SOLUÇÕES APROXIMADAS PARA PROBLEMAS DE
TOMADA DE DECISÃO SEQÜENCIAL

André da Motta Salles Barreto

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA CIVIL.

Aprovada por:

Prof. Nelson Francisco Favilla Ebecken, D. Sc.

Prof. Helio José Corrêa Barbosa, D. Sc.

Prof. Carlos Henrique Costa Ribeiro, PhD.

Prof[a]. Karla Tereza Figueiredo Leite, D. Sc.

Prof[a]. Beatriz de Souza Leite Pires de Lima, D. Sc.

RIO DE JANEIRO, RJ - BRASIL

MAIO DE 2008

BARRETO, ANDRÉ DA MOTTA SALLES

Soluções Aproximadas para Problemas de Tomada de Decisão Seqüencial [Rio de Janeiro] 2008

XXIII, 240 p. 29,7 cm (COPPE/UFRJ, D.Sc., Engenharia Civil, 2008)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Problemas de Tomada de Decisão Seqüencial
2. Processos de Decisão de Markov
3. Programação Dinâmica
4. Aprendizagem por Reforço

I. COPPE/UFRJ II. Título (série)

Para o meu pai, Jubel, minha mãe, Helena, e
meus irmãos: Tati, Mamu e Bruninho.

Agradecimentos

Agradeço à CAPES, que além de me conceder uma bolsa de doutorado me financiou por um período de um ano nos Estados Unidos. Durante esse período no exterior trabalhei sob a supervisão do professor Whitley, cujo apoio extrapolou em muito as suas obrigações para comigo. Agradeço também ao professor Anderson e ao Nate, meu amigo, que me mostraram o "caminho das pedras" na área de aprendizagem por reforço. O Nate também me mostrou o caminho para alguns *pubs* bem legais. Outras pessoas imprescindíveis durante esse período nos EUA foram os meus amigos Kadu e Marcinho, responsáveis por momentos hilários inesquecíveis.

De volta ao Brasil, agradeço ao Nelson, meu orientador, pela confiança depositada em mim. Com o seu bordão "vamos que vamos," ele vai resolvendo um-a-um os problemas que surgem, deixando o tempo livre para o que realmente interessa, que é a pesquisa propriamente dita. Agradeço também ao meu amigo Douglas, pelas ajudas técnicas e pelas discussões científicas. Não posso deixar de agradecer aos meus amigos históricos, a famosa "galera," pelos momentos de descontração que ajudaram tanto a espairecer.

Aos meus irmãos e cunhados, agradeço pela companhia e pelos momentos de humor no dia-a-dia. Aos meus sobrinhos, pela maneira doce de atrapalhar. Às tias, pelas visitas divertidas à granja. O próximo agradecimento, aos meus pais, é o mais difícil, porque não importa o que eu diga fica sempre a sensação de que poderia ter dito mais. Portanto, me limito a citar aqui as suas contribuições mais diretas para o desenvolvimento deste trabalho. O resto fica para os cartões de Natal. Ao meu pai, agradeço pelas discussões filosóficas e por ter me dado abrigo em um momento tão fundamental. Foi no silêncio do seu apartamento que a parte teórica desta tese foi desenvolvida. À minha mãe, pelo suporte afetivo, pela paciência com as alterações de humor e pela revisão cuidadosa do texto. Ah! Ótima a idéia do caderninho sem pauta, onde rabisquei as primeiras idéias.

Dois agradecimentos especiais. O primeiro deles é ao Helinho, o grande responsável pela minha carreira acadêmica até aqui. Foi em uma de suas palestras sobre os algoritmos genéticos, em meados de 1998, que eu decidi: "eu quero fazer o que esse cara faz!" Desde então o Helinho vem servindo como referência e inspiração. Finalmente, agradeço à Carol, com certeza a pessoa mais presente na minha vida durante esses anos de doutorado. É impossível listar todas as suas contribuições para esta tese. Obrigado pela companhia, pelo interesse no meu trabalho, pelo apoio em todos os níveis e pela paciência para esperar os "banhos para pensar."

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

SOLUÇÕES APROXIMADAS PARA PROBLEMAS DE TOMADA DE DECISÃO SEQÜENCIAL

André da Motta Salles Barreto
Maio/2008

Orientadores: Nelson Francisco Favilla Ebecken
Helio José Corrêa Barbosa

Programa: Engenharia Civil

Os problemas de tomada de decisão seqüencial envolvem uma série de escolhas sucessivas cujos efeitos podem se estender indefinidamente pelo futuro. Trata-se de um paradigma genérico que engloba desde tarefas simples do dia-a-dia até desafios enfrentados pela indústria. Uma maneira de solucionar esse tipo de problema é modelá-lo como um processo de decisão de Markov (MDP). Uma vez que um modelo formal do problema esteja disponível, pode-se recorrer à programação dinâmica ou à aprendizagem por reforço para determinar uma política de decisão ótima. No entanto, essas abordagens sofrem de uma séria questão de escalabilidade: problemas de tomada de decisão com um número razoavelmente grande de estados podem inviabilizá-las na prática, devido ao seu alto custo computacional. Uma forma de contornar essa questão é criar um modelo compacto do MDP. A abordagem apresentada neste trabalho, chamada fatoração estocástica, é uma proposta nesse sentido. A fatoração estocástica é a formalização de uma idéia bastante intuitiva: pode-se reduzir consideravelmente a dimensão de um MDP simplesmente redirecionando as suas transições para "estados arquetípicos" que representem bem a sua dinâmica. Resolvendo o problema no modelo reduzido, é possível encontrar uma política de decisão em uma pequena fração do tempo que levaria a solução do MDP original. O desempenho das políticas retornadas depende unicamente da qualidade da fatoração: em particular, uma fatoração estocástica exata leva garantidamente a uma das soluções ótimas do problema. Para demonstrar a efetividade desta abordagem na prática, os algoritmos derivados da fatoração estocástica são comparados com outras técnicas de programação dinâmica e aprendizagem por reforço em problemas de controle simples.

Abstract of the Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

APPROXIMATE SOLUTIONS FOR SEQUENTIAL DECISION-MAKING PROBLEMS

André da Motta Salles Barreto
May/2008

Advisors: Nelson Francisco Favilla Ebecken
Helio José Corrêa Barbosa

Department: Civil Engineering

In a sequential decision-making problem the consequences of a decision may last for an arbitrarily long time. This framework is generic enough to encompass tasks ranging from simple every-day-life decisions to complex challenges faced in industrial settings. One way to solve this kind of problem is to use a Markov decision process (MDP) as a model of the task. Once a formal description of the problem is available, dynamic programming and reinforcement-learning techniques may be used to find an optimal decision policy. However, these techniques do not scale well to large problems, since their computational cost grows fast with the number of possible states in the task. This work presents an approach to create a compact model of an MDP. The stochastic factorization developed here formalizes a simple idea: the dimension of a Markov decision process can be significantly reduced by simply redirecting its transitions to a small set of "archetypical states" which represent its dynamics well. Using this compact model, it is possible to find a decision policy within a small fraction of the time that would be required to solve the problem with the original MDP. The performance of the resulting policies depends only on the quality of the stochastic factorization: in particular, an exact factorization leads to an optimal solution with probability one. In order to demonstrate the effectiveness of the stochastic factorization in practice, the performance of the proposed algorithms is compared to that of other dynamic programming and reinforcement-learning techniques in simple benchmark tasks.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Símbolos

Palavras-chave

1. Problemas de Tomada de Decisão Seqüencial
2. Processos de Decisão de Markov
3. Programação Dinâmica
4. Aprendizagem por Reforço

Capítulo 1

Introdução

Em um artigo escrito no início da década de sessenta, Marvin Minsky [101] usa uma imagem interessante para ilustrar as incertezas que rondavam a então recém-nascida disciplina de inteligência artificial. De acordo com Minsky, se um ser extraterrestre visitasse a Terra, ele ficaria confuso em relação ao papel dos computadores digitais na sociedade humana. Por um lado, o visitante ouviria falar de "cérebros mecânicos" extraordinários, capazes de desempenhar atividades intelectuais de maneira prodigiosa a ponto de ferir o orgulho dos seus ingênuos criadores antropocentristas. Por outro lado, ele se depararia com máquinas subservientes, totalmente desprovidas de iniciativa e de imaginação, condenadas a executar tarefas simples que nada exigem além das suas interpretações literais e do seu processamento mecânico.

Passado quase meio século desde a publicação do artigo de Minsky, ainda não se tem notícia de nenhum visitante interplanetário, mas a situação mudou bastante por aqui. A inteligência artificial rompeu os limites dos laboratórios para se tornar uma disciplina com grande importância prática e comercial. Atualmente, algoritmos provenientes da área são usados para auxiliar na detecção de fraudes com cartão de crédito, como consultores no diagnóstico médico, como filtros de notícias e de mensagens eletrônicas, como pilotos de veículos autônomos, como ferramentas na análise de seqüências de DNA e como controladores de robôs desenvolvidos para trabalharem em ambientes inóspitos, entre muitas outras aplicações [102, 124, 60].

O que coloca os exemplos acima em posição de destaque é o fato de o sistema computacional não ter sido diretamente programado, mas sucessivamente refinado com base em algum tipo de experiência—daí algumas vezes esse ramo da inteligência artificial ser chamado de *aprendizagem de máquina*. Nos últimos anos, prevaleceu no estudo da aprendizagem de máquina um paradigma conhecido como *aprendizagem supervisionada*. Nesse cenário o sistema adaptativo é apresentado a uma série de exemplos do que seria o comportamento correto em determinadas situações e deve generalizar a partir dessa experiência limitada. A aprendizagem supervisionada é útil em situações em que é possível fornecer exemplos de entrada e saída, mas é difícil ou impossível descrever os detalhes do mecanismo que transforma a primeira na segunda. No caso de um sistema de reconhecimento facial, por exemplo, é muito simples fornecer exemplos relacionando a imagem de uma face com a identificação desejada, embora seja dificílimo descrever o processo usado por seres humanos para estabelecer essa relação [65].

