Processos de Decisão de Markov: um tutorial

Jerônimo Pellegrini [1]
Jacques Wainer [1]

Resumo: Há situações em que decisões devem ser tomadas em seqüência, e o resultado de cada decisão não é claro para o tomador de decisões. Estas situações podem ser formuladas matematicamente como processos de decisão de Markov, e dadas as probabilidades dos valores resultantes das decisões, é possível determinar uma política que maximize o valor esperado da seqüência de decisões. Este tutorial descreve os processos de decisão de Markov (tanto o caso completamente observável como o parcialmente observável) e discute brevemente alguns métodos para a sua solução. Processos semi-Markovianos não são discutidos.

Palavras-chave: processos de decisão de Markov, raciocínio com incerteza, planejamento probabilístico.

Abstract: There are situations where decisions must be made in sequence, and the result of each decision is not clear to the decision maker. These situations can be formulated mathematically as Markov decision processes, and given the probabilities of each value, it is possible to determine a policy that maximizes the expected outcome of a sequence of decisions. This tutorial explains Markov decision processes (both completely observable and partially observable) and briefly discusses some methods for solving them. Semi-Markov processes (where continuous time is modeled) are not discussed.

Keywords: Markov decicion processes, reasoning under uncertainty, probabilistic planning.

1 Introdução

A tomada de decisões em seqüência é uma atividade de grande importância. 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,

[1] Instituto de Computação, Unicamp, Caixa Postal 6176, CEP 13084-971, Campinas – SP, Brazil
{jeronimo,wainer}@ic.unicamp.br

etc.) [1, 2]. Outros exemplos são o sistema de navegação de um robô (que deve decidir para que direção ir, contando apenas com sensores imprecisos) [3]; sistemas de ajuda *online* (que devem tentar ajudar o usuário ao mesmo tempo em que tentam determinar o que realmente o usuário quer) [4]; uma empresa que deve decidir periodicamente a respeito de promoções e mudanças de preço. Até mesmo um diálogo envolve tomada de ações em seqüência e incerteza [5]. 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.

Este tutorial tem como objeto de estudo os processos de decisão de Markov, que são usados para modelar situações onde é necessário executar ações em seqüência 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 otimal* que determine a cada momento que ação tomar para maximizar a recompensa esperada (ou minimizar o custo esperado).

As seções seguintes apresentam primeiro os processos de decisão de Markov completamente observáveis, onde o estado do sistema é conhecido sempre que uma decisão é tomada, mas cada decisão tem resultado incerto. Em seguida, os processos de Markov parcialmente observáveis são apresentados. Nestes, o estado do sistema não é conhecido quando as decisões devem ser tomadas, e a única informação disponível para o tomador de decisões é a seqüência de decisões já tomadas e a seqüência de observações resultantes de cada decisão (uma observação é probabilisticamente relacionada ao estado do sistema e à decisão tomada).

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 [6–8]. 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 [9]. O exemplo a seguir é descrito por Puterman [8]:

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. 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* $s \in S$*.*

Pode-se também 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 $A = \cup_{s \in S} A_s$. Usaremos apenas "A", a fim de simplificar a notação.

Os conjuntos S e A podem ser finitos ou infinitos. Este tutorial tratará apenas do caso em que ambos são finitos.

Definição 2.2 (Horizonte)
O horizonte de um MDP é o número de épocas de decisão *disponíveis para a tomada de decisões. Usaremos* z *para denotar o horizonte de um MDP.*

O horizonte pode ser *finito* (quando há um número fixo de decisões a tomar), infinito (quando a tomada de decisão é feita repetidamente, sem a possibilidade de parada) ou *in-*

definido (semelhante ao horizonte *infinito*, mas com a possibilidade do processo parar se chegar a algum estado que tenha sido marcado como final[2]).

2.1 Política para um MDP

A Figura 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.

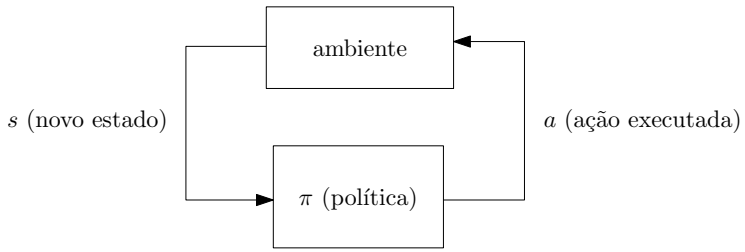


Figura 1. Funcionamento de um sistema modelado como MDP.

A cada época de decisão, o tomador de decisões usa uma *regra de decisão* para escolher a próxima ação. Uma forma simples de regra de decisão é um mapeamento direto de estados em ações (uma regra de decisão pode ser $d : S \mapsto A$, por exemplo). O conjunto de todas as regras de decisão (uma para cada época de decisão) é chamado de política.

Normalmente se quer encontrar uma política que otimize um dado critério de desempenho das decisões. 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).

[2]É possível também definir ações finais.

- *Não-estacionária*, se a ação tomada depende da época de decisão.

Quando uma política for estacionária, $\pi(s)$ pode ser usada como sinônimo de $d_k(s)$. Apesar de ser mais comum o uso de políticas estacionárias para horizonte infinito e não-estacionárias para horizonte finito, tanto políticas estacionárias como não estacionárias podem ser usadas com qualquer horizonte: um sistema com horizonte finito pode ter a mesma política para cada época de decisão; em um sistema com horizonte infinito, pode-se mudar a política periodicamente.

Uma política pode ainda ser:

- *Determinística*, quando cada estado é sempre mapeado em uma única ação;
- *Não-determinística* (*randomizada* ou *estocástica*), quando um estado é mapeado em um conjunto de ações, sendo que cada ação tem uma probabilidade de ser escolhida. Neste caso, cada regra de decisão é uma função $d_k : S \times A \mapsto [0, 1]$.

Finalmente, uma política pode ser classificada como

- *Markoviana* (ou sem memória), quando a escolha da ação depende apenas do estado corrente;
- *Não-Markoviana*, quando a escolha da ação depende de todo o histórico de ações e estados do sistema até o momento. As regras de decisão são definidas então como funções $d_k : \mathcal{H}_k \mapsto A$, onde $\mathcal{H}_k$ é o histórico de ações e estados até a época de decisão k.

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). Pode-se definir diversos critérios de desempenho (ou "de otimalidade") para MDPs, e entre os mais conhecidos podemos citar:

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

Nossa discussão ficará restrita ao critério da esperança da recompensa total descontada. Usando este critério de otimalidade e sendo A finito é possível demonstrar que:

- Para MDPs com horizonte finito onde há limites superior e inferior para as recompensas, existe uma política otimal que é Markoviana e determinística;
- Para MDPs com horizonte infinito, sempre existe uma política otimal que é estacionária e determinística.

Provas destas duas proposições podem ser encontradas no livro de Puterman [8] (páginas 90 e 154), para o caso em que S é finito. Para o caso geral as provas são mais complexas, e podem ser encontradas no livro de Bertsekas e Shreve [10].

A definição de política que usaremos neste tutorial é a que segue.

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.

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

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 $\pi = \{d_0, d_1, \cdots, d_{z-1}\}$,

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

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.

Quando não houver ambiguidade (ou seja, quando houver um único MDP), usaremos Π e $\mathcal{V}$ 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

$$Q^{\pi}(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) V^{\pi}(s'). \tag{1}$$

Para horizonte finito,

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

Dentre todas as possíveis políticas para um MDP, estamos interessados em uma que maximize a recompensa esperada durante a seqüência de decisões.

Definição 2.6 (Política otimal 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 otimal 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 $0 \leq k < z$.

Há uma única função que satisfaz este critério, sendo portanto chamada de *função valor ótima*. O mesmo não é verdade para políticas (é possível que duas políticas tenham a mesma função valor), por isso uma política cuja função valor é ótima é chamada de *otimal*.

