	2	3	44	772

Tabela 6.7: Número de estados nos LL-POMDPs dos experimentos.

6.6 Tempos de execução para LL-POMDPs

Esta seção discute experimentos realizados com algoritmos para a solução de LL-POMDPs. Os tempos foram tomados da mesma forma que os tempos apresentados para LL-MDPs. As tabelas nas próximas subseções listam todos os tempos para todos os métodos, mas muitos deles não são comparáveis: os problemas puros (sem MS ou LL) só podem ser comparados com problemas limitados por linguagem se o autômato tinha o único propósito de otimizar a busca pela política – de outra forma, dois problemas diferentes estariam sendo comparados. Para o problema do tratamento médico, os algoritmos LL são os únicos que suportam duração nas ações, e portanto só podem ser comparados entre si.

6.6.1 Noção de convergência

O `pomdp-solve` implementa três critérios de convergência: resíduo de Bellman, convergência fraca e convergência exata (descritos na subseção 3.4.1). O critério usado em todos os experimentos com o pomdp-solve é o da "convergência fraca". O *default* para o parâmetro de entrada para convergência é 10^{-9}, e não foi mudado.

Para o HSVI, o critério implementado por Smith para determinar a convergência é a diferença entre os limites máximo e mínimo (porque o HSVI mantém tanto um *lower bound* como um *upper bound*). O programa para quando a diferença entre esses limites é menor que um valor ϵ *para todos os estados de crença*. O valor default é 10^{-3}, mas foi mudado para 10^{-9}.

Os algoritmos adaptados para LL-POMDPs (LL-GRID e LLVI) só terminam após todas as funções valor terem convergido (conforme as equações 5.4 e 5.5).

6.6.2 Horizonte finito

Os tempos de execução dos algoritmos para horizonte finito estão descritos na Tabela 6.8. As colunas da tabela mostram a duração medida da resolução do LL-POMDP ou do POMDP subjacente usando as seguintes implementações:

- **INCPRUNE**: *incremental pruning* puro;
- **GRID**: método baseado em pontos implementado pelo `pomdp-solve`. A grade foi inicializada com a crença uniforme e uma busca em largura foi realizada em seguida.
- **MS+INCPRUNE**: multiplicação de estados seguida da resolução com o algoritmo de *incremental pruning* (a solução gerada é exata);
- **MS+GRID**: multiplicação de estados seguida da resolução com o algoritmo baseado em pontos de crença implementado pelo `pomdp-solve`;
- **LL-INCPRUNE**: algoritmo modificado de programação dinâmica e *incremental pruning* para o o passo de melhoria;
- **LL-GRID**: algoritmo modificado de programação dinâmica e o algoritmo baseado em pontos de crença do `pomdp-solve` para o passo de melhoria. A grade foi inicializada da mesma forma que na implementação GRID, e não usamos a busca restrita por pontos de crença. Isto implica no cálculo de mais pontos de crença do que o necessário, mas o algoritmo continua correto (as funções valor para cada estado do autômato são menores do que a função valor ótima para estes estados);

Dentro da tabela:

- As entradas em branco são aquelas onde não fazia sentido obter uma política ótima para o POMDP subjacente, ignorando o autômato $\mathcal{M}$;
- As entradas do tipo "$T_1(T_2/I)$" indicam que o tempo para executar todas as iterações foi T_1, mas na verdade os valores convergiram após I iterações, e portanto o tempo a ser considerado é T_2;
- As entradas com "> 43200" são casos onde a execução demorou mais que doze horas, e tivemos que interromper o processo.

Problema	Horizonte	INCPRUNE	GRID	MS+INCPRUNE	MS+GRID	LL-INCPRUNE	LL-GRID
Robô	2	0.01	0.17	36.48	10.97	0.14	6.68
Robô	3	133.73	0.31	> 43200	37.36	0.62	10.88
Robô	4	> 43200	0.57	> 43200	101.87	2.71	16.59
Robô	5	> 43200	1.08	> 43200	196.45	19.34	25.89
Robô	100	> 43200	103.67 (98.60,95)	> 43200	> 43200	6234.03 (1956.02,16)	3033.78
Tigre	10			1.57	0.09	1.21	0.06
Tigre	100			33.35	1.43	20.89	0.79
Tigre	500			60.90	7.39	38.38	4.71
Tigre	1000			96.63	14.97	67.03	12.86
Tigre	2000			169.90	31.25	147.33	40.43
Tigre8	10			45.75	0.10	1.18	0.12
Tigre8	100			14227.73	1.40	29.36	1.52
Tigre8	500			> 43200	7.38 (3.10, 221)	84.47	16.29
Tigre8	1000			> 43200	15.08	203.67	55.89
Médico	30					1325	31.63
Médico	50					> 43200	56.02
Rochas	10			> 43200	> 43200	> 43200	618.02

Tabela 6.8: Desempenho dos algoritmos para a solução de LL-POMDPs com horizonte finito.