No entanto, nem todas as aplicações se enquadram no cenário estudado pela aprendizagem supervisionada. Por exemplo, se o objetivo for desenvolver um sistema inteligente para o controle de múltiplos elevadores, pode ser difícil determinar o que seria a ação correta do sistema em determinadas situações. Outros exemplos seriam um sistema de navegação para um robô ou um controlador adaptativo para uma rede de irrigação. Todos esses exemplos compartilham uma propriedade em comum: embora seja difícil determinar qual seria a ação correta em cada situação, é relativamente fácil avaliar o desempenho do sistema em um determinado intervalo de tempo. Formalmente, esse tipo de tarefa é conhecida como um *problema de tomada de decisão seqüencial*.

A abordagem clássica para solucionar problemas de tomada de decisão seqüencial é a *programação dinâmica* [128, 22]. O modelo teórico adotado por essa disciplina permite a descrição de sistemas estocásticos com dinâmica altamente não-linear. Esse paradigma é genérico o suficiente para representar problemas provenientes de diversas áreas, como controle, alocação ótima de recursos e planejamento seqüencial. De fato, a classe de problemas que podem ser resolvidos pela programação dinâmica constitui, ironicamente, uma generalização dos problemas tratados pela aprendizagem supervisionada: embora esses últimos possam ser modelados como um processo de Markov, os problemas de tomada de decisão seqüencial não podem ser solucionados pelas técnicas convencionais

de reconhecimento de padrões [136, 147]. Além disso, a programação dinâmica constitui uma abordagem confiável para lidar com problemas de controle que dificilmente seriam resolvidos por métodos tradicionais de engenharia, devido à sua dinâmica não-linear e possivelmente corrompida por ruído [128, 161]. A programação dinâmica é hoje uma disciplina madura, bem fundamentada teoricamente e exaustivamente testada na prática—em alguns casos com um enorme impacto econômico [128].

Uma restrição que limita o uso da programação dinâmica é a necessidade de um modelo exato do fenômeno estudado, ou seja, de uma descrição precisa das probabilidades de transições entre os seus estados e do custo incorrido nessas transições. Nos casos em que essa informação não está disponível, pode-se recorrer à *aprendizagem por reforço* [161, 71]. A aprendizagem por reforço originou-se na área de inteligência artificial como uma solução para o problema em que um agente deve aprender a executar uma tarefa interagindo diretamente com ela. Embora tenha sido desenvolvida de maneira independente da programação dinâmica, a aprendizagem por reforço é atualmente reconhecida como uma abordagem estatística para esta última, em que amostras de transições e recompensas substituem um modelo explícito do problema de tomada de decisão seqüencial.

A programação dinâmica e a aprendizagem por reforço com certeza causariam admiração no ET imaginado por Minsky: com essas abordagens é possível programar agentes artificiais através de recompensas e punições apenas, da mesma forma que o faria um adestrador de animais. Trata-se de um modelo poderoso, que transfere para o sistema autônomo o desafio de descobrir *como* realizar uma determinada tarefa. Engana-se quem pensa que a utilidade desses métodos se restringe a problemas artificiais criados nos laboratórios das universidades. A programação dinâmica e a aprendizagem por reforço contam hoje com várias histórias de aplicações bem-sucedidas. Os problemas de tomada de decisão citados acima, por exemplo, foram todos resolvidos na prática [35, 91, 42, 57]. Outros exemplos interessantes de aplicação incluem: a alocação dinâmica de canais de comunicação de telefones celulares [148], a manutenção da qualidade do asfalto em malhas rodoviárias [128], o agendamento de tarefas em missões da NASA [192] e o controle de descarregadores de navios [142], além do programa *TD-Gammon* de Tesauro, um sistema baseado em aprendizagem por reforço que aprendeu a jogar gamão interagindo com

o jogo e é atualmente um dos melhores jogadores do mundo [164].

Uma das características mais importantes da programação dinâmica e da aprendizagem por reforço é a sua garantia de convergência para uma política de decisão ótima [128, 22]. Isso significa que, respeitadas algumas condições básicas, há a certeza de que as soluções encontradas por essas abordagens apresentarão o melhor desempenho possível na tarefa em questão. Para se ter uma idéia do alcance desta afirmação, basta imaginar que se fosse possível aplicar essas técnicas ao jogo de xadrez, a estratégia resultante sairia vencedora em um embate hipotético em que o russo Kasparov juntasse forças com o sistema *Deep Blue* [66]. Ou então, para um exemplo menos anedótico, basta pensar que as decisões tomadas por um controlador de vôo baseado na programação dinâmica jamais resultariam em um acidente aéreo.

O que impede a concretização dos cenários acima é o fato da programação dinâmica depender de uma representação exata do fenômeno para encontrar uma solução ótima, ou seja, cada estado do problema precisa ser armazenado separadamente na memória do computador. Isso é um sério obstáculo à aplicação dessa abordagem a problemas em que o número de estados é muito grande, como no jogo de xadrez, e efetivamente impede o seu uso no caso de espaços de estados infinitos, como é o caso de um sistema para o controle de tráfego aéreo. Evidentemente, a aprendizagem por reforço, como uma versão incremental da programação dinâmica, sofre do mesmo problema de escalabilidade.

Nos casos em que o número de estados é muito grande ou infinito, é necessário apelar para algum tipo de aproximação. A solução mais direta, que seria o uso de aproximadores, pode se mostrar problemática: como os cálculos realizados na programação dinâmica têm uma natureza recorrente, um erro de aproximação pode ser facilmente reabsorvido e potencializado pelo processo de aprendizagem, causando a divergência dos algoritmos ou a sua convergência para soluções muito ruins [27, 170]. Muitos dos casos de sucesso com o uso de aproximadores na área, incluindo alguns exemplos citados acima, se devem a um grande esforço humano no ajuste de parâmetros e detalhes de implementação [163, 71]. Isso claramente desestimula a adoção dessas técnicas em larga escala.

O desenvolvimento de técnicas estáveis de programação dinâmica aproximada está no centro das discussões da comunidade científica internacional. A possibilidade de resolver

problemas de tomada de decisão de grande porte torna-se particularmente importante à medida que o ser humano constrói sistemas complexos cujas dimensões extrapolam os seus limites de controle e compreensão absolutos. Exemplos nesse sentido são a internet, as malhas de distribuição de energia e de gás, as redes de telecomunicações e os sistemas de transporte, apenas para citar alguns [147]. O fato desses sistemas complexos estarem em funcionamento não significa que estejam operando de maneira ótima—e as suas falhas eventuais são fortes indícios de que esse não seja o caso. Paul Werbos, membro da *National Science Foundation* americana, chega a afirmar que a programação dinâmica aproximada é a melhor aposta para solucionar algumas das questões fundamentais para o crescimento sustentável da humanidade [147]. Werbos acredita também que a compreensão da mente e do cérebro—a "pedra filosofal" da inteligência artificial—passa necessariamente por alguma forma de programação dinâmica aproximada.

Obviamente, a abordagem proposta neste trabalho não solucionará as aflições da humanidade, nem tampouco fornecerá um modelo definitivo do funcionamento cerebral. No entanto, acredito que ela seja uma contribuição efetiva para o esforço conjunto no sentido de estender a aplicabilidade da programação dinâmica e da aprendizagem por reforço. A premissa básica que sustenta a proposta a ser apresentada é a de que há uma redundância intrínseca a muitos problemas de tomada de decisão. Isso significa que existem padrões na dinâmica de um problema que se repetem em vários estados diferentes. Quando esse é o caso, o modelo que descreve o processo de tomada de decisão pode ser reduzido a estados "arquetípicos" que retêm toda a informação relevante a respeito do problema. Essa é a idéia básica da *fatoração estocástica*, que começa a ser apresentada a seguir. Antes de iniciar, porém, é conveniente fornecer uma visão geral de como a tese está organizada.

No Capítulo 2 discuto em detalhes os problemas de tomada de decisão seqüencial. Além de uma definição mais precisa, apresento as premissas básicas a respeito do problema e o modelo formal usado para estudar o processo de tomada de decisão. Nesse capítulo discuto também a programação dinâmica e a aprendizagem por reforço. O Capítulo 3 pode ser considerado o "núcleo teórico" da tese. É nele que apresento a fatoração estocástica, uma estratégia para reduzir o número de estados de um problema de tomada de decisão. A fatoração estocástica é abordada inicialmente de uma maneira intuitiva, como um processo de redirecionamento de transições. A seguir, esse processo é forma-

lizado como uma proposição. Ainda nesse capítulo a fatoração estocástica á analisada em detalhes, tanto do ponto de vista teórico quanto através de experimentos computacionais. Como ficará claro, embora essa abordagem seja bem fundamentada teoricamente, o seu custo computacional pode ser um obstáculo na prática. No Capítulo 4 discuto uma possível solução para essa questão. Basicamente, a solução proposta consiste em realizar a fatoração no espaço de estados do problema ao invés de lidar diretamente com as variáveis descrevendo a sua dinâmica. Dois algoritmos que utilizam essa estratégia são introduzidos. Através de uma série de experimentos apresentados no decorrer do capítulo, ficará demonstrado que tais métodos são capazes de reduzir drasticamente o custo computacional envolvido na solução de um problema de tomada de decisão. Finalmente, no Capítulo 5 apresento uma visão geral do trabalho, enfatizo as conclusões mais importantes e aponto algumas direções possíveis para trabalhos futuros.