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

Se π^* é uma política otimal, 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, entre os quais apenas alguns serão abordados neste tutorial. 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. Dentre os algoritmos discutidos neste tutorial, o de iteração de valores é o único que pode produzir políticas não estacionárias para problemas de horizonte finito.

Definiremos um mapeamento $h : S \times A \times \mathcal{V} \mapsto \mathbb{R}$, 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:

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

Definiremos também 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:

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

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

$$\begin{aligned} V_k(s) &= HV_{k+1}(s) \\ &= \max_{a \in A} h(s,a,V_{k+1}) \\ &= \max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a)V_{k+1}(s') \right]. \end{aligned} \tag{2}$$

A Equação 2 pode ser usada para implementar o algoritmo de *iteração de valores* (Algoritmo 1).

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

```
para cada s ∈ S faça
    V_0(s) ← max_{a∈A} R(s, a);
fim
i ← 1;
enquanto critério de parada não satisfeito faça
    para cada s ∈ S faça
        V_i(s) ← max_{a∈A} [R(s, a) + γ Σ_{s'∈S} T(s'|s, a) V_{i−1}(s')];
    fim
    i ← i + 1;
fim
Devolva V;
```

Algoritmo 1: Iteração de valores para MDPs.

No algoritmo de iteração de valores i 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 função valor ótima. 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 pára 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 seqüência 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 seqüência 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 seqüência N converge para x^*.*

Seja $||\cdot||$ uma norma no espaço $\mathcal{V}$ tal que $||V|| = \max_{s \in S} |V(s)|$. $\mathcal{V}$ é um espaço de Banach. Além disso, o operador H é uma contração em $\mathcal{V}$ (a prova pode ser encontrada no livro de Puterman [8]). Assim, o teorema de Banach garante que H tem um ponto fixo único V^* e que a seqüência $V_k = HV_{k-1}$, 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 conhecido critério de convergência usa o Teorema 2.2, cuja prova é dada também por Puterman [8].

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 $V_k(s)$ nunca difere de $V^*(s)$ por mais de $\frac{\delta}{1-\gamma}$.

Assim, o critério de parada deve ser

$$\forall s \in S, \ \ |V(s) - V'(s)| \leq \frac{\epsilon(1-\gamma)}{2\gamma} \tag{3}$$

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. Este algoritmo é mais eficiente que o de iteração de valores.

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 [8] atribui a idéia de iteração de políticas a Howard [11] e Bellman [12, 13]. O algoritmo de iteração de políticas se baseia no seguintes Teoremas 2.3 e 2.4 de Bellman e Dreyfus [14]:

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 otimal.

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 otimal 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^*, otimal para o MDP dado como entrada.

1 Inicialize π aleatoriamente
2 **repita**
3 $\quad \pi' \leftarrow \pi$;
4 $\quad$ Avalie a política atual resolvendo o sistema linear:
5 $\quad \forall s \in S,\ \ V(s) = R(s, \pi'(s)) + \gamma \sum_{s' \in S} T(s, \pi'(s), s') V(s')$;
6 $\quad$ **para cada** $s \in S$ **faça**
7 $\quad\quad \forall a \in A,\ \ Q^{\pi'}(s, a) \leftarrow R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V(s')$;
8 $\quad$ **fim**
9 $\quad$ **para cada** $s \in S$ **faça**
10 $\quad\quad$ Melhore a política:;
11 $\quad\quad \pi(s) \leftarrow \arg\max_{a \in A} Q^{\pi'}(s, a)$;
12 $\quad$ **fim**
13 **até** *que* $\pi = \pi'$;
14 Devolva π

Algoritmo 2: Iteração de políticas para MDPs.

O Algoritmo 2 mostra uma versão detalhada da iteração de políticas. V^{π} é o valor esperado, usando a política π.

Algoritmo modificado de Iteração de Políticas Puterman [8] descreve uma versão modificada do algoritmo de iteração de políticas que converge com menos iterações do que o Algoritmo 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 nas linhas 4 e 5 do algoritmo 2 é trocada por uma variante de iteração de valores onde a política é fixa (detalhada no Algoritmo 3).

1 **repita**
2 $\quad$ **para cada** $s \in S$ **faça**
3 $\quad\quad V(s) \leftarrow R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V(s')$
4 $\quad$ **fim**
5 **até** *Até que a maior modificação em um valor seja menor que algum* ϵ ;

Algoritmo 3: Avaliação da política no algoritmo modificado de iteração de políticas.

2.2.3 Programação Linear O problema de encontrar a função valor ótima para um MDP de horizonte infinito pode ser formulado como um problema de programação linear. A formulação como programa linear permite adicionar restrições ao modelo. É possível, por exemplo, especificar um valores mínimos e máximos para a recompensa esperada de cada um dos estados. A eficiência deste método depende de como o programa linear é resolvido.

Na formulação de um MDP como programa 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 a função valor 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 1 mostra a função de transição, e a Tabela 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 1. Probabilidades de transição em um MDP.

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

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

	a_0	a_1	a_2
s_0	10	1	30
s_1	50	20	2

Tabela 2. Recompensas em um MDP.

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 função valor 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 sucessivas interações como ambiente [15]. O RTDP executa a melhor que a política corrente puder determinar, atualizando o valor de cada estado visitado usando a equação de otimalidade. O RTDP pode ser usado continuamente por um tomador de decisões, de forma a melhorar a função valor à medida que interage com o ambiente (e este é o motivo do nome "*real time* dynamic programming"), ou pode ser usado com um ambiente simulado. Quando o RTDP é usado com um simulador de ambiente, torna-se funcionalmente semelhante aos outros algoritmos discutidos, que determinam uma política para uso posterior.

O Algoritmo 4 mostra a versão simulada do RTDP. Diversas funções podem ser usadas para implementar a heurística de valor inicial para h [15]. O RTDP é diferente dos outros algoritmos apresentados quanto à política encontrada: um de seus parâmetros de entrada é o estado inicial. Isto permite que o algoritmo encontre uma função valor parcial (definida apenas para os estados alcançáveis a partir do estado inicial, mas ótima para todos estes estados desde que o estado inicial do ambiente seja o mesmo usado na determinação da política).

Uma função valor total pode ser encontrada se o RTDP for executado em várias rodadas, cada uma iniciando 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 seqüência, 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). Quando o fator de desconto é estritamente menod que um e todos os estados são visitados periodicamente na simulação, o RTDP converge para a função valor ótima do MDP. Há outros algoritmos derivados do

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

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

Algoritmo 4: Algoritmo RTDP para MDPs.

RTDP, como o LRTDP [16], o BRTDP [17] e o FRTDP [18].

2.2.5 Outros métodos Uma exposição extensa sobre MDPs é dada por Puterman [8] e por Bertsekas [6], que dão detalhes de outros algoritmos. Hauskrecht [7] também mostra diversos algoritmos para MDPs. Os métodos descritos neste capítulo são os métodos clássicos; há uma grande quantidade de métodos e algoritmos melhores [19–22] para a solução de MDPs que não abordaremos.

3 Processos de Decisão de Markov Parcialmente Observáveis

Um processo de decisão de Markov parcialmente observável (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 possível, 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. Cassandra [23] mostra possibilidades em manutenção de máquinas, navegação de robôs, controle de elevadores, visão computacional, modelagem de comportamento em ecossistemas, aplicações 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 [1, 7]; o trabalho de 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 [24]; e o trabalho de 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 [25].

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 está o tigre);
- 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 sair 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 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.*

Este tutorial trata apenas do caso em que S, A e Ω são finitos.

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.

Como o atual do sistema não é acessível ao tomador de decisões, é possível usar o histórico anterior de ações e observações para escolher a melhor ação.