6.6.3 Horizonte infinito

Para horizonte infinito, usamos as seguintes implementações:

- **INCPRUNE**: *incremental pruning*
- **GRID**: algoritmo baseado em pontos de crença do `pomdp-solve`;
- **PBVI**: PBVI de Pineau;
- **HSVI**: HSVI de Smith e Simmons;
- **MS+INCPRUNE**: multiplicação de estados mais INCPRUNE;
- **MS+GRID**: multiplicação de estados mais GRID;
- **MS+PBVI**: multiplicação de estados mais PBVI;
- **MS+HSVI**: multiplicação de estados mais HSVI;
- **LL-INCPRUNE**: programação dinâmica LL com INCPRUNE;
- **LL-GRID**: programação dinâmica LL com GRID.

A Tabela 6.9 mostra os tempos de execução. As colunas POMDP DP e HSVI referem-se ao POMDP subjacente; A coluna "MS+HSVI" refere-se ao POMDP com estados multiplicados resolvido com HSVI. Os tempos são em segundos: cada entrada mostra o tempo que o algoritmo demorou para atingir o critério de convergência, e "> 43200" significa que o algoritmo não convergiu para alguma solução após 43200 segundos.

As entradas são da forma:

- GRID,INCPRUNE: "T (I)", onde T é o tempo de execução, I é o número de iterações antes da convergência;
- PBVI: "T (I){B}", onde T é o tempo de execução, I é o número de iterações antes da convergência, e B é o tamanho do conjunto de pontos de crença (e consequentemente o número de vetores α da política);
- LL-GRID, LL-INCPRUNE: "T (I/E)", onde T é o tempo de execução, I é o número de iterações (passos de programação dinâmica) e E é o número de épocas de decisão processadas. Cada época de decisão pode envolver mais de um passo de programação dinâmica (porque os passos são separados para cada transição do autômato).

O método MS+INCPRUNE aplicado no problema Tigre8 não terminou no prazo estabelecido, mas o número de vetores se estabilizou em 83 na iteração 131, após 4118 segundos. O mesmo aconteceu com o método MS+GRID quando aplicado ao problema do diagnóstico médico: após 98.12 segundos e 123 iterações, o número de vetores se estabilizou em 61. Em ambos os casos, o erro residual calculado era menor que 10^{-5}.

Para o problema do tratamento médico, apenas o tempo dos algoritmos LL é mostrado; neste caso, a variante do LL é aquela que suporta a duração de ações.

6.7 Discussão

Discutimos agora os resultados, sob dois pontos de vista: os LL-MDPs e LL-POMDPs como formas de otimizar os MDPs/POMDPs subjacentes e em seguida, como ferramenta de modelagem.

6.7.1 Limitações como otimizações

A multiplicação de estados implica também na multiplicação do tamanho da representação da política: cada vetor, que originalmente tinha tamanho $|S|$, passa a ter tamanho $|S||E|$ (para LL-POMDPs com observações seletoras, $|S||E||\Omega|$).

Em um LL-MDP, o número de estados percorrido a cada iteração pode diminuir, da mesma forma como o número de vetores α para LL-POMDPs (os tempos dos experimentos com LL-MDPs mostram isso claramente). No entanto, isso depende da estrutura do autômato: para LL-MDPs, o algoritmo LLVI será vantajoso se houver poucas transições entre os estados (porque cada transição é um passo a mais de programação dinâmica). Para o problema do robô coletor de rochas modelado como LL-MDP, o autômato que proíbe movimentos inúteis deixou a busca pela política mais lenta, porque adiciona muitas arestas ao autômato – já o autômato que separa logicamente o número de rochas já recolhidas acelerou o processo.

Para LL-POMDPs, os conjuntos de vetores são maiores quando os estados são multiplicados, porque as funções valor para todos os estados do autômato são representadas em um só conjunto de vetores. Como a quantidade de vetores pode crescer de forma duplamente exponencial, como descrito no Capítulo 3, a abordagem da multiplicação de estados pode se tornar inviável para problemas com muitos estados e observações ou com autômatos com muitos estados. O algoritmo que representa as funções-valor separadamente por estado do autômato tem a vantagem de manter vetores pequenos e em menor número para passos exatos de programação dinâmica. Esta vantagem, no entanto, só existe quando não há no autômato muitas arestas entre cada par de estados adjacentes no autômato $\mathcal{W}$ (isto já foi discutido no Capítulo 5).

Problema	γ	INCPRUNE	GRID	PBVI	HSVI
Robô	0.75	> 43200	98.60(95)	0.38(1460){64}	< 1

Problema	γ	MS+INCPRUNE	MS+GRID	MS+PBVI	MS+HSVI	LL-INCPRUNE	LL-GRID
Robô	0.75	> 43200	> 43200	102.14(1385){64}	64	1956.02(16/912)	> 43200
Tigre	0.75	24.31 (125)	0.38 (119)	0.02(274){31}	< 1	20.26 (80/239)	0.48(80/239)
Tigre8	0.75	> 43200 (—)	1.64 (118)	0.28(277){35}	< 1	27.43 (79/611)	0.87(79/611)
Médico	0.75					> 43200	106.75(91/728)
Rochas	0.95	> 43200	> 43200	7633(673){64}	12	> 43200	> 43200

Tabela 6.9: Desempenho dos algoritmos para a solução de LL-POMDPs com horizonte infinito.