Sugiro que a leitura do trabalho seja antecedida por uma consulta ao Apêndice A, que apresenta as principais convenções adotadas no texto.

Capítulo 2

Problemas de Tomada de Decisão Seqüencial

"As for a future life, every man must judge for himself between conflicting vague probabilities."

Charles Darwin

Em seu livro *O Homem que Confundiu sua Mulher com um Chapéu* o neurologista inglês Oliver Sacks narra, além do evento que deu título ao livro, várias outras histórias fascinantes sobre pacientes com disfunções neurológicas raras [138]. Uma delas é a história de Jimmie G., um homem de 45 anos que sofria de uma patologia conhecida como síndrome de Korsakov. Embora Jimmie apresentasse a maioria das faculdades mentais intactas—de acordo com Sacks, ele era "um homem esperto, observador e lógico"—, o seu cérebro era incapaz de registrar qualquer acontecimento novo por mais de alguns minutos. Mesmo após tê-lo examinado várias vezes e ter inclusive adquirido um certo grau de empatia com o paciente, Sacks tinha que se apresentar como se fosse um completo desconhecido a cada nova sessão. Em uma dessas sessões, Sacks pediu a Jimmie que anotasse o nome de alguns objetos em um pedaço de papel. Após alguns minutos, o médico perguntou ao paciente se ele seria capaz de dizer o nome dos objetos anotados e, para a sua surpresa, ele não se lembrava nem mesmo de *ter anotado* alguma coisa. Essa perda extrema de memória recente confinava Jimmie a um ponto específico da sua história; ele

estava “isolado num momento único da existência, rodeado por um fosso ou lacuna de esquecimento...”[1]

De todos os aspectos extraordinários da história de Jimmie G., um tem uma relevância particular para o assunto que será tratado neste capítulo. Sacks conta que Jimmie era muito habilidoso na solução de problemas e quebra-cabeças complexos, derrotando-o facilmente em jogos como o da dama e o jogo-da-velha. No entanto, ele era incapaz de repetir a proeza no jogo de xadrez. Por que exatamente isso acontecia? Sacks atribui a inabilidade de Jimmie à lentidão dos movimentos do xadrez. Como um jogo de xadrez típico se estende por um período que extrapola a sua capacidade de memorização, o paciente era incapaz de associar uma jogada com as suas conseqüências vários passos à frente. A habilidade de Jimmie estava restrita, portanto, aos desafios que “coubessem” na janela de tempo que definia a sua existência.

O jogo de xadrez é um exemplo típico de um tipo de problema chamado formalmente de *problema de tomada de decisão seqüencial*. A característica fundamental dessa classe de problemas é que as decisões envolvidas têm um efeito cumulativo, ou seja, as conseqüências de uma determinada ação podem se estender por um intervalo indefinido de tempo. Essa definição engloba uma grande quantidade de problemas. De fato, é difícil imaginar uma situação real em que a execução de uma ação não tenha um efeito a longo prazo (a imagem da borboleta que bate as asas causando um tornado já é quase um clichê). No entanto, neste trabalho pretendo me concentrar em um tipo especial de problema de tomada de decisão, a saber, aqueles que podem ser modelados como um processo de decisão de Markov. De uma maneira superficial, pode-se dizer que o que caracteriza um problema desse tipo é o fato de todas as informações relevantes estarem disponíveis no momento da tomada de decisão. O xadrez, por exemplo, se enquadra nessa definição.

Os processos de decisão de Markov são apresentados formalmente na Seção 2.1, onde também discuto como avaliar uma seqüência de decisões. A abordagem normalmente adotada para lidar com esse tipo de problema é a programação dinâmica, discutida na

[1] O filme *Memento*, dirigido por Christopher Nolan, conta de maneira brilhante a história de um personagem que sofre da mesma disfunção de Jimmie. Para dar alguma coerência à sua vida, o personagem se comunica com ele mesmo através de bilhetes, tatuagens e fotos *Polaroids*. Não deixa de ser curioso o fato de o assassino procurado pelo personagem do filme se chamar “John ou James G.”

Seção 2.2. A programação dinâmica é uma disciplina bem fundamentada teoricamente e muito testada na prática. No entanto, ela apresenta um sério inconveniente: a sua aplicação depende de um modelo exato do processo de tomada de decisão. Quando um modelo não está disponível, pode-se utilizar os métodos provenientes da aprendizagem por reforço. Como será discutido na Seção 2.3, esses métodos podem ser vistos como a programação dinâmica feita de forma incremental. A programação dinâmica é considerada a única abordagem viável com garantias teóricas em relação à solução encontrada para um problema de tomada de decisão. No entanto, ela sofre de um sério problema de escalabilidade, tanto originalmente quanto na sua versão incremental. Quando o problema é grande, é necessário recorrer a algum tipo de aproximação. A Seção 2.4 apresenta algumas possibilidades nesse sentido. Finalmente, na Seção 2.5 discuto rapidamente outras abordagens que podem ser usadas para resolver um problema de tomada de decisão seqüencial.

2.1 Processos de Decisão de Markov

Para que seja possível lidar com problemas de decisão seqüenciais de uma maneira sistemática é necessário estabelecer um modelo que descreva formalmente como ocorre o processo de tomada de decisão. No modelo adotado neste trabalho as decisões são tomadas por um *agente* que interage com um *ambiente*. Note que esses são termos genéricos que englobam algumas nomenclaturas bem estabelecidas como casos particulares. No caso de problemas de controle, por exemplo, o agente seria o controlador e o ambiente seria a planta. Quando se lida com jogos, por outro lado, o agente é um jogador e o ambiente é o jogo em questão. Em geral a interação do agente com o ambiente se dá de maneira discreta, ou seja, o agente deve tomar as decisões em instantes de tempo específicos $t = 1, 2, ..., T$. A cada instante de tempo t o agente se encontra em um *estado* s_i e deve selecionar uma *ação* a de um conjunto de ações disponíveis. A execução da ação a no estado s_i move o agente para um novo estado s_j, onde ele deve escolher novamente uma ação, reiniciando o ciclo. A interação do agente com o ambiente pode ocorrer indefinidamente ($T = \infty$) ou até que ele alcance um estado terminal ($T < \infty$). No caso em que $T < \infty$, a seqüência de ações executadas pelo agente de um estado inicial qualquer

até um estado terminal é chamada de *episódio*.

O conjunto formado por todos os estados possíveis do ambiente é chamado de *espaço de estados* e normalmente denotado por S. Neste capítulo irei me concentrar no caso em que o espaço S é finito, embora potencialmente muito grande. No entanto, a maioria das idéias se aplicam sem modificações ao caso em que esse espaço é um conjunto infinito enumerável [128]. No Capítulo 4 discuto espaços de estados contínuos. O conjunto de ações disponíveis no estado s_i é denotado por $A(s_i)$, aqui também uma coleção finita de elementos. O conjunto $A = \cup_{s_i \in S} A(s_i)$ constitui o *espaço de ações*. Para simplificar a notação, vou considerar que $A(s_i) = A$ para todo $s_i \in S$. O modelo adotado pode ser generalizado permitindo-se que S e A variem com t. Embora essa modificação tenha pouco efeito sobre a teoria, ela constitui uma complicação desnecessária, uma vez que não traz nenhum benefício para a maioria das aplicações práticas [128].

Observe que o modelo descrito acima não representa de fato um processo de tomada de decisão, porque não há um critério definido para avaliar as decisões tomadas pelo agente. Para que uma escolha se caracterize como uma decisão é necessário estabelecer uma maneira de comparar as diferentes alternativas. É aí que entra o conceito de *recompensa*. Uma recompensa é um valor escalar que serve como medida da qualidade de uma decisão. Quanto maior a recompensa, melhor para o agente.[2] No modelo adotado aqui as recompensas são entregues logo após a execução de uma ação, ou seja, a cada transição $s_i \xrightarrow{a} s_j$ o agente recebe uma recompensa $r \in \mathbb{R}$. O objetivo do agente é maximizar a quantidade total de recompensas recebidas na interação com o ambiente.

A Figura 2.1 mostra de maneira esquemática o processo de interação do agente com o ambiente. Note que, apesar de simples, esse modelo absorve de maneira natural as tensões envolvidas em problemas de tomada de decisão seqüencial, em que benefícios imediatos devem ser confrontados com as suas conseqüências a longo prazo. No caso do jogo de xadrez, por exemplo, é razoável associar uma recompensa com a obtenção de cada peça do adversário. Parece lógico, no entanto, que a recompensa associada ao rei deva ser maior do que a soma de todas as outras—afinal de contas, a captura do rei adversário caracteriza a vitória do jogo. Como o que importa para o agente é o total de recompensas

[2] Quando a recompensa tem uma conotação negativa, ela é às vezes chamada de *custo*. Obviamente, o objetivo do agente nesse caso é minimizar o custo.

recolhidas a longo prazo, ele seria capaz de abrir mão de uma peça valiosa se uma jogada alternativa, mesmo que aparentemente ingênua, resultasse em uma maior probabilidade de capturar o rei adversário.

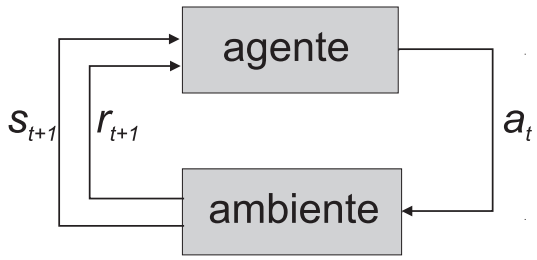


Figura 2.1: Modelo de tomada de decisão.