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 igual a 30 para abrir a porta onde o tigre não está, e uma recompensa negativa (um custo igual a 100) para abrir a porta onde o tigre está;
- Ω: há duas observações, `tigre_esq` e `tigre_dir`;
- O: a ação `ouvir` dá a observação equivalente ao estado atual (ou seja, a observação "correta") com probabilidade 0.9, e a observação oposta com probabilidade 0.1. As ações de abrir portas dão sempre a observação correta.

O problema modelado desta forma pode ser resolvido por algoritmos que determinam uma política otimal 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.

3.1 Dinâmica de sistemas modelados como POMDPs

A dinâmica de um sistema modelado como POMDP é mostrada na Figura 2: 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

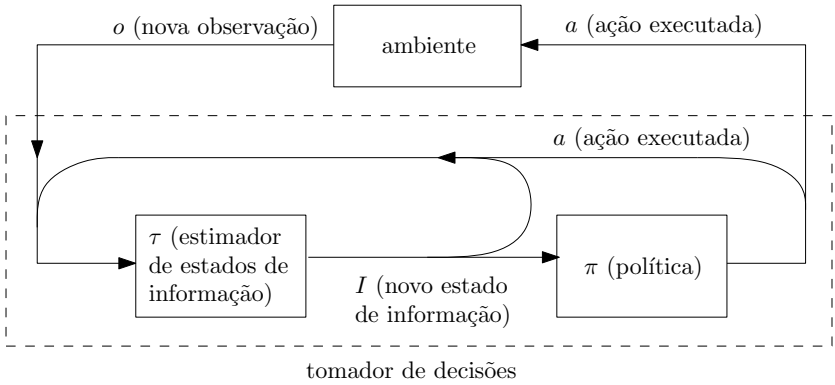


Figura 2. Funcionamento de um sistema modelado como POMDP.

estado de informação. Nesta figura há uma aresta levando o novo estado de informação de τ a τ para enfatizar o fato de que o estado de informação da época de decisão anterior é usado para determinar o da próxima época de decisão.

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. Há uma distribuição inicial de probabilidades sobre os estados, que refletem a *crença* do tomador de decisões a respeito do estado inicial. Ao tomar uma decisão e executar uma ação a, o agente perceberá uma observação o. Sabendo a distribuição de probabilidades anterior, a ação executada e a observação resultante, o agente pode calcular uma nova distribuição de probabilidades sobre os estados (ou seja, um novo "estado de crença"). No problema do tigre o estado de crença inicial é $(0.5, 0.5)$, porque o tomador de decisões nada sabe sobre a localização do tigre. Após executar a ação "ouvir" e receber a observação "tigre à direita" ele poderá melhorar seu estado de crença, que pode mudar para $(0.20, 0.70)$ resultando em um

maior grau de certeza a respeito do que há atrás de cada porta.

No estado de crença inicial, $(0.5, 0.5)$, as recompensas esperadas para as ações são:

- Ouvir: -1;
- Abrir qualquer uma das portas: $(0.5 \times -100 + 0.5 \times 30) = -45$.

Já para o estado de crença $(0.05, 0.95)$, as recompensas são:

- Ouvir: -1;
- Abrir a porta da esquerda: $(0.05 \times -100 + 0.95 \times 30) = 23.5$;
- Abrir a porta da direita: $(0.05 \times 30 - 100 \times 0.95) = -93.5$.

Este exemplo mostra como a escolha da ação depende do estado de crença: quando o tomador de decisões não tem certeza suficiente sobre o estado atual (por exemplo no estado de crença $(0.5, 0.5)$, a melhor decisão (ou seja, a que tem maior recompensa esperada) é "ouvir", porque evita o risco do encontro com o tigre (que aparece no exemplo na forma da recompensa igual a -100) e ajuda a melhorar o estado de crença. Já com o estado de crença $(0.05, 0.95)$ a melhora ação é abrir a porta da esquerda. O tomador de decisões repete a ação "ouvir" até que o estado de crença mude o suficiente para que a recompensa esperada das ações de abrir portas seja maior do que a de ouvir.

POMDPs onde o estado de informação é representado desta forma são muitas vezes chamados de POMDPs de crença (*belief POMDPs*). Em um POMDP com $|S|$ estados, o espaço dos estados de crença é um $(|S| - 1)$-simplex. Denotaremos 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).

3.2 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 [7].

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. O modelo "para a frente" é usado nas discussões teóricas a seguir.

3.3 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 de informação $\tau : \mathcal{B} \times A \times \Omega \mapsto \mathcal{B}$ 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:

$$b_a(o) = \sum_{s \in S} O(o|s,a) b_a(s). \tag{4}$$

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:

$$\rho(b,a) = \sum_{s \in S} b(s) R(s,a). \tag{5}$$

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 [7, 26].

3.4 Política para um POMDP

Da mesma forma que políticas para MDPs, a política para um POMDP pode ou não ser Markoviana, determinística e estacionária.

Uma política para um POMDP de crença pode ser vista como uma política para o MDP sobre estados de informação. A definição 3.3 é de uma política para POMDPs, Markoviana *com relação aos estados de informação*, mas não-Markoviana com relação aos estados do POMDP como originalmente descrito.

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.

É possível definir *políticas reativas* para POMDPs, onde o tomador de decisões consulta apenas a observação mais recente para escolher a próxima ação (sendo então estritamente Markovianas, uma vez que não dependem do histórico de ações e observações passadas). Basta que as regras de decisão sejam definidas como funções $d_k : \Omega \mapsto A$ que determinam uma ação, dada uma observação. As políticas reativas podem ter desempenho pior que as políticas que fazem uso do estado de informação. Singh, Jaakkola e Jordan [27] mostraram que políticas reativas determinísticas podem ser arbitrariamente piores que políticas reativas estocásticas. As políticas estritamente Markovianas não serão discutidas neste tutorial.

Os mesmos critérios de otimalidade usados para MDPs podem ser usados para POMDPs, e a definição de política otimal para POMDP é a mesma que para MDPs, exceto por um fato: a política para POMDPs onde deve mapear estados de informação, e não estados, em ações. Similarmente, funções valor são definidas para POMDPs (mapeando estados de informação em valores).

A função valor ótima para um POMDP de crença é única, existe uma política otimal determinística, e para horizonte infinito existe uma política otimal que é estacionária. Isto decorre trivialmente do POMDP ser um MDP sobre estados de crença.

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 otimal 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

$$Q^\pi(b, a) = \rho(b, a) + \gamma \sum_{o \in \Omega} Pr(o|b, a) V^\pi(\tau(b, a, o)), \tag{6}$$

onde τ é a função de transição para estados de crença (definida na seção 3.3) e $\rho(b, a)$ é a recompensa imediata para uma ação em um estado de crença b, definida na Equação 5 (na página 154).

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 estendido para funções valor, planos condicionais e outras formas de representar o mapeamento de estados de informação em ações.

3.4.1 Representação por 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 (ou seja, mantém-se uma função valor). Cada vetor, quando multiplicado por um estado de crença b, dará a recompensa esperada desde que a ação associada a ele seja tomada e uma política otimal seja seguida até a última época de decisão. Este conjunto de hiperplanos é normalmente denotado por Γ. Neste tutorial usaremos Γ para denotar um vetor de conjuntos, sendo que Γ_i é a política da i-ésima época de decisão. Quando a política for estacionária, todos os Γ_i serão iguais.

A representação por hiperplanos só é possível devido a um fato demonstrado por Sondik [28]: a função valor ótima para um POMDP com 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 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 otimal. Na figura, o vetor α_4 é inútil (porque é dominado pelos outros). A ação representada pelo vetor α_1 é otimal sempre que o estado de crença tiver $p(s_0) \geq 0.7$. A ação do vetor α_2 é otimal quando $0.4 \leq p(s_1) \leq 0.7$, e a do vetor α_3 quando $p(s_0) \leq 0.4$.