No problema da navegação, o LL-INCPRUNE convergiu com apenas 16 iterações (e na verdade, com as restrições do autômato não há como o robô andar mais do que 8 vezes sem passar pelo mesmo estado – mas há a possibilidade de o robô andar em ciclos, com os movimentos N, E, S, W, por exemplo). Já o LL-GRID apresentou problemas para convergir.

Os algoritmos PBVI e HSVI constam na tabela apenas para que seja possível uma comparação, mas deve ficar claro que nenhum deles funciona para horizonte finito, e o HSVI não poderia ser modificado em um "LL-HSVI", como já discutido no Capítulo 5.

6.7.2 Limitações como ferramenta de modelagem

Para os problemas onde as limitações do autômato eram parte da modelagem, as colunas relevantes na tabela de tempos são as que mostram "MS" e "LL".

Até onde sabemos, o processo de modelagem de MDPs e POMDPs não é feito com autômatos ou outras ferramentas matemáticas exceto diagramas de influência e ferramentas de decomposição. As situações em que há a necessidade de impor ordem em ações normalmente são modeladas de forma *ad-hoc* (como no trabalho de Peek em diagnóstico e tratamento médico [56]) ou apresentando o MDP ou POMDP já multiplicado. O uso de autômatos oferece as seguintes vantagens sobre estas abordagens:

- O número de estados após a multiplicação é muito grande, como visto na Tabela 6.7, tornando a modelagem manual praticamente impossível. Programas que geram MDPs ou POMDPs já multiplicados têm que ser refeitos para cada problema. O uso de LL-MDPs e LL-POMDPs permite representar as restrições separadamente, e mudá-las sem que se tenha que dar atenção ao problema subjacente;

- O autômato $\mathcal{M}$ pode ser facilmente combinado com outro, $\mathcal{M}'$. Por exemplo, pode-se obter as linguagens $L(\mathcal{M}) \cup L(\mathcal{M}')$ e $L(\mathcal{M}) \cap L(\mathcal{M}')$, ou outra que seja conveniente. Estas modificações podem ser realizadas apenas nos autômatos, sem que a modelagem do problema subjacente tenha que ser repensada.

6.7.3 LL-POMDPs com duração para eventos

Implementamos o algoritmo LL-INCPRUNE com duração para eventos, e o usamos para encontrar políticas ótimas para o problema do diagnóstico médico descrito neste capítulo. A Tabela 6.10 mostra o tempo necessário para obter políticas de horizontes 30 e 50.

A Tabela 6.11 mostra a evolução do número de vetores α gerados a cada iteração do algoritmo. Na iteração 13 o algoritmo passou a considerar a ação TrataA_2. Antes disso ela não era considerada, porque havia a possibilidade dela durar 12 unidades de tempo, e

Horizonte	LL-INCPRUNE	LL-GRID
30	1325	31.63
50	> 43200	56.02

Tabela 6.10: Tempos de execução para obtenção de políticas de horizonte finito com durações nas ações.

o horizonte seria excedido. Como a recompensa desta ação é alta (porque pode levar ao estado saudável), diversos outros vetores dominados foram eliminados nesta iteração.

Época	Vetores	Época	Vetores	Época	Vetores
1	6	11	2368	21	553
2	6	12	1958	22	561
3	10	13	588	23	559
4	28	14	509	24	560
5	62	15	493	25	640
6	138	16	473	26	700
7	262	17	492	27	668
8	530	18	496	28	629
9	1016	19	519	29	596
10	1714	20	536	30	637

Tabela 6.11: Número de vetores α por iteração ao resolver o problema do diagnóstico.

6.8 Sumário

Este capítulo apresentou alguns problemas modelados como LL-MDPs e LL-POMDPs e os tempos de execução para resolvê-los, usando duas abordagens: a multiplicação dos estados do MDP (ou POMDP), e a modificação do passo de programação dinâmica. Com isto, o capítulo confirma a validade e viabilidade das contribuições da tese:

- Mostramos que a solução de LL-MDPs e LL-POMDPs (inclusive com duração probabilística para eventos para LL-POMDPs) de horizonte finito é viável;
- Para um experimento com LL-MDPs, o algoritmo LLVI mostrou-se melhor que a multiplicação de estados;
- Para LL-POMDPs sem durações, o algoritmo LL-INCPRUNE se mostrou muito melhor do que a abordagem da multiplicação de estados do POMDP subjacente com INCPRUNE puro. Mostramos que a partir de um POMDP, o LL-POMDP

otimizado (como no caso da navegação de robô) pode ser resolvido em menos tempo (e instâncias maiores podem ser resolvidas);

- Para os algoritmos baseados em pontos de crença, notamos que em alguns casos a multiplicação de estados é melhor, e em outros o algoritmo LL-GRID é melhor;
- Apontamos algumas vantagens da modelagem usando autômatos.