O modelo de tomada de decisão mostrado na Figura 2.1 é também bastante flexível, podendo ser usado para descrever as mais diversas situações. Os estados do sistema, por exemplo, podem ser desde a leitura direta de sensores até a descrição simbólica de uma situação abstrata. Da mesma forma, as ações podem se referir a controles físicos ou atividades estritamente "mentais." No controle de um braço mecânico, por exemplo, os estados podem ser vetores numéricos com informações a respeito da posição do braço ou, no outro extremo, uma descrição simbólica do tipo "braço esticado" ou "braço dobrado." De forma equivalente, as ações poderiam ser o torque a ser aplicado em cada articulação do braço mecânico ou ações abstratas do tipo "esticar" ou "dobrar." Estendendo um pouco o raciocínio, pode-se facilmente imaginar uma situação em que uma das ações disponíveis seja "focar a atenção" em um determinado aspecto do problema. Como observam Sutton e Barto [161], as ações podem representar qualquer decisão que se pretenda aprender a tomar, e os estados qualquer informação potencialmente útil nesse processo. Por isso, algumas vezes irei me referir ao processo de solução de um problema de tomada de decisão simplesmente como *aprendizagem*.

É importante ressaltar que os limites entre agente e ambiente não precisam coincidir com os limites físicos de um sistema, e devem ser definidos de acordo com os propósitos da tarefa em questão. Em geral, tudo o que estiver fora do controle absoluto do agente deve ser considerado como parte do ambiente. Considere que o problema de tomada de decisão seja dirigir um automóvel, por exemplo. Nesse caso o agente não pode intervir diretamente no tanque de combustível; a única maneira de evitar uma potencial falta de

combustível é executar uma ação (ou uma seqüência de ações) para reabastecer. Portanto, nesse exemplo hipotético o tanque de combustível seria parte do ambiente tanto quanto as ruas e os sinais de trânsito.

Em alguns casos é possível desmembrar um problema em uma hierarquia de tarefas, de forma que as ações disponíveis em um determinado nível correspondam a uma seqüência de ações em um nível inferior [9]. No caso do automóvel, por exemplo, a ação "virar à esquerda" poderia disparar uma série de ações envolvendo desde a sinalização de advertência até o movimento do volante propriamente dito. Essa estratégia de criar "macro-ações" pode simplificar de maneira significativa tarefas em que a mesma seqüência de ações é utilizada em diferentes situações. Observe que se um esquema hierárquico for adotado, o tempo de duração de cada ação pode ser diferente, dependendo do comprimento da seqüência de ações executadas no nível inferior. Isso ilustra uma flexibilidade extra do modelo: a noção de tempo do ponto de vista do agente não precisa coincidir com a passagem de tempo real. Em outras palavras, o intervalo de tempo transcorrido entre os instantes t e $t+1$ pode ser diferente daquele transcorrido entre $t+1$ e $t+2$.

Embora o modelo de tomada de decisão descrito acima seja bastante genérico, existem alguns cenários que não podem ser representados por ele de maneira adequada. Esse é o caso, por exemplo, de problemas em que a interação do agente com o ambiente se dá ininterruptamente. É possível modificar o modelo adotado para contemplar essa situação, mas esse assunto foge ao escopo deste trabalho. Para uma descrição detalhada dessa e outras generalizações do modelo acima, o leitor pode recorrer ao excelente livro de Puterman [128].

2.1.1 A propriedade markoviana

A discussão acima pode causar a falsa impressão de que a dinâmica de um ambiente é necessariamente determinística, ou seja, de que a execução de uma determinada ação a em um estado s_i resulta sempre no mesmo estado s_j. Embora em alguns casos isso seja de fato verdade, de uma maneira geral a evolução de um sistema segue uma dinâmica estocástica—o que equivale a dizer que existe um certo grau de incerteza associado a cada

transição. É por isso que os problemas estudados neste capítulo são às vezes chamados de *problemas de tomada de decisão seqüencial envolvendo incerteza*. Trata-se de uma nomenclatura precisa, não há dúvida, mas longa demais. Daqui para frente, fica entendido que o termo "problema de tomada de decisão" se refere ao caso acima.

Matematicamente, a incerteza é incorporada ao modelo de tomada de decisão através de uma distribuição de probabilidades. Isso significa que o estado s_j que segue a execução da ação a no estado s_i é amostrado de uma distribuição definida sobre o espaço de estados S. Em outras palavras, existe uma função P que atribui a cada estado $s_j \in S$ um valor no intervalo $[0, 1]$ representando a sua probabilidade de seguir a transição. A função P é chamada de *função de probabilidade de transição*. No caso mais geral a função de probabilidade de transição depende de toda a história do sistema até o instante t, o que torna a descrição do modelo uma tarefa complexa do ponto de vista computacional. Felizmente, sabe-se que em muitos casos é possível fazer uma simplificação substancial que não acarreta qualquer perda de informação. Isso ocorre quando a dinâmica de um sistema apresenta a *propriedade markoviana*. Em um sistema markoviano a definição do estado s_j que segue a execução da ação a no estado s_i depende unicamente destes dois últimos, ou seja, a probabilidade de ocorrência do estado s_j é dada por uma função $P(s_j|s_i, a)$. Como aqui serão considerados apenas espaços de ações finitos, é possível desmembrar a função P em uma família de funções P^a, uma para cada ação $a \in A$. Nesse caso,

$$P^a(s_j|s_i) = P(s_j|s_i, a).$$

Essa modificação não é essencial, mas ela simplifica consideravelmente a discussão subseqüente. Em geral, considera-se que

$$\sum_{j=1}^{|S|} P^a(s_j|s_i) = 1 \text{ para todo } a \in A.$$

Embora essa restrição não seja estritamente necessária [128], no que segue vou considerar que ela seja atendida.

Em um sistema markoviano a recompensa que segue uma transição também independe das transições anteriores. Na formulação mais geral do modelo de tomada de decisão a recompensa r associada com a transição $s_i \xrightarrow{a} s_j$ depende não apenas de s_i

e a, como também de s_j. Nesse caso, considera-se que as recompensas sejam dadas por uma família de funções $R^a : S \times S \mapsto \mathbb{R}$ definidas de maneira análoga às funções P^a. Especificamente, a recompensa resultante da transição $s_i \xrightarrow{a} s_j$ seria dada por $R^a(s_i, s_j)$. Para garantir a regularidade do modelo, normalmente exige-se que as funções R^a sejam limitadas, isso é,

$$|R^a(s_i, s_j)| \leq r_{\max} < \infty, \text{ para todo } s_i, s_j \in S \text{ e todo } a \in A.$$

Observe que como não se pode saber de antemão o estado resultante de uma transição (a não ser em problemas determinísticos), não faz muito sentido do ponto de vista do agente distinguir as recompensas de acordo com o estado s_j. Freqüentemente, o que importa para a solução de problemas de tomada de decisão seqüencial é o *valor esperado* da recompensa associada com s_i e a, dado por

$$r^a(s_i) = \sum_{j=1}^{|S|} R^a(s_i, s_j) P^a(s_j|s_i). \tag{2.1}$$

No caso de problemas de tomada de decisão episódicos, em que o agente interage com o ambiente por um período finito de tempo, pode-se incluir um estado terminal s_i ao modelo simplesmente fazendo $P^a(s_i|s_i) = 1$ e $r^a(s_i) = 0$, para todo $a \in A$ [161].

A família de funções P^a e r^a definem um *processo de decisão de Markov*, ou simplesmente MDP[†]. Os MDPs fornecem uma descrição formal de um problema de tomada de decisão seqüencial. Eles constituem o modelo fundamental que serviu de base para o desenvolvimento de toda a teoria da programação dinâmica, que será estudada na próxima seção. Neste trabalho serão considerados apenas problemas que podem ser modelados como um MDP. Fica entendido, portanto, que o termo "problema de tomada de decisão seqüencial" se refere a problemas que atendam à propriedade markoviana.

Quando o espaço de estados S é um conjunto finito, é possível representar as funções $P^a : S \times S \mapsto [0, 1]$ e $r^a : S \mapsto \mathbb{R}$ de forma matricial. Nesse caso, cada função P^a é uma matriz $\mathbf{P}^a \in \mathbb{R}^{|S| \times |S|}$. O elemento p^a_{ij} representa a probabilidade de transição do estado s_i para o estado s_j quando a ação a é executada, ou seja, $p^a_{ij} = P^a(s_j|s_i)$. Note que os elementos em uma linha de $\mathbf{P}^a$ são não-negativos e têm soma 1. As matrizes com

[†]*Markov decision process* (ao longo do trabalho adoto a convenção de utilizar as siglas em inglês, para facilitar a correspondência com a literatura internacional. Veja o Apêndice A para detalhes).

essa propriedade são chamadas de *matrizes linha-estocásticas* ou simplesmente *matrizes estocásticas*. As matrizes estocásticas irão exercer um papel fundamental nos resultados teóricos do Capítulo 3. No caso de um MDP, cada matriz estocástica $\mathbf{P}^a$ é uma *matriz de transições*. De maneira análoga, pode-se definir os *vetores-recompensa* $\mathbf{r}^a \in \mathbb{R}^{|S|}$, em que $r_i^a = r^a(s_i)$. Note, portanto, que um MDP finito fica definido por um conjunto de $|A|$ matrizes $\mathbf{P}^a \in \mathbb{R}^{|S|\times|S|}$ e o mesmo número de vetores $\mathbf{r}^a \in \mathbb{R}^{|S|}$.