Para determinar uma ação otimal dado um estado de crença b, basta escolher o vetor α

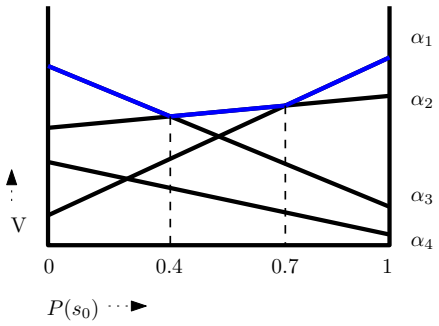


Figura 3. Política para POMDP representada como conjunto de hiperplanos.

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 da forma descrita por Sondik, mas é aproximável por uma política desta forma.

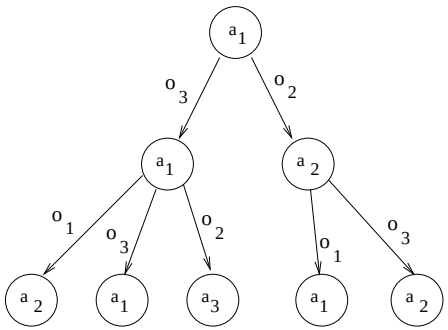
3.4.2 Representação por discretização O simplex do espaço dos estados de crença pode ser discretizado e mapeado em ações [29–31]. Esta técnica é utilizada com algoritmos de aprendizado, e claramente leva a uma solução não-exata. A grade obtida pela discretização pode ter granularidade e regularidade diferentes dependendo do algoritmo. Alguns destes algoritmos são apresentados na subseção 3.5.4.

3.4.3 Representação por planos condicionais Um plano condicional [32] é uma estrutura recursiva que pode ser usada para representar políticas para POMDPs. Um plano condicional é composto de:

- Uma ação a ser executada;
- Uma lista de planos condicionais que podem ser executados em seguida. A escolha do próximo plano depende da observação obtida.

Um plano condicional pode ser representado como uma árvore ou como uma função recursiva (como na Figura 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 4. Política para POMDP representada como árvore (esquerda) e como plano recursivo (direita).

3.4.4 Representação por controladores finitos Uma forma de representar políticas de horizonte infinito é usando controladores finitos [33]. 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 5.

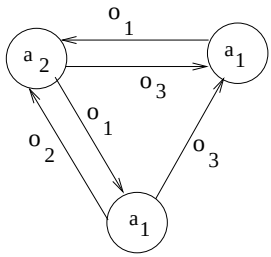


Figura 5. Política para POMDP representada como controlador de estados finitos.

O controlador pode também ser estocástico: para cada estado do controlador, há uma distribuição de probabilidade sobre as ações [34]. 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.

3.4.5 Representação por 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 seqüências em ações [35], usando uma janela deslizante no tempo. Por exemplo, se $k = 3$, apenas as três últimas ações e observações são usadas na política, que consistirá em um mapeamento de todas as possíveis seqüências de três ações e observações em ações. Se o histórico completo é $(a_5, o_1, a_8, o_3, a_8, o_1, a_4, o_2, a_1, o_1)$, então a política usará apenas $(a_8, o_1, a_4, o_2, a_1, o_1)$ para determinar a próxima ação.

A equação de otimalidade, quando escrita em termos de estados de informação, é:

$$V(I_t) = \max_{a \in A} \sum s \in S \rho(s,a) P(s|I_t) + \gamma \sum_{o \in \Omega} P(o|I_t, a) V(\tau(I_t, o, a)),$$

onde I_t é o estado de informação no momento t. Com históricos limitados, I_t é substituído por $\hat{I}_t^k$ (uma lista das últimas k ações e observações):

$$\hat{I}_t^k = (a^{t-k}, o^{t-k+1}, a^{t-k+1}, \cdots, a^{t-1}, o_t).$$

A representação de política por histórico limitado permite que algoritmos mais rápidos sejam usados, mas as políticas encontradas podem ter desempenho arbitrariamente ruim (se ações e observações muito importantes acontecem no início do histórico, elas não estarão mais disponíveis em épocas de decisão mais adiante no tempo).

3.4.6 Representação por redes neurais Redes neurais podem ser treinadas para aproximar a função valor ótima, usando várias arquiteturas diferentes [36, 37]. Algumas das possibilidades são:

- Uma rede tem como entrada as k ações e observações mais recentes, e a saída é uma aproximação da função valor para todos os estados;
- Uma rede tem como entrada a ação e a observação executadas, além de algumas variáveis usadas para representar o estado de informação (um estado de crença, por exemplo). A saída é, como no caso anterior, uma aproximação da função valor para todos os estados.
- Usa-se uma rede neural para cada ação, e cada rede é treinada para aproximar o valor esperado da ação de acordo com um estado de informação. Para escolher uma ação, o estado de informação é usado como entrada uma vez em cada rede, e a ação com melhor valor é escolhida.

Representação	Horizonte	Solução exata
Hiperplanos	F/I	S
Discretização	F/I	N
Planos	F	S
Controladores	I	S
Controladores estocásticos	I	S
Históricos limitados	F/I	N
Redes neurais	I^*	N

Tabela 3. Métodos para a representação de políticas para POMDPs

3.4.7 Comparação entre representações de políticas A tabela 3 resume os métodos usados para representar políticas para POMDPs e suas características. A representação por redes neurais poderia em tese ser adaptada para horizonte finito, embora até onde saibamos isto nunca tenha sido feito. Há outras formas de representar políticas para POMDPs que este tutorial não descreve, inclusive formas mistas (por exemplo, o algoritmo HSVI representa limites inferior e superior da função valor, usando hiperplanos para o limite inferior [38]).

3.5 Algoritmos

Esta seção apresenta algoritmos para determinar políticas otimais para POMDPs. Esta Não é uma lista exaustiva nem completamente detalhada, porque há uma quantidade muito grande de algoritmos que são muito diferentes entre si.

O problema de se encontrar uma função valor ótima para um POMDP é P-ESPAÇO difícil [39] para horizonte finito e indecidível para horizonte infinito (mas decidível e P-ESPAÇO difícil para ϵ-aproximações [7, 40]). A dificuldade em resolver POMDPs está principalmente em dois fatores:

- A *maldição da dimensionalidade*: para um problema com $|S|$ estados, uma política otimal deve ser definida sobre um espaço contínuo de dimensão $|S| - 1$;
- A *maldição do histórico*: a busca por uma política otimal assemelha-se a uma busca em largura por todos os possíveis históricos de ação e observação, simulando o POMDP. O número de seqüências 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 [7, 24, 25, 41]. Todos os algoritmos que abordaremos foram de-

senvolvidos 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 função valor 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 para muitos há garantia de convergência para uma ϵ-aproximação da função valor ótima.

3.5.1 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 uma política otimal:

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

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 5. A função

valor nestes algoritmos é representada por conjuntos de hiperplanos denotados Γ_i, onde i é o número da iteração do algoritmo. Dado um horizonte z, o algoritmo calcula inicialmente a função valor para horizonte 1, em seguida para horizonte 2, até z, ou seja, da última época de decisão para a primeira (por isso dizemos que o conjunto Γ_i representa a época de decisão $z - i$).

A diferença entre os algoritmos de iteração de valores está apenas na forma como calculam Γ_i a partir de Γ_{i-1}, que no Algoritmo 5 consiste na chamada da subrotina `passo_dp` na linha 11.

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 ($r_a(s) = R(s, a)$).

Algoritmo de Sondik O algoritmo mais simples (e menos eficiente) que se conhece é o algoritmo de enumeração de Sondik [42], 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 têm 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 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 4 mostra o tamanho dos conjuntos Γ_i para alguns passos, usando os parâmetros do exemplo mostrado.