Os problemas modelados são listados na Tabela 6.12. Os problemas com uma única observação são completamente observáveis (MDPs). A coluna "Obs. Sel." diz se o problema tinha observações seletoras; a coluna "Restrições" diz que tipo de restrição o autômato impõe: (O) é uma otimização, (M) é uma modificação no problema original.

Problema	$\|S\|$	$\|A\|$	$\|\Omega\|$	$\|E\|$	Obs. Sel.	Restrições	Durações
Robô 4,4,2	256	6	1	5		M	
Robô 5,5,2	800	6	1	5		M	
AdmSis 3,4	8	7	1	64		M	
AdmSis 4,3	16	9	1	81		M	
AdmSis 4,4	16	9	1	256		M	
AdmSis 5,2	32	11	1	32		M	
AdmSis 6,2	64	13	1	64		M	
Robô	10	8	3	15	$\surd$	O	
Tigre	2	3	2	3		M	
Tigre-8	2	4	2	8		M	
Médico	3	6	4	4		M	$\surd$
Rochas	257	9	2	3		M	

Tabela 6.12: Problemas modelados para experimentos.

Capítulo 7

Trabalhos Futuros

Este capítulo apresenta idéias relacionadas a LL-MDPs e LL-POMDPs que ainda não implementamos e não temos prova de que estejam corretas. As idéias encontram-se em diferentes estágios de desenvolvimento.

Para LL-POMDPs, muitas das idéias neste capítulo só não foram ainda implementadas porque a implementação original dos algoritmos não está disponível, e a implementação levaria tempo demais (o `pomdp-solve` de Tony Cassandra tem mais de dez mil linhas de código, e o `zmdp` de Trey Smith tem mais de nove mil). A Tabela 7 mostra os algoritmos descritos nesta tese e as implementações a que tivemos acesso. A coluna "LL" diz se pudemos diretamente adaptar o algoritmo para trabalhar com LL-POMDPs: um "S" significa que adaptamos e testamos a implementação; um "T" significa que temos o trabalho teórico pronto; um "N" significa que a adaptação pode ser possível, mas não fizemos o trabalho teórico ainda. A coluna "Fonte" diz se tivemos acesso ao código fonte da implementação. A última coluna é uma referência à implementação.

Algoritmo	Horizonte	Exato	LL	Fonte	Ref. Implementação
It. de valores	F/I	S	S	S	Anthony Cassandra [18]
PolCa	F/I	N	T	N	
It. de políticas (Hansen)	I	N	T	N	
BPI	I	N	T	N	
RTDP	I	N	T	S	Trey Smith [73]
PBVI	I	N	T	S	Joelle Pineau*
HSVI	I	N	N	S	Trey Smith [73]

Tabela 7.1: Implementações de algoritmos para solução de POMDPs.

O PBVI que temos é uma adaptação feita por Pineau no `pomdp-solve` de Anthony Cassandra. A versão adaptada é a 4.0 (diferente da versão que usamos para nossas modificações). A implementação do RTDP mencionada na tabela é diferente da idéia

original de Bonet e Geffner, incorporando um trabalho de otimização do algoritmo por Trey Smith [76].

7.1 Decomposição hierárquica

A combinação dos algoritmos (LL-MDP-DP com PolCa, ou LL-POMDP-DP com PolCa+) permitiria a solução de instâncias grandes de LL-MDPs e LL-POMDPs.

Os LL-MDPs e LL-POMDPs podem ser decompostos hierarquicamente pelos algoritmos PolCa e PolCa+, com apenas uma restrição a mais: eventos em transições diferentes, vindo de um mesmo estado de $\mathcal{M}$, não podem ser agrupados em uma mesma subtarefa. Cada subtarefa é então resolvida como um LL-MDP ou LL-POMDP, usando o mesmo autômato.

Durante a execução da política, o tomador de decisões se lembrará do estado do autômato e escolherá recursivamente a ação indicada para aquele estado.

LL-POMDPs com durações para eventos também podem ser decompostos. A política gerada por LL-POMDPs com durações para eventos pode ser executada da mesma forma que LL-POMDPs usando o PolCa+, exceto que o tomador de decisões também levará em conta o número de unidades de tempo passadas desde a última decisão. Sabemos como usar o PolCa+ para decompor POMDPs, mas não pudemos obter a implementação original.

7.2 Processos de decisão Semi-Markovianos

Os SMDPs (processos de decisão semi-Markovianos) são bem conhecidos e há algoritmos eficientes de programação dinâmica para resolvê-los [69]. Já os POSMDPs são mais difíceis, mas há heurísticas e aproximações para POSMDPs (como visto na seção 3.5). O algoritmo apresentado nesta tese para LL-POMDPs com durações pode ser usado como heurística para resolver POSMDPs. No entanto, este algoritmo funciona apenas para ações com durações discretas, não sendo necessariamente a melhor heurística.