Uma premissa básica quando se usa os MDPs como um modelo de tomada de decisão é que a dinâmica do sistema subjacente atenda à propriedade markoviana. É importante que se compreenda exatamente o que significa essa suposição. Considerar que um determinado sistema seja markoviano equivale a dizer que os seus estados contêm toda a informação requerida para a descrição da sua dinâmica. Voltando ao exemplo do automóvel, suponha que ao tentar dirigir um veículo o agente tenha informação a respeito da sua posição e velocidade. Nesse caso, é possível prever com bastante precisão o efeito das ações de acelerar ou frear, por exemplo. Como em geral é impraticável levar em consideração todas as forças agindo sobre o veículo, é conveniente descrever a sua dinâmica em termos de probabilidades de transição. Imagine agora que o velocímetro do automóvel não esteja funcionando. Sem a informação sobre a velocidade, é impossível determinar precisamente qual seria o efeito de acelerar ou frear o veículo. Ou seja, os estados do sistema deixam de ser markovianos.

Note que é possível "construir" um estado markoviano utilizando um mecanismo de memória. No exemplo dado, bastaria guardar duas posições sucessivas do automóvel para se ter uma estimativa da sua velocidade. Pode-se argumentar, portanto, que a propriedade markoviana é uma característica do modelo, e não do sistema modelado. No entanto, é difícil afirmar com certeza se nos problemas de interesse seria sempre possível formular um estado markoviano. Essa é, inclusive, uma questão com implicações filosóficas! Voltarei a esse assunto no Capítulo 4.

2.1.2 Políticas de decisão

A estratégia usada pelo agente para selecionar as ações pode ser formalizada como uma *política de decisão*. Uma política de decisão é essencialmente uma função π que transforma informação a respeito do ambiente em ações. Pode-se definir diferentes categorias de políticas, de acordo com a informação usada e o valor retornado por essa função. Se a seleção da ação a ser executada no estado s_i levar em consideração todas ou parte das transições executadas até então, a política é dita dependente do histórico. Talvez não muito surpreendentemente, se a decisão de π for baseada unicamente em s_i, a política é dita markoviana. Além disso, π pode ser classificada como determinística ou estocástica. No primeiro caso ela retorna uma ação a específica em cada estado s_i, enquanto no segundo caso ela retorna uma distribuição de probabilidades sobre o espaço de ações A.

As políticas mais genéricas são as políticas estocásticas dependentes do histórico de transições. No outro extremo encontram-se as políticas determinísticas markovianas. Note que esta última categoria representa um caso particular da primeira. É razoável supor que o melhor desempenho em um MDP só seja alcançado se a busca por uma política ótima for realizada no espaço mais genérico possível. Felizmente, esse não é o caso. É sabido que no modelo adotado neste trabalho existe uma política markoviana determinística cujo desempenho não pode ser superado por nenhuma outra política [128, 22]. Na próxima seção deixarei claro o critério usado para avaliar o desempenho de uma política de decisão.

Uma política markoviana determinística é uma função $\pi : S \mapsto A$ que associa cada estado s_i a uma ação a. As probabilidades de transição de um agente que se encontra no estado s_i e age de acordo com π são dadas por $P^{\pi(s_i)}(\cdot|s_i)$, onde $\pi(s_i) \in A$ é a ação selecionada por π no estado s_i. Seguindo raciocínio análogo, pode-se dizer que o valor esperado da recompensa a ser recebida nesse estado é $r^{\pi(s_i)}(s_i)$. As funções P^π e r^π descrevendo o comportamento da política π dão origem a um *processo de Markov*. Um processo de Markov nada mais é do que um MDP em que não existem decisões a serem tomadas. Note que as funções P^a e r^a correspondem ao processo de Markov que seria induzido pela política $\pi(s_i) = a$ para todo $s_i \in S$. Algumas vezes refere-se às matrizes $\mathbf{P}^\pi$ isoladamente como *cadeias de Markov*.

No caso de um MDP finito, a política π pode ser representada por um vetor em $A^{|S|}$ em que o i-ésimo elemento é a ação a ser executada no estado s_i. Além disso, pode-se definir a matriz $\mathbf{P}^\pi \in \mathbb{R}^{|S|\times|S|}$ descrevendo as probabilidades de transição da política π. A i-ésima linha dessa matriz corresponderia à linha de mesma ordem da matriz $\mathbf{P}^{\pi(s_i)}$, ou seja, $\mathbf{p}_i^\pi = \mathbf{p}_i^{\pi(s_i)}$. Da mesma maneira, os elementos do vetor $\mathbf{r}^\pi$ seriam dados por $r_i^\pi = r_i^{\pi(s_i)}$. A matriz $\mathbf{P}^\pi$ e o vetor $\mathbf{r}^\pi$ definem um processo de Markov finito.

2.1.3 Objetivo do problema de tomada de decisão

Na seção anterior afirmei que a busca por uma política ótima pode se restringir ao espaço das políticas markovianas determinísticas. Para que seja possível caracterizar uma política ótima, no entanto, é necessário estabelecer *o quê* exatamente se pretende otimizar. O objetivo de um problema de tomada de decisão seqüencial é encontrar a política π^* que maximize as recompensas recolhidas a longo prazo. Note, no entanto, que essa afirmação é muito vaga para ser usada em um problema de otimização. No restante desta seção tentarei reformulá-la em termos mais rigorosos.

A definição precisa do que seria o objetivo em um problema de tomada de decisão requer uma certa elaboração. Para que a tarefa de tomada de decisão possa ser interpretada como um problema de otimização, é necessário definir uma função-objetivo a ser otimizada. Suponha por enquanto que o objetivo do agente seja maximizar uma função das recompensas recebidas a partir de um determinado estado s_i. Uma escolha óbvia para a função-objetivo seria a soma das recompensas recolhidas por π a partir de s_i:

$$R^\pi(s_i) = r_1^i + r_2^i + ... + r_T^i = \sum_{t=1}^{T} r_t^i, \tag{2.2}$$

onde r_t^i é a recompensa recebida por π na t-ésima transição da sua interação com o ambiente iniciada em s_i. Note, porém, que a função acima pode se tornar problemática nos casos em que $T = \infty$, ou seja, quando a interação do agente com o ambiente não puder ser naturalmente dividida em episódios. Nesse caso, se as recompensas forem todas positivas, por exemplo, torna-se impossível diferenciar as políticas π—uma vez que todas elas teriam uma avaliação $R^\pi(s_i) = \infty$. Mesmo no caso de tarefas episódicas a função (2.2) pode causar dificuldades quando existirem políticas de decisão que nunca atinjam um

estado terminal [128, 22].

Pode-se impor algumas restrições à série (2.2) de forma a garantir que ela seja limitada mesmo quando $T = \infty$. Com isso é possível definir categorias de MDPs para os quais essa série é uma medida adequada [128]. No entanto, uma solução mais genérica que se aplica a qualquer MDP é introduzir na função-objetivo um *fator de desconto* γ. Com ele, a nova versão da função a ser maximizada ficaria assim:

$$R_{\gamma}^{\pi}(s_i) = r_1^i + \gamma r_2^i + \gamma^2 r_3^i + ... + \gamma^{T-1} r_T^i = \sum_{t=1}^{T} \gamma^{t-1} r_t^i, \tag{2.3}$$

onde $0 \leq \gamma < 1$. O fator de desconto γ transforma (2.2) em uma série geométrica que tem convergência garantida para qualquer política π [89]. É interessante notar, entretanto, que esse parâmetro não é apenas um "truque" matemático. Em um problema de tomada de decisão financeiro, por exemplo, o parâmetro γ cumpriria naturalmente o papel da taxa de juros. O fator de desconto pode ser interpretado também como um recurso que permite incorporar ao modelo uma medida da incerteza em relação ao futuro. Sob esse ponto de vista, o parâmetro γ representaria a possibilidade de a série (2.3) ser interrompida por eventos como falência de uma firma em um modelo econômico, falha do sistema em um modelo de produção ou morte de um animal em um modelo ecológico [128].

Embora a série (2.3) seja mais bem-comportada do que (2.2), ela ainda não constitui uma escolha adequada como função-objetivo para o processo de otimização. Isso porque essa série representa o desempenho da política π em uma *interação específica* com o MDP. Lembre-se que as transições de um MDP não são necessariamente determinísticas, e portanto (2.3) pode ser uma medida bastante distorcida da verdadeira qualidade de π. Para enxergar isso, imagine que o problema em questão seja o jogo de xadrez usado como exemplo no início deste capítulo. Suponha que π represente a estratégia de jogo usada pelo paciente Jimmie. Usar (2.3) como critério de avaliação seria o mesmo que julgar a qualidade do xadrez de Jimmie baseando-se em um único jogo. Não é preciso um Oliver Sacks para perceber que se trata de uma medida inadequada.