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

```
para cada i, 0 ≤ i < z faça
    Γ_i ← ∅ ;
fim
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 5: Iteração de valores para POMDPs com horizonte finito.

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

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

O algoritmo de Sondik é mencionado aqui apenas por sua importância histórica, e por ser a base de outros algoritmos exatos. Na prática, ele é muito ineficiente.

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

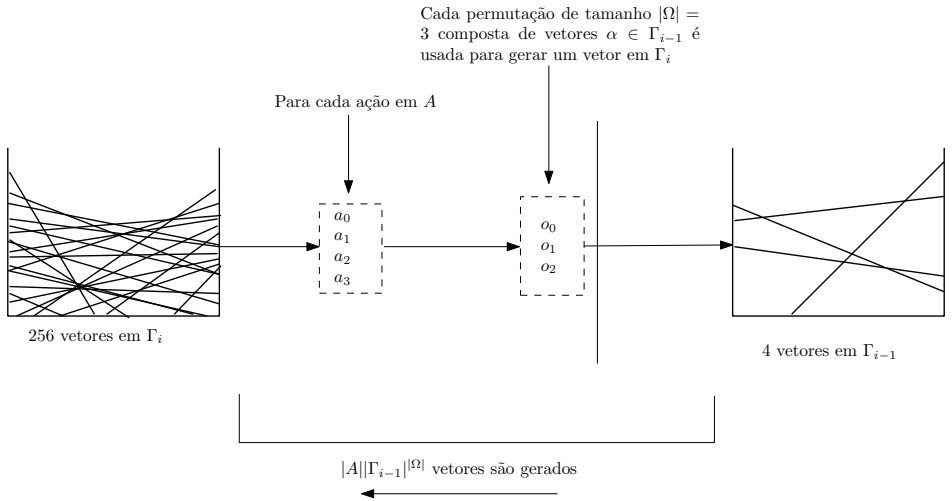


Figura 6. Um passo de programação dinâmica do algoritmo de Sondik.

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 [26].

O algoritmo de Monahan [43] é semelhante ao de Sondik, exceto por fazer um teste de dominância no conjunto de vetores ao final do passo de programação dinâmica, retornando uma representação parcimoniosa da função.

Embora mais eficiente que o algoritmo de Sondik, o algoritmo de Monahan não é muito eficiente.

O algoritmo de poda incremental de Zhang e Liu O teste de dominância é muito caro se realizado para todos os vetores de uma só vez. Zhang e Liu propuseram a poda incremental (*incremental pruning*) dos vetores [44, 45]. O algoritmo faz testes de dominância após calcular vetores para cada ação e conjunto de observações. O Algoritmo 7 mostra a construção da função valor por poda incremental. O símbolo $\oplus$ representa adição de pares ($\Gamma_1 \oplus \Gamma_2 = \{\alpha_1 + \alpha_2 | \alpha_1 \in \Gamma, \alpha_2 \in \Gamma_2\}$).

O algoritmo de poda incremental é uma boa escolha para a obtenção de soluções exatas, embora seja possível usar versões melhoradas, como o a iteração de valores restrita (*restricted value iteration*) [46].

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

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

Algoritmo 6: passo_dp - algoritmo de Sondik.

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 função valor ótima. A função valor ótima para horizonte infinito 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 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 pára quando $e \leq Z\frac{1-\gamma}{2\gamma}$, onde Z é um parâmetro de entrada.

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 7: passo_dp - poda incremental.

3.5.2 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, e são muito mais eficientes do que a iteração de valores. Um dos algoritmos de iteração de políticas para POMDPs é o de Hansen [47, 48], que mantém um hiperplano associado a cada nó de um controlador determinístico. Apesar da representação da política ser por um controlador o algoritmo usa os hiperplanos associados a cada nó para realizar testes de dominância.

Outro algoritmo é o BPI de Poupart [25, 41], que representa a política como controlador estocástico. O BPI determina para cada nó do controlador uma distribuição de probabilidades sobre as ações, sendo que este nó gerará ações de acordo com estas probabilidades. Determina também para cada nó, ação e observação, qual será o próximo nó. A cada passo do BPI, um novo controlador é obtido. A figura 7 mostra uma iteração do BPI. Cada hiperplano é a função valor associada à estratégia estocástica de um nó do controlador. A variável δ mede a distância entre o hiperplano α_n e a função valor que *seria gerada* por um passo de programação dinâmica, de forma que a função valor do controlador se mantém tangente à função valor ótima. A melhoria em cada iteração é conseguida através da resolução de um programa linear. 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

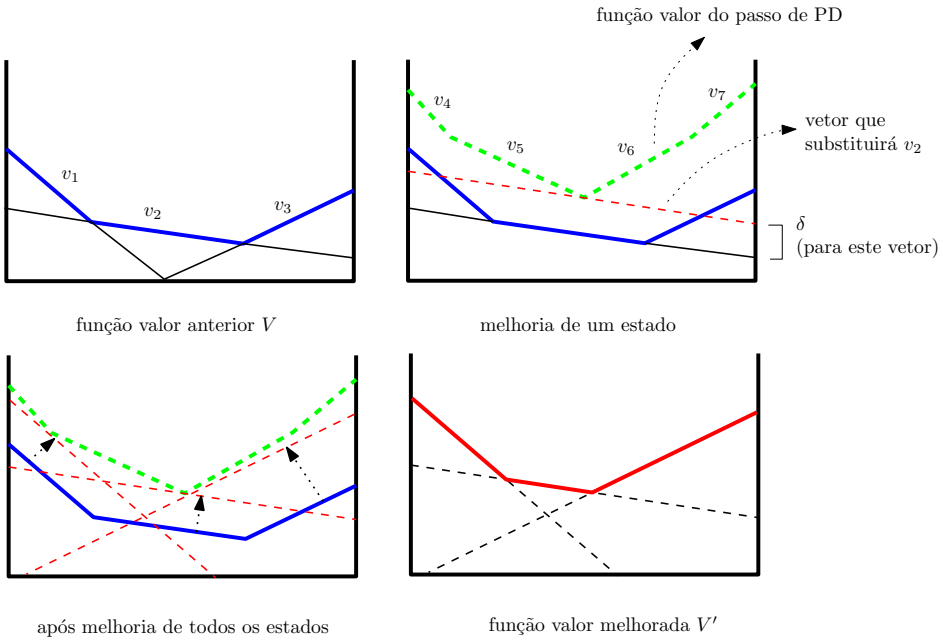


Figura 7. Melhoria da política no algoritmo BPI.

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;

A cada iteração do algoritmo, o seguinte programa linear é resolvido para cada estado k:

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

A primeira restrição determina que os valores para x e y naquele nó devem ser tais que o seu hiperplano seja maximizado, mas sem exceder a função valor ótima que seria gerada se os hiperplanos de todos os nós fossem usados para realizar um passo de programação dinâmica.

Além destes algoritmos, há também métodos de subida pelo gradiente para determinação de um controlador para POMDPs, como o de Aberdeen e Baxter [49, 50], o de Meuleau e outros [51] e o Pegasus de Ng e Jordan [52].

3.5.3 Formulação como programa linear com restrições quadráticas Amato, Bernstein e Zilberstein mostraram que, dado um número fixo de nós, é possível construir um controlador ótimo para um POMDP usando programação linear com resrtições quadráticas [53] (QCLP). Os controladores encontrados usando QCLP foram comparados com os controladores encontrados usando o BPI com o mesmo número de nós, sendo que os controladores obtidos usando QCLP tinham melhor desempenho (recompensa esperada) que aqueles obtidos pelo algoritmo BPI (isto porque não há garantia de que o BPI encontre um controlador ótimo, e o controlador encontrado pelo QCLP é sempre ótimo para aquele número de nós).

A formulação como QCLP é bastante simples: seja z uma variável representando a recompensa esperada total; $y(k,s)$ a recompensa para o estado s do POMDP no nó k do controlador; $x(k',a,k,o)$ a probabilidade de a próxima ação ser a e o próximo nó ser k', dado que o nó atual é k e a última observação foi o. As probabilidades x definem o comportamento do controlador. É possível encontrar valores ótimos para estas probabilidades, bem como para

as recompensas y, resolvendo o seguinte programa linear com restrições quadráticas:

$$\text{maximize } z = b_0(s_0)y(k_0, s_0) + b_0(s_1)y(k_0, s_1) + \cdots + b_0(s_{|S-1|})y(k_0, s_{|S-1|}),$$

sujeito a um conjunto de restrições que representa a equação de otimalidade:

$$\begin{aligned} \forall k, s, \quad y(k,s) \quad &= \quad \sum_{a \in A} \left[\left(\sum_{k'} x(k', a, k, o_0) \right) R(s,a) \right. \\ &+ \quad \left. \gamma \sum_{s'} T(s'|s,a) \sum_{o} O(o|s',a) \sum_{k'} x(k', a, k, o) y(k', s') \right] \end{aligned}$$

e restrições de probabilidade:

$$\forall k, o \quad \sum_{k', a} x(k', a, k, o) = 1$$

$$\forall k, o, a \quad \sum_{k'} x(k', a, k, o) = \sum_{k'} x(k', a, k, o_0)$$

$$\forall k', a, k, o \quad x(k', a, k, o) \geq 0.$$

Os inventores do QCLP ainda não puderam comparar o desempenho do algoritmo QCLP com o de outros algoritmos.

Apesar de encontrar um controlador ótimo para um dado número de nós, o QCLP tem dois problemas: um deles é que depende de bons resolvedores de programas não-lineares (de forma semelhante à formulação de MDPs como programas lineares); outro problema é que é necessário escolher, a priori, o número de nós do controlador. Uma seqüência de más escolhas pode levar à construção de vários controladores até que o controlador ideal seja obtido.

3.5.4 Discretização do espaço de crença Há uma classe de heurísticas para resolver POMDPs que se baseiam 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. Os primeiros métodos baseados em grades de pontos de crença predatam os métodos exatos [54, 55]. De acordo com Cassandra, todos apresentavam desempenho muito ruim para problemas médios [26]. Há hoje algoritmos melhores nesta classe [31, 56], e são particularmente interessantes os métodos que aumentam a grade à medida que passos de programação dinâmica são executados, como no PBVI de Pineau [24, 57] e o HSVI de Smith e Simmons [3, 38]. 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 [58].

As próximas subseções apresentam um método simples de discretização e o RTDP-Bel, que simula a execução do POMDP enquanto procura pela função valor ótima.

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 [7] 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; uma delas é começar com a crença uniforme ($b(s) = \frac{1}{|S|}$), e depois aumentar o espaço de crença calculando os possíveis estados resultantes usando o estimador de estados (até um certo limite). O Algoritmo 8 mostra a primeira parte deste método. Se o estado de crença inicial for fornecido com o problema, é melhor que ele seja usado ao invés da crença uniforme.

Entrada: Um conjunto S de estados.
Saída: Um conjunto l de pontos de crença.