A adaptação dos SMDPs e POSMDPs de forma a definir LL-SMDPs e LL-POSMDPs e o desenvolvimento de algoritmos para estes novos problemas seria interessante. Os LL-SMDPs podem ser resolvidos por qualquer algoritmo adaptado, da mesma forma que os LL-MDPs. Já os LL-POSMDPs dependem da adaptação de heurísticas para POSMDPs.

7.3 Autômatos temporizados

Há diversas situações em que um tomador de decisão tem um prazo (ou carência) para executar uma ação. Se o prazo expirar, o estado atual do autômato deve mudar para outro que proíba a ação; antes de terminar o período de carência, certa ação não pode ser executada. Isso seria possível definindo, por exemplo, LL-MDPs com autômatos temporizados [5, 4, 6]. Um autômato temporizado tem um conjunto de relógios, e o estado atual depende não apenas da sequência de ações executada, mas também dos relógios.

7.4 Linguagens livres de contexto

Os LL-MDPs e LL-POMDPs foram definidos nesta tese usando linguagens regulares e autômatos finitos. Há situações onde se queira restringir sequências de ações de forma que formem linguagem livre de contexto [40]. Por exemplo:

- Uma sequência de ações pode ter que ser desfeita na ordem reversa em que as ações foram executadas;
- Uma determinada ação deve ser executada para cada vez que uma observação é notada, mesmo que não imediatamente;
- Uma sequência de ações pode resultar em uma sequência de observações, na ordem inversa. Um autômato com pilha pode relacionar, então, cada ação com sua observação.

Para estas situações, os algoritmos para LL-POMDPs poderiam ser adaptados para usar autômatos com pilha (os algoritmos LLVI podem guardar em cada estado, além de uma função valor, o "estado simulado da pilha").

7.5 Mais algoritmos

Um trabalho interessante é a investigação de outros algoritmos para MDPs e POMDPs, tentando adaptá-los para que funcionem com LL-MDPs e LL-POMDPs. O BPI e o FRTDP são particularmente interessantes por apresentarem bom desempenho na prática.

7.6 Compressão de estados

Há algoritmos que permitem a representação de POMDPs de forma compacta. Um deles é o *Value-Directed Compression*, de Pascal Poupart [65, 66]. Estes algoritmos permitem

acelerar a busca por uma solução ótima, além de permitir a solução de instâncias maiores de problemas.

7.7 Sumário

Este capítulo mostra trabalhos em andamento relacionados a LL-POMDPs:

- Decomposição hierárquica de LL-POMDPs;
- Definição de LL-SMDPs e LL-POSMDPs, e desenvolvimento de algoritmos para estes;
- Adaptação dos LL-POMDPs para usar autômatos temporizados, e desenvolvimento de algoritmos para estes problemas;
- Adaptação dos LL-POMDPs para linguagens livres de contexto e desenvolvimento de algoritmos para estes problemas;
- Adaptação de mais algoritmos;
- Uso de compressão de estados.

Capítulo 8

Conclusão

A motivação inicial para esta tese foi uma experiência anterior na modelagem do processo de diagnóstico e tratamento de enfermidades [57]. Tradicionalmente, a especificação de uma ordem necessária para ações exige a manipulação do modelo de forma não-intuitiva e *ad-hoc*. Esta tese aponta para o fato destes *frameworks* serem suficientes mas inadequados para a modelagem de muitas situações; mostra então como generalizá-los de forma a beneficiar tanto o processo de modelagem como, em alguns casos, a busca de políticas ótimas. As contribuições deste trabalho são:

- A definição dos MDPs e POMDPs limitados por linguagem. A relevância destes foi indicada com exemplos: há problemas que não são facilmente modelados como MDPs, mas que podem ser modelados de forma mais simples como LL-MDPs. A modelagem do processo de Markov e a modelagem da ordem em que as ações acontecem passam a ser processos distintos;

- A elaboração de duas formas de resolver LL-MDPs e LL-POMDPs:

  - Uma forma de adaptação de algoritmos já existentes para resolver LL-MDPs e LL-POMDPs;

  - Algoritmos que transformam LL-MDPs e LL-POMDPs em MDPs e POMDPs tradicionais, aumentando o espaço de estados;

- Como consequência da apresentação do algoritmo de multiplicação de estados, este trabalho também mostra que LL-MDPs e LL-POMDPs não são mais expressivos do que MDPs e POMDPs;

- A definição de LL-POMDPs onde os eventos podem ter duração probabilística discreta, e a elaboração de algoritmos para a solução destes;

- A implementação de alguns dos algoritmos descritos:
  - A multiplicação de estados para LL-MDPs e LL-POMDPs;
  - O algoritmo LL-VI para LL-MDPs;
  - Os algoritmos LL (para *incremental pruning* e grade finita) para LL-POMDPs;

  Estas implementações foram usadas em um estudo experimental.