Fica claro, portanto, que a avaliação de uma política π deve refletir a *expectativa* em relação ao seu desempenho no MDP. Matematicamente, isso corresponde ao valor esperado de $R_{\gamma}(s_i)$ calculado em relação à política π. O valor esperado $E^{\pi}\{R_{\gamma}(s_i)\}$ é a

soma de todos os valores possíveis de (2.3), cada um ponderado pela sua probabilidade de ocorrência. Ele pode ser escrito como:

$$E^{\pi}\{R_{\gamma}(s_i)\} = E^{\pi}\left\{\sum_{t=1}^{T}\gamma^{t-1}r_t^i\right\} = \sum_{t=1}^{T}\gamma^{t-1}E^{\pi}\{r_t^i\}. \tag{2.4}$$

O valor esperado da recompensa recebida na primeira transição de π é $r^{\pi(s_i)}(s_i)$. Seguindo a mesma lógica usada em (2.1), pode-se calcular o valor esperado da recompensa recebida por π na segunda transição:

$$E^{\pi}\{r_2^i\} = \sum_{j=1}^{|S|}P^{\pi(s_i)}(s_j|s_i)r^{\pi(s_j)}(s_j). \tag{2.5}$$

Indo um passo além, pode-se escrever

$$E^{\pi}\{r_3^i\} = \sum_{j=1}^{|S|}P^{\pi(s_i)}(s_j|s_i)\sum_{k=1}^{|S|}P^{\pi(s_j)}(s_k|s_j)r^{\pi(s_k)}(s_k), \tag{2.6}$$

que fornece o valor esperado da recompensa recebida na terceira transição de π no MDP. Prosseguindo com esse raciocínio, é possível (pelo menos em princípio) calcular a expectativa em relação à recompensa recebida por π em cada transição futura. Uma maneira intuitiva de enxergar a expressão (2.4) é a seguinte: imagine que um agente seguindo a política π executasse um número infinito de episódios, todos eles iniciados no estado s_i. Nesse caso, $E^{\pi}\{R_{\gamma}(s_i)\}$ representaria o valor médio de (2.3) em todos esses episódios. No caso de tarefas não-episódicas, a interação do agente com o ambiente se reduziria a um único episódio de comprimento infinito, e $E^{\pi}\{R_{\gamma}(s_i)\}$ seria a soma descontada das recompensas recebidas assintoticamente a partir de s_i.

A função (2.4) constitui uma medida adequada para avaliar o desempenho de uma política π. Do ponto de vista prático, no entanto, o seu uso depende da solução de duas questões. A primeira delas é óbvia: como efetivamente calcular (2.4)? Executar um número infinito de episódios com a política π está evidentemente fora de questão. Além disso, a expressão (2.4) representa o desempenho de π *a partir de um estado específico*. Em geral, o objetivo em um problema de tomada de decisão é encontrar uma política que apresente um bom desempenho em todo o espaço de estados S. Existiria uma política π^* que maximiza (2.4) para todo $s_i \in S$? Essas duas perguntas começam a ser respondidas na próxima seção.

2.2 Programação Dinâmica

A programação dinâmica é amplamente reconhecida como a única solução para problemas de tomada de decisão seqüencial que é ao mesmo tempo bem fundamentada teoricamente e viável computacionalmente [128, 161, 147]. Ela se baseia fortemente no conceito de função de valor, que reflete o desempenho de uma política de decisão em um horizonte de tempo potencialmente infinito. Como será discutido, todos os algoritmos da programação dinâmica podem ser vistos como uma coevolução entre uma função de valor e uma política, que competem localmente mas cooperam uma com a outra em uma escala maior.

2.2.1 Função de valor

Um conceito absolutamente imprescindível para a teoria da programação dinâmica é aquele de uma *função de valor*. A função de valor de uma política π associa a cada estado s_i do espaço S o valor esperado da série de recompensas recolhidas por π a partir desse estado. Mais sucintamente, pode-se escrever:

$$\begin{aligned} V^{\pi}: \quad & S \mapsto \mathbb{R} \\ & V^{\pi}(s_i) = E^{\pi}\{R_{\gamma}(s_i)\}. \end{aligned} \tag{2.7}$$

É a partir do conceito de função de valor que Bellman estabeleceu uma relação recursiva entre os estados de um MDP que ficou conhecida como a *equação de Bellman* [13]. A equação de Bellman constitui o cerne dos algoritmos de programação dinâmica. É ela que torna a busca pela política ótima de um MDP um processo computacionalmente viável. No que segue mostrarei como essa equação pode ser derivada para uma política π.

Equação de Bellman

Considere que o MDP em questão tenha um espaço de estados finito e horizonte de interação infinito, ou seja, na derivação abaixo $|S| < \infty$ e $T = \infty$ (os demais casos podem ser derivados seguindo raciocínio análogo, desde que se faça os ajustes necessários). A

partir de (2.4) e (2.7) pode-se escrever

$$\begin{aligned} V^{\pi}(s_i) &= E^{\pi}\left\{r_1^i + \gamma r_2^i + \gamma^2 r_3^i + ...\right\} \\ &= E^{\pi}\left\{r_1^i\right\} + \gamma E^{\pi}\left\{r_2^i + \gamma r_3^i + \gamma^2 r_4^i + ...\right\}. \end{aligned} \tag{2.8}$$

Observe que o termo $E^{\pi}\left\{r_2^i + \gamma r_3^i + ...\right\}$ é equivalente à expressão (2.4), mas nesse caso a seqüência de recompensas é considerada a partir da segunda transição do agente. Uma maneira de interpretar esse termo é imaginar a série (2.4) sendo iniciada em vários estados s_j diferentes. A probabilidade de a seqüência de transições se iniciar no estado s_j corresponde à probabilidade de transição de s_i para s_j quando a ação $\pi(s_i)$ é executada no primeiro. Seguindo esse raciocínio, pode-se reescrever (2.8) da seguinte maneira:

$$\begin{aligned} V^{\pi}(s_i) &= E^{\pi}\left\{r_1^i\right\} + \gamma \textstyle\sum_{j=1}^{|S|} P^{\pi(s_i)}(s_j|s_i) E^{\pi}\left\{r_1^j + \gamma r_2^j + \gamma^2 r_3^j + ...\right\} \\ &= E^{\pi}\left\{r_1^i\right\} + \gamma \textstyle\sum_{j=1}^{|S|} P^{\pi(s_i)}(s_j|s_i) E^{\pi}\{R_{\gamma}(s_j)\}. \end{aligned}$$

Lembrando que $E^{\pi}\left\{r_1^i\right\} = r^{\pi(s_i)}(s_i)$ e observando (2.7), a expressão acima se torna

$$V^{\pi}(s_i) = r^{\pi(s_i)}(s_i) + \gamma \textstyle\sum_{j=1}^{|S|} P^{\pi(s_i)}(s_j|s_i) V^{\pi}(s_j), \tag{2.9}$$

que é a equação de Bellman da política π.

No caso de espaços de estados finitos, a equação de Bellman pode ser escrita de forma vetorial:

$$\mathbf{v}^{\pi} = \mathbf{r}^{\pi} + \gamma \mathbf{P}^{\pi} \mathbf{v}^{\pi}, \tag{2.10}$$

onde $\mathbf{v}^{\pi}$ é um vetor em $\mathbb{R}^{|S|}$ em que $v_i^{\pi} = V^{\pi}(s_i)$. A expressão (2.10) deixa evidente que a equação de Bellman de uma política π é na verdade um sistema linear com $|S|$ equações. Logo, a função de valor de π pode ser facilmente determinada como:

$$\mathbf{v}^{\pi} = (\mathbf{I} - \gamma \mathbf{P}^{\pi})^{-1} \mathbf{r}^{\pi}, \tag{2.11}$$

onde $\mathbf{I}$ é a matriz identidade de dimensão $|S|$. Os fatos de $\mathbf{P}^{\pi}$ ser uma matriz estocástica e $0 \leq \gamma < 1$ garantem a existência e unicidade de uma solução para o sistema acima [50, 22]. Note que a equação (2.11) soluciona uma das questões levantadas no final da Seção 2.1.3: a partir dela, é possível determinar $E^{\pi}\{R_{\gamma}(s_i)\}$ para todos os estados s_i do espaço de estados. Resta saber como comparar duas políticas com base nas suas funções de valor.

Políticas ótimas de um MDP

Sejam π_1 e π_2 duas políticas definidas sobre o mesmo MDP. Como o objetivo do problema de tomada de decisão é encontrar uma política que maximize o valor esperado das recompensas coletadas, pode-se usar as funções V^{π_1} e V^{π_2} como critério para compará-las. Uma maneira "conservadora" de fazê-lo seria a seguinte:

$$\pi_1 \geq \pi_2 \iff V^{\pi_1}(s_i) \geq V^{\pi_2}(s_i) \text{ para todo } s_i \in S. \tag{2.12}$$

Adicionalmente, $\pi_1 > \pi_2$ se (2.12) se cumpre e $V^{\pi_1}(s_i) > V^{\pi_2}(s_i)$ para pelo menos um $s_i \in S$. Note que a expressão (2.12) estabelece apenas uma relação de ordem parcial no espaço das políticas, uma vez que nem todos os pares de elementos desse conjunto podem ser diretamente comparados [80]. Por exemplo, é possível que $V^{\pi_1}(s_1) > V^{\pi_2}(s_1)$, mas $V^{\pi_1}(s_2) < V^{\pi_2}(s_2)$. Nesse caso nada pode ser dito a respeito da ordem de π_1 e π_2 segundo o critério adotado. Pode-se suspeitar, portanto, que não exista uma *única* política ótima π^* que maximiza (2.4) sobre S, mas um conjunto formado por políticas que não podem ser superadas em todo o espaço de estados. Se esse for o caso, afirmar que a política π pertence ao conjunto das políticas ótimas significa dizer que não existe $\pi_j > \pi$, embora possam existir políticas que superem π em *alguns estados* do espaço de estados (*i.e.*, $V^{\pi_j}(s_i) > V^{\pi}(s_i)$ para alguns estados $s_i \in S$, mas não todos).[3]

Tentarei mostrar de uma maneira intuitiva por que existe pelo menos uma política que maximiza (2.4) em todo o espaço de estados. Suponha por um momento que isso não seja verdade. Então, existe um conjunto de políticas ótimas em que cada representante é, por assim dizer, "especializada" em uma região específica de S—ou seja, a escolha por uma dessas políticas necessariamente sacrifica o desempenho do agente nas demais regiões do espaço de estados. Imagine agora que se conheça a região em que cada política se comporta de maneira ótima. Ora, se esse é o caso, pode-se simplesmente "trocar" de política de acordo com a região em que se encontra o agente, sempre escolhendo aquela que maximiza as recompensas coletadas. Como essa política híbrida é ela mesma uma política válida, deve existir pelo menos uma política ótima que maximiza as recompensas em todo o espaço S. Note que, caso exista um conjunto de políticas com essa propriedade,

[3] O critério (2.12) coincide com a noção de *dominância* usada em problemas de otimização multiobjetivo [45]. Nesse contexto o conjunto das políticas ótimas seria chamado de *fronteira de Pareto*.

todas elas compartilham a mesma função de valor V^*.