```
1 l ← ∅;
2 para cada s ∈ S faça
3 |   b(s) ← 1/|S|;
4 fim
5 l ← l ∪ {b};
6 Devolva l
```

Algoritmo 8: Inicialização uniforme do espaço de crença em uma grade.

Outra maneira de inicializar a grade é incluir todos os cantos do simplex de crença ($b(s) = 1, \ \forall s \in S$).

Estes métodos são seguidos de uma busca em largura nos possíveis estados de crença seguintes, usando o estimador de estados e simulando cada par de ação e observação, como mostra o Algoritmo 9.

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**
  $b \leftarrow B$.proximo;
  para cada $a \in A, o \in \Omega$ **faça**
    $b' = \tau(b, a, o)$;
    B.enfilere(b');
  fim
fim
Devolva B

Algoritmo 9: Simulação de estados de crença para inicialização da grade.

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

O valor de uma ação em um estado de crença é calculado usando as Equações 5 e 6.

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 [59].

O Algoritmo 10 executa uma rodada do RTDP-Bel, dado um estado de crença inicial (A definição de $b_a(o)$ está na Equação 4). A escolha do conjunto de estados iniciais (que se resume na discretização do espaço de estados) já foi discutida na subseção 3.4.2.

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

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

Algoritmo 10: Algoritmo RTDP para POMDPs.

Hauskrecht discute diversas outras heurísticas que podem ser usadas como limite inferior para funções valor de POMDPs [7].

3.5.5 Outros algoritmos A tabela 5 compara alguns dos algoritmos para a solução de POMDPs. Há uma grande quantidade de algoritmos não cobertos neste tutorial. Uma visão mais ampla dos métodos para solução de POMDPs pode ser obtida nos trabalhos de Hauskrecht [7], Murphy [60], Cassandra [26] e Smith [61]. Há também outros trabalhos, como algoritmos paralelos [62]; outros algoritmos [16, 18, 52, 63] e métodos de decomposição de POMDPs [64].

4 Mais sobre MDPs e POMDPs

Os livros de Puterman [8] e Bertsekas [6] tratam formalmente de MDPs, de forma aprofundada, além de tratarem de processos de decisão semi-Markovianos. No entanto, algoritmos mais novos e eficientes para a solução de MDPs encontram-se em outras fontes [19, 22,

	Horizonte	Solução exata
Iteração de valores	F/I	S (para horizonte finito)
Iteração de políticas	I	N
QCLP	I	S (para número fixo de nós)
Discretização	F/I	N
RTDP	I	N

Tabela 5. Métodos para solução de POMDPs.

64].

Para um tratamento mais extenso sobre POMDPs, é interessante consultar os trabalhos de Cassandra [26] e Hasukrecht [7]. Técnicas eficientes para representação e solução de POMDPs são dadas por Poupart [25], Pineau [57, 64] e Smith [3, 18].

Os MDPs e POMDPs permitem modelar problemas onde o momento de cada decisão não é importante. Já os processos semi-Markovianos incluem a noção de tempo contínuo, e permitem modelar problemas onde o momento em que uma época de decisão ocorre é importante (permitindo assim modelar durações para as épocas de decisão). Tanto Puterman [8] como Bertsekas [6] discutem SMDPs. Já os POSMDPs são pouco estudados, sendo interessantes os trabalhos de Mahadevan [65] e Yu [66]. Há também um trabalho de White [67] que aproxima POSMDPs usando tempo discreto, e o de Hansen e Zhou [33], onde é discutida a aproximação de POSMDPs através da política do POMDP subjacente ao POSMDP.

Há também variantes de MDPs e POMDPs, como os MDPs descentralizados [68–70], limitados por linguagem [71], MDPs com parâmetros imprecisos [72–74] e MDPs contínuos [75]. Littman e Szepesvári [76, 77] descrevem também uma formulação generalizada para processos de decisão de Markov, jogos de Markov e diversas variantes de MDPs.

Referências

1 HAUSKRECHT, M. Dynamic decision making in stochastic partially observable medical domains: Ischemic heart disease example. In: KERAVNOU, E. et al. (Ed.). *6th Conference on Artificial Intelligence in Medicine*. [S.l.]: Springer, 1997. (Lecture Notes in Artificial Intelligence, v. 1211), p. 296–299.

2 PELLEGRINI, J.; WAINER, J. On the use of POMDPs to model diagnosis and treatment of diseases. In: *Encontro Nacional de Inteligência Artificial (ENIA)*. Campinas, SP, Brazil: Sociedade Brasileira de Computação, 2003.

3 SMITH, T.; SIMMONS, R. Point-based POMDP algorithms: Improved analysis and implementation. In: VELOSO, M. M.; KAMBHAMPATI, S. (Ed.). *National Conference on Artificial Intelligence (AAAI)*. [S.l.]: AAAI Press, 2005. ISBN 1-57735-236-X.

4 HUI, B.; BOUTILIER, C. Who's asking for help? a Bayesian approach to intelligent assistance. In: PARIS, C.; SIDNER, C. L. (Ed.). *International Conference on Intelligent User Interfaces (IUI)*. Sydney, Australia: ACM, 2006. ISBN 1-59593-287-9.

5 PINEAU, J.; THRUN, S. Hierarchical POMDP decomposition for a conversational robot. In: *Workshop on Hierarchy and Memory in Reinforcement Learning*. William College, MA, USA: [s.n.], 2001.

6 BERTSEKAS, D. *Dynamic programming and optimal control*. Belmont, MA: Athena Scientific, 2001. ISBN 1-886259-08-6.

7 HAUSKRECHT, M. *Planning and control in stochastic domains with imperfect information*. Tese (Doutorado) — EECS, Massachussets Institute of Technology, 1997.

8 PUTERMAN, M. L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York, NY: Wiley-Interscience, 1994. ISBN 0471619779.

9 WHITE, D. J. A survey of applications of Markov decision processes. *The Journal of the Operational Research Society*, v. 44, n. 11, p. 1073–1096, 1993.

10 BERTSEKAS, D.; SHREVE, S. E. *Stochastic Optimal Control: the discrete time case*. Belmont, MA: Athena Scientific, 1996. ISBN 1-886529-03-5.

11 HOWARD, R. *Dynamic Programming and Markov Processes*. Cambridge, MA: MIT Press, 1960.

12 BELLMAN, R. E. A problem in the sequential design of experiments. *Sankhya*, v. 16, p. 221–229, 1956.

13 BELLMAN, R. E.; GLICKSBERT, I.; GROSS, O. On the optimal inventory policy. *Management Science*, v. 2, n. 1, p. 83–104, 1955.