A separação da definição do processo de Markov e da definição da ordem de ações (e observações, no caso dos LL-POMDPs) facilita o processo de modelagem, porque permite que os dois aspectos sejam tratados de forma independente e permite o reuso do POMDP subjacente e do autômato.

Quanto ao desempenho dos métodos para solucionar LL-MDPs e LL-POMDPs, a escolha entre multiplicação de estados e algoritmos modificados depende da situação, e os experimentos realizados mostram isto claramente. Para LL-MDPs, o uso do algoritmo LL-VI pode valer a pena quando há poucas arestas entre cada par de estados do autômato; para LL-POMDPs, o LL-INCPRUNE pode ser uma boa opção para horizonte finito, especialmente quando a precisão da solução é importante.

Esta tese mostra também diversos trabalhos em andamento, funcionando também como um projeto de futuras pesquisas. Muitos destes projetos já têm sua relevância indicada no Capítulo 7, que lista também o estado atual de cada um deles.

Referências Bibliográficas

[1] Douglas Aberdeen. *Policy-Gradient Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, Australian National University, 2003.

[2] Douglas Aberdeen. A (revised) survey of approximate methods for solving Partially Observable Markov Decision Processes. Technical report, National ICT Australia, Canberra, Australia, 2003.

[3] Douglas Aberdeen and Jonathan Baxter. Scaling internal-state policy-gradient methods for POMDPs. In *International Conference on Machine Learning*, 2002.

[4] Rajeev Alur. Timed automata. In *Computer Aided Verification*, pages 8–22, 1999.

[5] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[6] Rajeev Alur, Salvatore La Torre, and P. Madhusudan. Perturbed timed automata. In *International Workshop on Hybrid Systems: Computation and Control*, 2005.

[7] Christopher Amato, Danial S. Bernstein, and Shlomo Zilberstein. Solving POMDPs using quadratically constrained linear programs. In *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2006.

[8] Andrew G. Barto, Steven J. Bradtke, and Satinder P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72:81–138, 1995.

[9] R. E. Bellman. A problem in the sequential design of experiments. *Sankhya*, 16:221–229, 1956.

[10] R. E. Bellman and S. Dreyfus. *Applied Dynamic Programming*. Princeton University Press, New Jersey, 1962.

[11] R. E. Bellman, I. Glicksbert, and O. Gross. On the optimal inventory policy. *Management Sciences*, 2:83–104, 1955.

[12] Michel Berkelaar, Kjell Eikland, Peter Notebaert, et al. LP-solve: open source (mixed-integer) linear programming system. Disponível em `http://groups.yahoo.com/group/lp_solve/` (Maio de 2006).

[13] Daniel S. Bernstein, Eric A. Hansen, and Shlomo Zilberstein. Bounded policy iteration for decentralized POMDPs. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, 2005.

[14] Dimitri Bertsekas. *Dynamic programming and optimal control*. Athena Scientific, 2001.

[15] Blai Bonet and Héctor Geffner. Labeled rtdp: Improving the convergence of real-time dynamic programming. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2003.

[16] Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

[17] Craig Boutilier and David Poole. Computing optimal policies for partially observable decision processes using compact representations. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 1168–1175, 1996.

[18] Anthony Rocco Cassandra. POMDP-solve: polici iteration methods for POMDPs. Disponível em `http://cassandra.org/pomdp/index.shtml` (Maio de 2006).

[19] Anthony Rocco Cassandra. *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, Depertment of Computer Science, Brown University, 1998.

[20] Anthony Rocco Cassandra. A survey of POMDP applications. Presented at the AAAI Fall Symposium, 1998.

[21] Anthony Rocco Cassandra, Michael Littman, and Nevin Zhang. Incremental pruning: a simple, fast, exact method for partially observable Markov decision processes. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-97)*, 1997.

[22] Hsien-Te Cheng. *Algorithms for partially observable Markov decision processes*. PhD thesis, University of British Columbia, Columbia, Canada, 1988.

[23] P. Dayan and G. Hinton. Feudal reinforcement learning. In *Neural Information Processing Systems (NIPS)*, 1993.

[24] T. Dean and S. H. Lin. Decomposition techniques for planning in stochastic domains. Technical report, Dept of Computer Science, Brown University, Providence, Rhode Island, 1995.

[25] Thomas G. Diettrich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research (JAIR)*, 13:227–303, 2000.

[26] James E. Eckles. Optimim maintenance with incomplete information. *Operations Research*, 16, 1968.

[27] ZhengZhu Feng and Eric Hansen. Approximate planning for factored POMDPs. In *Sixth European Conference on Planning*, 2001.

[28] Héctor Geffner and Blai Bonet. Solving large POMDPs by real time dynamic programming. In *Working Notes of the Fall AAAI Symposium on POMDPs*, 1998.