2.2.2 Iteração de política

O raciocínio acima deixa claro que existe pelo menos uma política π^* tal que $\pi^* \geq \pi_i$, qualquer que seja π_i. A pergunta óbvia, nesse caso, é como encontrar uma dessas políticas. Existem $|A|^{|S|}$ políticas de decisão diferentes em um MDP finito. Logo, é fácil projetar um algoritmo que retorne uma política ótima para esse MDP: basta gerar as políticas uma-a-uma, calcular as suas funções de valor usando (2.11) e então compará-las segundo (2.12). Embora esse algoritmo encontre uma política π^* em um número finito de passos, não é difícil perceber que um MDP de tamanho modesto já tornaria o seu custo computacional proibitivo. Só para que se tenha uma idéia, os computadores mais potentes da atualidade, capazes de executar cerca de 10^{15} operações por segundo, demorariam mais de dez anos para encontrar a política ótima de um MDP com apenas mil estados e cinco ações disponíveis. Os algoritmos de programação dinâmica permitem que a mesma operação seja executada em um computador caseiro em menos de um minuto!

A estratégia usada pelos algoritmos de programação dinâmica é explorar o espaço de políticas de uma maneira mais eficiente. Ao invés de gerar as políticas candidatas uma-a-uma, pode-se usar informação disponível sobre as políticas criadas anteriormente para criar novas políticas de uma forma mais inteligente. Todos os algoritmos de programação dinâmica têm a mesma estrutura básica, que consiste em executar alternadamente duas fases: *avaliação* e *melhoria de política*. A avaliação de política é o cálculo da função de valor V^π de uma política de decisão π. Uma das maneiras de implementar essa operação é usar a expressão (2.11), mas existem outras alternativas. A melhoria de política, por sua vez, consiste em usar a função V^π para gerar uma política $\pi' \geq \pi$. Abaixo mostro como isso pode ser feito.

Seja π uma política qualquer. Suponha que a função de valor de π tenha sido determinada, possivelmente através de (2.11). É possível com base nessa função melhorar o desempenho de π em um estado s_i isoladamente. Como discutido,$V^\pi(s_i)$ representa o valor esperado da série de recompensas coletadas por π a partir de s_i. Portanto, melhorar

o desempenho de π em s_i equivale a aumentar o valor de $V^{\pi}(s_i)$. Para tal, é necessário avaliar o efeito de se executar cada ação a em s_i e seguir π daí em diante—ou seja, fixa-se π nos estados $s_j \neq s_i$ e testa-se em s_i cada ação $a \in A$. A política π pode então ser modificada, substituindo-se $\pi(s_i)$ pela ação que resultar no maior acúmulo de recompensas. Essa operação pode ser realizada de uma maneira simples:

$$\pi(s_i) \in \operatorname*{argmax}_{a \in A} \left[r^a(s_i) + \gamma \sum_{j=1}^{|S|} P^a(s_j|s_i) V^{\pi}(s_j) \right] . \tag{2.13}$$

Note que (2.13) é viável computacionalmente graças ao conceito de função de valor, que torna acessível informação a respeito do comportamento assintótico do agente. A equação de Bellman também é fundamental, porque é ela que relaciona o valor dos estados do MDP. É possível que mais de uma ação a atinja o máximo em (2.13). Nesse caso, pode-se simplesmente escolher uma dessas ações ao acaso. O resultado da aplicação de (2.13) a uma política π é uma política π' que tem um desempenho tão bom quanto, se não melhor, do que aquele da política original—ou seja, $\pi' \geq \pi$ [128, 22].

Algoritmo

Com os conceitos apresentados até aqui já é possível definir um algoritmo de programação dinâmica que retorna uma das políticas ótimas de um MDP. Nesse algoritmo as fases de avaliação e melhoria de política seriam respectivamente as expressões (2.11) e (2.13). O algoritmo iniciaria com uma política arbitrária π_0, cuja função de valor seria calculada através de (2.11). Com base em V^{π_0}, uma nova política π_1 seria gerada pela operação (2.13)—o estado s_i envolvido nessa operação poderia ser sorteado ao acaso. A expressão (2.11) poderia então ser usada para calcular a função de valor de π_1, e assim por diante. Os dois passos acima, avaliação e melhoria de política, seriam executados alternadamente, dando origem a uma seqüências de políticas $\pi_0, \pi_1, ..., \pi_n$. Essa seqüência seria interrompida quando não houvesse como melhorar o desempenho da política atual.

Não é difícil mostrar com base em resultados da área que esse algoritmo básico convergiria para uma das políticas ótimas de um MDP finito com probabilidade um [22]. Além disso, o algoritmo visitaria o espaço de políticas de uma maneira muito mais eficiente do que a estratégia de gerar-e-testar descrita acima, pois na seqüência de políticas

geradas $\pi_t \leq \pi_{t+1}$. No entanto, esse algoritmo pode ser modificado para tornar a busca por uma política ótima ainda mais eficiente. Note que quando a avaliação de política é feita pela expressão (2.11) ela envolve a inversão de uma matriz $|S| \times |S|$, e portanto consome um número de operações aritméticas na ordem de $|S|^3$.[4] Parece um desperdício de esforço computacional investir um número de operações dessa magnitude para avaliar uma política π_{t+1} que se diferencia de π_t em apenas um estado $s_i \in S$. Pode-se, portanto, pensar em aplicar a operação (2.13) não apenas a um único estado s_i, mas a todo o espaço de estados. Essa modificação da estratégia acima dá origem ao conhecido *algoritmo de iteração de política* [64], mostrado em pseudo-código no Algoritmo 2.1.

Algoritmo 2.1 Iteração de política

Requer $\mathbf{P}^a \in \mathbb{R}^{|S|\times|S|}$, $\mathbf{r}^a \in \mathbb{R}^{|S|}$ para cada $a \in A$, $\gamma \in [0, 1)$
Retorna $\pi = \pi^*$
  $\pi' \leftarrow$ vetor aleatório em $A^{|S|}$
  repita
    $\pi \leftarrow \pi'$
    $\mathbf{v}^\pi \leftarrow (\mathbf{I} - \gamma\mathbf{P}^\pi)^{-1}\mathbf{r}^\pi$ ▷ Avaliação de política
    para $i \leftarrow 1$ **até** $|S|$ **faça**
      $\pi'_i \leftarrow \mathrm{des}\left[\mathrm{argmax}_{a\in A}\left(r_i^a + \gamma\sum_{j=1}^{|S|} p_{ij}^a v_j^\pi\right)\right]$ ▷ Melhoria de política
    fim para
  até $\pi = \pi'$

A matriz $\mathbf{P}^\pi$ e o vetor $\mathbf{r}^\pi$ que aparecem no Algoritmo 2.1 descrevem o processo de Markov induzido por π no MDP. Eles podem ser gerados de forma trivial, como descrito no final da Seção 2.1.2. Para garantir que o Algoritmo 2.1 termine em um número finito de operações, é necessário estabelecer uma estratégia consistente para escolher uma ação $a \in A$ quando mais de uma delas maximizar (2.13). Senão, é possível que o algoritmo de iteração de política entre em um ciclo em que as mesmas políticas fiquem se alternando indefinidamente. Essa estratégia de "desempate" na escolha da ação a é representada de maneira genérica no Algoritmo 2.1 pela função "des." Se for possível estabelecer uma noção de ordem entre os elementos de A, pode-se por exemplo realizar a melhoria de política escolhendo sempre o menor $a \in A$ que maximiza (2.13). Nesse caso, $\mathrm{des} \equiv \min$.

[4] Press *et al.* [127] mostram que é possível inverter uma matriz $|S| \times |S|$ em $O\left(|S|^{\log_2^7}\right)$ operações aritméticas. Como para a argumentação deste trabalho essa melhoria é insignificante, vou adotar a noção usual de que essa é uma operação $O\left(|S|^3\right)$.

O algoritmo de iteração de política ilustra de maneira clara o mecanismo subjacente em todos os algoritmos da programação dinâmica, em que as operações de avaliação e melhoria de política são sempre aplicadas de forma alternada. Como observam Sutton e Barto [161], essas operações podem ser vistas como processos conflitantes. Na avaliação de política o objetivo é encontrar uma função de valor V^π que seja consistente com uma política π. Por outro lado, na melhoria de política procura-se tornar a política π consistente com a informação disponível em V^π. Claramente, a execução de uma dessas operações afasta a outra do seu objetivo: ao se atualizar a política π, a função de valor V^π deixa de representá-la de maneira fiel. Da mesma forma, o cálculo de V^π deixa as decisões em π incoerentes com a informação disponível a respeito da coleta de recompensas.