14 BELLMAN, R. E.; DREYFUS, S. *Applied Dynamic Programming*. New Jersey: Princeton University Press, 1962.

15 BARTO, A. G.; BRADTKE, S. J.; SINGH, S. P. Learning to act using real-time dynamic programming. *Artificial Intelligence*, v. 72, p. 81–138, 1995.

16 BONET, B.; GEFFNER, H. Labeled RTDP: Improving the convergence of real-time dynamic programming. In: GIUNCHIGLIA, E.; MUSCETTOLA, N.; NAU, D. (Ed.). *International Conference on Automated Planning and Scheduling (ICAPS)*. Trento, Italy: AAAI Press, 2003.

17 MCMAHAN, H. B.; LIKHACHEV, M.; GORDON, G. J. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In: RAEDT, L. D.; WROBEL, S. (Ed.). *International Conference on Machine Learning (ICML)*. Bonn, Germany: ACM, 2005. ISBN 1-59593-180-5.

18 SMITH, T.; SIMMONS, R. Focused real-time dynamic programming for MDPs: Squeezing more out of a heuristic. In: *National Conference on Artificial Intelligence (AAAI)*. [S.l.]: AAAI Press, 2006.

19 GUESTRIN, C. et al. Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research (JAIR)*, v. 19, p. 399–468, 2003.

20 MITCHELL, T. M. *Machine Learning*. [S.l.]: McGraw-Hill, 1997. ISBN 0-07-042807-7.

21 SUTTON, R. S.; BARTO, A. G. *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998. ISBN 0-262-19398-1.

22 WINGATE, D.; SEPPI, K. D. Prioritization methods for accelerating MDP solvers. *Journal of Machine Learning Research (JMLR)*, v. 6, p. 851–881, 2005.

23 CASSANDRA, A. R. *A survey of POMDP applications*. 1998. Presented at the AAAI Fall Symposium.

24 PINEAU, J. *Tractable Planning Under Uncertainty: Exploiting Structure*. Tese (Doutorado) — Robotics Institute, Carnegie-Mellon University, 2004.

25 POUPART, P. *Exploiting Structure to Efficiently Solve Large Scale Partially Observable Markov Decision Processes*. Tese (Doutorado) — University of Toronto, 2005.

26 CASSANDRA, A. R. *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes*. Tese (Doutorado) — Depertment of Computer Science, Brown University, 1998.

27 SINGH, S.; JAAKKOLA, T.; JORDAN, M. I. Learning without state-estimation in partially observable markovian decision processes. In: *International Conference on Machine Leadning (ICML)*. New Brunswick, NJ, USA: Morgan Kaufmann, 1994. p. 284–292. ISBN 1-55860-335-2.

28 SONDIK, E. J.; SMALLWOOD, R. D. The optimal control of partially observable Markov processes over the infinite horizon. *Operations Research*, v. 26, n. 2, p. 1071–1088, 1973.

29 LOVEJOY, W. S. Computationally feasible bounds for partially observable Markov decison processes. *Operations Research*, v. 39, p. 192–175, 1991.

30 ROY, N.; GORDON, G.; THRUN, S. Finding approximate POMDP solutions through belief compression. *Journal of Artificial Intelligence Research (JAIR)*, v. 23, p. 1–40, 2005.

31 ZHOU, R.; HANSEN, E. An improved grid-based approximation algorithm for POMDPs. In: NEBEL, B. (Ed.). *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. Seattle, Washington, USA: Morgan Kaufmann, 2001. p. 707–716.

32 KAELBLING, L. P.; LITTMAN, M. L.; CASSANDRA, A. R. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, v. 101, 1998.

33 HANSEN, E.; ZHOU, R. Synthesis of hierarchical finite-state controllers for POMDPs. In: GIUNCHIGLIA, E. et al. (Ed.). *International Conference on Automated Planning and Scheduling (ICAPS)*. Trento, Italy: AAAI Press, 2003.

34 POUPART, P.; BOUTILIER, C. Bounded finite state controllers. In: THRUN, S.; SAUL, L. K.; SCHÖLKOPF, B. (Ed.). *Conference on Neural Information Processing Systems (NIPS)*. [S.l.]: MIT Press, 2003. v. 16. ISBN 0-262-20152-6.

35 WHITE, C. C.; SCHERER, W. T. Finite memory suboptimal design for partially observed Markov decision processes. *Operations Research*, v. 42, n. 3, p. 439–455, 1994.

36 LIN, L.-J.; MITCHELL, T. M. *Memory Approaches to Reinforcement Learning in Non-Markovian Domains*. Relatório técnico CMU-CS-92-138, School of Computer Science, Carnegie-Mellon University, 1992.

37 BERTSEKAS, D.; TSITSIKLIS, J. *Neuro-Dynamic Programming*. Belmont, MA: Athena Scientific, 1996. ISBN 1-886529-10-8.

38 SMITH, T.; SIMMONS, R. Heuristic search value iteration for POMDPs. In: CHICKERING, M.; HALPERN, J. (Ed.). *20^{th} Conference on Uncertainty in Artificial Intelligence (UAI)*. Banff, Canada: AUAI Press, 2004.

39 PAPADIMITRIOU, C. H.; TSITSIKLIS, J. N. The complexity of Markov decision processes. *Mathematics of Operations Research*, v. 12, n. 3, p. 441–450, 1987.

40 ABERDEEN, D. *A (Revised) Survey of Approximate Methods for Solving Partially Observable Markov Decision Processes*. Relatório técnico, National ICT Australia, Canberra, Australia, 2003.

41 POUPART, P.; BOUTILIER, C. VDCBPI: an approximate scalable algorithm for large scale POMDPs. In: *Conference on Neural Information Processing Systems (NIPS)*. [S.l.: s.n.], 2004. v. 17.

42 SONDIK, E. *The optimal control of partially observable Markov decision processes*. Tese (Doutorado) — Stanford University, 1971.

43 MONAHAN, G. E. A survey of partially observable Markov decision processes: Theory, models and algorithms. *Management Science*, v. 28, n. 1, p. 1–16, 1982.

44 CASSANDRA, A. R.; LITTMAN, M.; ZHANG, N. Incremental pruning: a simple, fast, exact method for partially observable Markov decision processes. In: GEIGER, P. P. S. D. (Ed.). *13^{th} Annual Conference on Uncertainty in Artificial Intelligence (UAI)*. Providence, RI: Morgan Kauffman, 1997.

45 ZHANG, N.; LIU, W. *Planning in stochastic domains: Problem characteristics and approximation*. Relatório técnico HKUST-CS96-31, Hong Kong University of Science and Technology, 1996.

46 ZHANG, W.; ZHANG, N. Restricted value iteration: theory and algorithms. *Journal of Artificial Intelligence Research (JAIR)*, p. 123–165, 2005.

47 HANSEN, E. A. An improved policy iteration algorithm for partially observable MDPs. In: JORDAN, M. I.; KEARNS, M. J.; SOLLA, S. A. (Ed.). *Conference on Neural Information Processing Systems (NIPS)*. [S.l.]: The MIT Press, 1997. ISBN 0-262-10076-2.