[29] Claudia Goldman and Shlomo Zilberstein. Decentralized control of cooperative systems: Categorization and complexity analysis. *Journal of Artificial Intelligence Research*, 22, 2004.

[30] Carlon Guestrin, Daphne Koller, and Ronald Parr. Solving factored pomdps with linear value functions. In *IJCAI-01 workshop on Palling under Uncertainty and Incomplete Information*, 2001.

[31] Carlos Guestrin, Daphne Koller, Ronald Parr, and Shobha Venkataraman. Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence (JAIR)*, 19:399–468, 2003.

[32] Eric Hansen and Rong Zhou. Synthesis of hierarchical finite-state controllers for POMDPs. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS-03)*, 2003.

[33] Eric A. Hansen. An improved policy iteration algorithm for partially observable mdps. In *Tenth Neural Information Processing Systems Conference (NIPS-97)*, 1997.

[34] Eric A. Hansen. Solving POMDPs by searching in policy space. In *Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*, 1998.

[35] Milos Hauskrecht. Dynamic decision making in stochastic partially observable medical domains: Ischemic heart disease example. In Elpida Keravnou, Catherine Garbay, Robert Baud, and Jeremy Wyatt, editors, *6th Conference on Artificial Intelligence*

in Medicine, volume 1211 of *Lecture Notes in Artificial Intelligence*, pages 296–299. Springer, 1997.

[36] Milos Hauskrecht. *Planning and control in stochastic domains with imperfect information.* PhD thesis, EECS, Massachussets Institute of Technology, 1997.

[37] Milos Hauskrecht. Combining perfectly and partially obervable MDPs. Working paper, 1998.

[38] Milos Hauskrecht. Planning treatment of ischemic heart disease with partially observable markov decision processes. *Artificial Intelligence in Medicine*, 18:221–244, 2000.

[39] Milos Hauskrecht. Value-function approximations for partially observable markov decision processes. *Journal of Artificial Intelligence Research*, 13:33–94, 2000.

[40] John E. Hopcroft, Rajeev Motwani, and Jeffrey Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison Wesley, 2001.

[41] R. Howard. *Dynamic Programming and Markov Processes.* MIT Press, Cambridge, MA, 1960.

[42] Bowen Hui and Craig Boutilier. Who's asking for help? a bayesian approach to intelligent assistance. In *International Conference on Intelligent User Interfaces*, 2006.

[43] Leslie Pack Kaelbling. Hierarchical reinforcement learning. In *ICML-93*, 1993.

[44] Leslie Pack Kaelbling, Michael L. Littman, and Anthony Rocco Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101, 1998.

[45] Long-Ji Lin and Tom M. Mitchell. Memory approaches to reinforcement learning in non-markovian domains. Technical Report CMU-CS-92-138, School of Computer Science, Carnegie-Mellon University, 1992.

[46] John Loch and Satinder Singh. Using eligibility traces to find the best memoryless policy in partially observable Markov decision processes. In *International Conference on Machine Learning*, pages 323–331, 1998.

[47] William S. Lovejoy. Computationally feasible bounds for partially observable markov decison processes. *Operations Research*, 39:192–175, 1991.

[48] S. Mahadevan and N. Khaleeli. Robust mobile robot navigation using partially-observable semi-markov decision processes, 1999.

[49] H. B. McMahan, M. Likhachev, and G. J. Gordon. Bounded real-time dynamic programming: Rtdp with monotone upper bounds and performance guarantees. In *Proceedings of ICML*, 2005.

[50] Nicolas Meuleau, Kee-Eung Kim, Leslie Pack Kaelbling, and Anthony R. Cassandra. Solving POMDPs by searching the space of finite policies. In *Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 417–426, 1999.

[51] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[52] G. E. Monahan. A survey of partially observable Markov decision processes: Theory, models and algorithms. *Management Science*, 28(1):1–16, 1982.

[53] Kevin P. Murphy. A survey of POMDP solution techniques. Technical report, University of California at Berkeley, 2000.

[54] Andrew Y. Ng and Michael Jordan. PEGASUS:A policy search method for large MDPs and POMDPs. In *Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 406–415, 2000.

[55] Christos H. Papadimitriou and J. N. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

[56] Niels Peek. A specialised POMDP form and algorithm for clinical patient management. In A. Abu-Hanna & P. Lucas, editor, *Working Notes of AIMDM'99 Workshop on Prognostic Models in Medicine*, pages 39–43, 1999.

[57] Jerônimo Pellegrini and Jacques Wainer. On the use of POMDPs to model diagnosis and treatment of diseases. In *ENIA'2003*, Campinas, SP, Brazil, 2003.

[58] Jerônimo Pellegrini and Jacques Wainer. Language limited Markov decision processes. *Journal of Artificial Intelligence Research*, Submitted in March 2006.

[59] Leonid Peshkin, Nicolas Meuleau, and Leslie P. Kaelbling. Learning policies with external memory. In *International Conference on Machine Learning*, 1999.

[60] Joelle Pineau. *Tractable Planning Under Uncertainty: Exploiting Structure*. PhD thesis, Robotics Institute, Carnegie-Mellon University, 2004.

[61] Joelle Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

[62] Joelle Pineau and Sebastian Thrun. Hierarchical POMDP decomposition for a conversational robot. In *Workshop on Hierarchy and Memory in Reinforcement Learning (ICML), William College, MA, USA*, 2001.

[63] Joelle Pineau and Sebastian Thrun. An integrated approach to hierarchy and abstraction for POMDPs. Technical Report CMU-RI-TR-02-21, School of Computer Science - Carnegie Mellon University, 2002.

[64] Optimal policies for partially observable Markov systems. J. s. kakalik. Technical Report TR-18, Massachussets Institute of Technology, 1965.

[65] Pascal Poupart. *Exploiting Structure to Efficiently Solve Large Scale Partially Observable Markov Decision Processes*. PhD thesis, University of Toronto, 2005.

[66] Pascal Poupart and Craig Boutilier. Value-directed compression of pomdps. In *Neural Information Processing Systems (NIPS)*, volume 15, 2002.

[67] Pascal Poupart and Craig Boutilier. Bounded finite state controllers. In *Neural Information Processing Systems (NIPS)*, volume 16, 2003.

[68] Pascal Poupart and Craig Boutilier. VDCBPI: an approximate scalable algorithm for large scale pomdps. In *Neural Information Processing Systems (NIPS)*, volume 17, 2004.

[69] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, 1994.

[70] Larry D. Pyeatt and Adele E. Howe. A parallel algorithm for POMDP solution. In *ECP '99: Proceedings of the 5th European Conference on Planning*, pages 73–83, London, UK, 2000. Springer-Verlag.

[71] Nicholas Roy and Geoffrey Gordon. Exponential family pca for belief compression in POMDPs. In *Neural Information Processing Systems (NIPS)*, 2003.

[72] Nicholas Roy, Geoffrey Gordon, and Sebastian Thrun. Finding approximate POMDP solutions through belief compression. *Journal of Artificial Intelligence Research (JAIR)*, 23:1–40, 2005.

[73] Trey Smith. zmdp: approximate value iteration algorithms for MDPs and POMDPs. Disponível em `http://www.cs.cmu.edu/~trey/zmdp/` (Maio de 2006).

[74] Trey Smith and Reid Simmons. Heuristic search value iteration for POMDPs. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI-2004)*, 2004.

[75] Trey Smith and Reid Simmons. Point-based POMDP algorithms: Improved analysis and implementation. In *Proceedings of AAAI*, 2005.

[76] Trey Smith and Reid Simmons. Focused real-time dynamic programming for MDPs: Squeezing more out of a heuristic. In *Proceedings of AAAI*, 2006.

[77] E. Sondik. *The optimal control of partially observable Markov decision processes*. PhD thesis, Stanford University, 1971.

[78] E. J. Sondik and R. D. Smallwood. The optimal control of partially observable Markov processes over the infinite horizon. *Operations Research*, 26(2):1071–1088, 1973.

[79] Matthijs T. J. Spaan and Nikos Vlassis. Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research (JAIR)*, 24:195–220, 2005.

[80] Vladislav B. Tadić Sumeetpal S. Singh and Arnaud Doucet. A policy gradient method for semi-markov decision processes with application to call admission control. Technical Report CUED/F-INFENG/TR-523, Cambridge University, 2005.

[81] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[82] Georgious Theocaurous and Leslie Pack Kaelbling. Approximate planning in POMDPs with macro-actions. In *Neural Information Processing Systems (NIPS)*, 2003.

[83] C. White. Procedures for the solution of a finite-horizon, partially observed, semi-Markov optimization problem. *Operations Research*, 24(2):348–358, 1976.

[84] D. J. White. A survey of applications of markov decision processes. *The Journal of the Operational Research Society*, 44(11):1073–1096, 1993.

[85] David Wingate and Kevin D. Seppi. Prioritization methods for accelerating MDP solvers. *Journal of Machine Learning Research (JMLR)*, 6:851–881, 2005.

[86] Huizhen Janey Yu. *Approximate Solution Methods for Partially Observable Markov and Semi-Markov Decision Processes*. PhD thesis, Massachussets Institute of Technology, 2006.

[87] Nevin Zhang and W. Liu. Planning in stochastic domains: Problem characteristics and approximation. Technical Report HKUST-CS96-31, Hong Kong University of Science and Technology, 1996.

[88] W. Zhang and N. Zhang. Solving informative partially observable markov decision processes. In *Proceedings of the 6th European Conference on Planning (ECP)*, 2001.

[89] Weihong Zhang and Nevin Zhang. Restricte value iteration: theory and algorithms. *Journal of Artificial Intelligence Research*, pages 123–165, 2005.

[90] Rong Zhou and Eric Hansen. An improved grid-based approximation algorithm for POMDPs. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 707–716, 2001.