48 HANSEN, E. A. Solving POMDPs by searching in policy space. In: COOPER, S. M. G. (Ed.). *14^{th} Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI)*. Madison, WI: Morgan Kauffman, 1998.

49 ABERDEEN, D. *Policy-Gradient Algorithms for Partially Observable Markov Decision Processes*. Tese (Doutorado) — Australian National University, 2003.

50 ABERDEEN, D.; BAXTER, J. Scaling internal-state policy-gradient methods for POMDPs. In: SAMMUT, C.; HOFFMANN, A. G. (Ed.). *International Conference on Machine Learning (ICML)*. Sydney, NSW, Australia: Morgan Kaufmann, 2002. ISBN 1-55860-873-7.

51 MEULEAU, N. et al. Solving POMDPs by searching the space of finite policies. In: LASKEY, K.; PRADE, H. (Ed.). *15^{th} Conference on Uncertainty in Artificial Intelligence (UAI)*. Stockholm, Sweden: Morgan Kaufmann, 1999. p. 417–426.

52 NG, A. Y.; JORDAN, M. PEGASUS:A policy search method for large MDPs and POMDPs. In: BOUTILIER, C.; GOLDSZMIDT, M. (Ed.). *16^{th} Conference on Uncertainty in Artificial Intelligence (UAI)*. Stanford, CA: Morgan Kaufmann, 2000. p. 406–415.

53 AMATO, C.; BERNSTEIN, D. S.; ZILBERSTEIN, S. Solving POMDPs using quadratically constrained linear programs. In: VELOSO, M. M. (Ed.). *International Joint Conference on Artificial Intelligence (IJCAI)*. Hyderabad, India: [s.n.], 2007. p. 2418–2424.

54 ECKLES, J. E. Optimim maintenance with incomplete information. *Operations Research*, v. 16, n. 5, p. 1058–1067, 1968.

55 KAKALIK, J. S. *Optimal policies for partially observable Markov systems*. Relatório técnico TR-18, Massachussets Institute of Technology, 1965.

56 THEOCAUROUS, G.; KAELBLING, L. P. Approximate planning in POMDPs with macro-actions. In: THRUN, S.; SAUL, L. K.; SCHÖLKOPF, B. (Ed.). *Conference on Neural Information Processing Systems (NIPS)*. [S.l.]: MIT Press, 2003. ISBN 0-262-20152-6.

57 PINEAU, J.; GORDON, G.; THRUN, S. Point-based value iteration: An anytime algorithm for POMDPs. In: GOTTLOB, G.; WALSH, T. (Ed.). *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. Acapulco, Mexico: Morgan Kaufmann, 2003.

58 SPAAN, M. T. J.; VLASSIS, N. Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research (JAIR)*, v. 24, p. 195–220, 2005.

59 GEFFNER, H.; BONET, B. Solving large POMDPs by real time dynamic programming. In: *Working Notes of the Fall AAAI Symposium on POMDPs*. [S.l.: s.n.], 1998.

60 MURPHY, K. P. *A Survey of POMDP Solution Techniques*. Relatório técnico, University of California at Berkeley, 2000.

61 SMITH, T. *Probabilistic Planning for Robotic Exploration*. Tese (Doutorado) — The Robotics Institute, Carnegie Mellon University, 2007.

62 PYEATT, L. D.; HOWE, A. E. A parallel algorithm for POMDP solution. In: BIUNDO, S.; FOX, M. (Ed.). *5^{th} European Conference on Planning (ECP)*. London, UK: Springer-Verlag, 2000. p. 73–83. ISBN 3-540-67866-2.

63 FENG, Z.; ZILBERSTEIN, S. Region-based incremental pruning for POMDPs. In: CHICKERING, D. M.; HALPERN, J. Y. (Ed.). *20^{th} Conference on Uncertainty in Artificial Intelligence (UAI)*. Banff, Canada: AUAI Press, 2004.

64 PINEAU, J.; THRUN, S. *An integrated approach to hierarchy and abstraction for POMDPs*. 2002. Relatório técnico CMU-RI-TR-02-21, School of Computer Science - Carnegie Mellon University, 2002.

65 MAHADEVAN, S. Partially observable semi-Markov decision processes: Theory and applications in engineering and cognitive science. In: *American Association for Artificial Intelligence Fall Symposium*. [S.l.: s.n.], 1998.

66 YU, H. J. *Approximate Solution Methods for Partially Observable Markov and Semi-Markov Decision Processes*. Tese (Doutorado) — Massachussets Institute of Technology, 2006.

67 WHITE, C. C. Procedures for the solution of a finite-horizon, partially observed, semi-Markov optimization problem. *Operations Research*, v. 24, n. 2, p. 348–358, 1976.

68 GOLDMAN, C.; ZILBERSTEIN, S. Decentralized control of cooperative systems: Categorization and complexity analysis. *Journal of Artificial Intelligence Research (JAIR)*, v. 22, 2004.

69 HANSEN, E. A.; BERNSTEIN, D. S.; ZILBERSTEIN, S. Dynamic programming for partially observable stochastic games. In: MCGUINNESS, D. L.; FERGUSON, G. (Ed.). *National Conference on Artificial Intelligence (AAAI)*. [S.l.]: AAAI Press, 2004. p. 709–715. ISBN 0-262-51183-5.

70 NAIR, R. et al. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In: GOTTLOB, G.; WALSH, T. (Ed.). *Proceedings of the International Joint conference on Artificial Intelligence (IJCAI)*. Acapulco, Mexico: Morgan Kaufmann, 2003.

71 PELLEGRINI, J. *Language Limited Markov Devision Processes*. Tese (Doutorado) — Instituto de Computação – Unicamp, Brazil, 2006.

72 TREVIZAN, F.; BARROS, L. N. de; COZMAN, F. G. Unifying nondeterministic and probabilistic planning through imprecise Markov decision processes. In: SICHMAN, J. S.; COELHO, H.; REZENDE, S. O. (Ed.). *Proceedings of the SBIA/IBERAMIA Joint Conference*. Ribeirão Preto, Brazil: Springer, 2006. ISBN 3-540-45462-4.

73 WHITE, C. C. Markov decision processes with imprecise transition probabilities. *Operations Research*, v. 42, n. 4, p. 739–749, 1994.

74 ITOH, H.; NAKAMURA, K. Partially observable Markov decision processes with imprecise parameters. *Artificial Intelligence*, v. 171, p. 453–490, 2007.

75 PORTA, J. M. et al. Point-based value iteration for continuous POMDPs. *Journal of Machine Learning Research*, v. 7, p. 23292367, 2006.

76 SZEPESVÁRI, C.; LITTMAN, M. L. *Generalized Markov Decision Processes: Dynamic-programming and Reinforcement-learning Algorithms*. Relatório técnico CS-96-11, Department of Computer Science, Brown University, 1996.

77 LITTMAN, M. L.; SZEPESVÁRI, C. A generalized reinforcement-learning model: Convergence and applications. In: SAITTA, L. (Ed.). *International Conference on Machine Learning (ICML)*. Bari, Italy: Morgan Kaufmann, 1996.