No entanto, quando se observa essa interação do ponto de vista do problema de tomada de decisão, a avaliação e a melhoria de política estão de fato cooperando entre si para alcançar o objetivo final, que é encontrar uma das políticas ótimas do MDP. Isso é conseqüência de um fato fundamental a respeito de um processo de decisão de Markov: *as políticas ótimas são as únicas políticas de decisão que não podem ser melhoradas com base na sua função de valor*. Ou seja, a única situação em que a melhoria de política não altera uma política π é quando π já é uma das políticas ótimas do MDP. Como em todos os outros casos o resultado da melhoria de política é uma política $\pi' > \pi$, a alternância entre essa operação e a avaliação de política culmina necessariamente em uma política ótima do MDP. Uma maneira de visualizar a cooperação entre a avaliação e a melhoria de política é imaginar que os resultados dessas operações são lugares geométricos que se intersecionam em um único ponto—justamente o ponto em que $\pi = \pi^*$ e $V^\pi = V^*$. A Figura 2.2 ilustra essa idéia usando duas retas para representar o espaço associado com cada operação.

No caso do algoritmo de iteração de política, a avaliação de uma política π requer $O(|S|^3)$ operações aritméticas, como discutido acima. Essa operação é representada pelas setas pontilhadas na Figura 2.2. A melhoria de política, representada na figura pelas setas contínuas, é $O(|S|^2|A|)$. Sabe-se que a alternância dessas duas operações converge garantidamente para uma das políticas ótimas de um MDP [128, 22]. No entanto, o número de iterações necessárias para que essa convergência ocorra é ainda uma questão em aberto [92]. Mansour e Singh [93] derivaram uma cota superior de $O(2^{|S|}/|S|)$ para o nú-

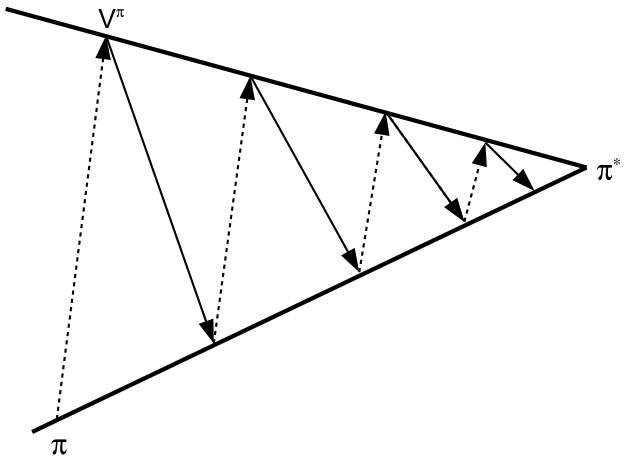


Figura 2.2: Iteração de política. As setas pontilhadas representam a avaliação de política e as setas contínuas são a melhoria de política. Esta representação do algoritmo de iteração de política é praticamente uma reprodução de uma das figuras que aparecem na Seção 4.6 do livro de Sutton e Barto [161].

mero de passos executados pela iteração de política, mas, como notam os próprios autores, esse limite pode estar muito superestimado. Na prática, a política ótima de um MDP é em geral encontrada em menos de $|S|$ iterações [93]. Adicionalmente, Puterman [128] mostra que as funções de valor V^π das políticas geradas pela iteração de política aproximam-se da função V^* a uma taxa pelo menos linear. O autor também apresenta condições em que essa convergência é quadrática.

2.2.3 Iteração de política generalizada

Como em geral $|A| \ll |S|$, o custo computacional de cada passo do algoritmo de iteração de política é dominado pelo cálculo da função de valor V^π. Esse custo pode ser reduzido drasticamente se a avaliação de política não for feita de maneira exata. Ao invés de resolver o sistema (2.10) a cada iteração, pode-se usar um método iterativo para obter uma aproximação da função V^π. Partindo de uma função V_0^π arbitrária, é possível obter uma aproximação tão próxima de V^π quanto se queira, através da aplicação sucessiva da seguinte regra de atualização:

$$V_{t+1}^\pi(s_i) \leftarrow r^{\pi(s_i)}(s_i) + \gamma \sum_{j=1}^{|S|} P^{\pi(s_i)}(s_j|s_i) V_t^\pi(s_j). \tag{2.14}$$

Note que a expressão (2.14) é simplesmente a equação de Bellman convertida em uma regra de atualização. Se o valor de todos os estados do MDP for atualizado um número infinito de vezes, a seqüência $\{V_0^\pi, V_1^\pi, ...\}$ gerada por (2.14) converge para a verdadeira função de valor V^π [128, 22].[5] A convergência da seqüência acima ocorre mesmo se os estados forem atualizados de maneira assíncrona. Isso quer dizer que o valor do estado s_i pode ser atualizado várias vezes entre duas atualizações de um outro estado s_j.

Embora a seqüência gerada por (2.14) convirja no limite para a verdadeira função V^π, só faz sentido adotar essa regra no lugar de (2.11) se a avaliação de π for interrompida após algumas iterações. Como cada aplicação de (2.14) é $O(|S|)$, a atualização de todo o espaço de estados consome $O(|S|^2)$ operações. Portanto, é possível atualizar $|S|$ vezes o valor de todos os estados $s_i \in S$ e ainda manter o tempo de processamento compatível com (2.11). No entanto, é justamente a possibilidade de alocar esse esforço de computação de forma heterogênea sobre o espaço S que torna (2.14) uma alternativa particularmente atraente [161]. Por exemplo, ao invés de atualizar o valor dos estados um-a-um, pode-se selecionar um subconjunto de S que seja considerado importante e atualizar o valor dos seus estados mais freqüentemente do que o dos demais. É possível inclusive adotar um método adaptativo que selecione os estados a serem atualizados em tempo de execução [105, 117]. Se um método eficiente for utilizado na seleção dos estados, pode-se interromper a avaliação de uma política após algumas poucas aplicações de (2.14). Isso significa uma grande economia de tempo de computação em relação à execução de (2.11).

Quando a avaliação de política é realizada por (2.11), a função V^π resultante é exata. Isso significa que cada aplicação de (2.13) a uma política π resulta necessariamente em uma política de melhor qualidade. Como o número de operações aritméticas envolvidas em cada execução de (2.11) é fixo, faz sentido realizar a melhoria de política em todo o espaço de estados antes de cada avaliação. Por outro lado, quando a avaliação de política é feita iterativamente através de (2.14), a função de valor resultante é apenas uma aproximação de V^π. Nesse caso, a operação (2.13) deixa de ser exata. Além disso, o

[5] Como aqui o espaço de estados é finito, dizer que a seqüência $\{V_0^\pi, V_1^\pi, ...\}$ converge para V^π equivale a escrever

$$\lim_{t\to\infty} \parallel \mathbf{v}_t^\pi - \mathbf{v}^\pi \parallel \longrightarrow 0,$$

onde $\parallel \cdot \parallel$ é uma norma definida em $\mathbb{R}^{|S|}$[80].

número de operações executadas na avaliação subseqüente varia conforme as características da política sendo avaliada. Portanto, pode ser interessante usar (2.13) para atualizar π em apenas um subconjunto de S. Assim como no caso anterior, a seleção dos estados em que π será modificada pode ser feita segundo vários critérios, inclusive levando em consideração informação disponível apenas em tempo de execução.

As idéias apresentadas nesta seção permitem a definição de um algoritmo muito flexível chamado de *iteração de política generalizada* [22, 161]. A iteração de política generalizada é descrita passo-a-passo no Algoritmo 2.2. A diferença desse algoritmo em relação à iteração de política original é que as fases de avaliação e melhoria de política podem ser feitas de forma aproximada. Se assintoticamente todos os estados $s_i \in S$ forem submetidos um número infinito de vezes às operações (2.14) e (2.13), a iteração de política generalizada converge garantidamente para uma política ótima do MDP [22].

Algoritmo 2.2 Iteração de política generalizada

Requer $\mathbf{P}^a \in \mathbb{R}^{|S|\times|S|}$, $\mathbf{r}^a \in \mathbb{R}^{|S|}$ para cada $a \in A$, $\gamma \in [0, 1)$, $n_v \in \mathbb{N}_*^+$
Retorna $\pi \approx \pi^*$
  $\pi \leftarrow$ vetor aleatório em $A^{|S|}$
  $\mathbf{v}^\pi \leftarrow (\mathbf{I} - \gamma\mathbf{P}^\pi)^{-1}\mathbf{r}^\pi$
  repita
    $S^v \leftarrow$ subconjunto de $\{1, 2, ..., |S|\}$
    para $k \leftarrow 1$ **até** n **faça**
      para $i \in S^v$ **faça** $v_i^\pi \leftarrow r_i^\pi + \gamma \sum_{j=1}^{|S|} p_{ij}^\pi v_j^\pi$ ▷ Avaliação de política
    fim para
    $S^\pi \leftarrow$ subconjunto de $\{1, 2, ..., |S|\}$
    para $i \in S^\pi$ **faça** $\pi_i \leftarrow \text{des}\left[\text{argmax}_{a\in A}\left(r_i^a + \gamma \sum_{j=1}^{|S|} p_{ij}^a v_j^\pi\right)\right]$ ▷ Melhoria de política
  até critério de parada satisfeito

Observe que o Algoritmo 2.2 requer um parâmetro a mais do que o Algoritmo 2.1. O parâmetro n_v indica o número de vezes em que a regra (2.14) será aplicada a cada estado selecionado para a avaliação. Esse parâmetro pode ser definido em tempo de execução, de acordo com o andamento do processo de avaliação de uma política. Note também que o critério de parada para a iteração de política generalizada foi colocado de maneira genérica. Isso porque essa definição depende das estratégias usadas para a seleção dos estados onde a avaliação e a melhoria de política serão realizadas. Por exemplo, se $S^\pi =$

$\varnothing$ em uma iteração, o critério de parada usado no Algoritmo 2.1 pode falhar, porque o fato de $\pi = \pi'$ não implica na otimalidade dessa política de decisão.

A iteração de política generalizada confere grande flexibilidade às etapas de avaliação e melhoria de política. É natural perguntar se existe uma maneira ótima de realizar essas duas operações. Obviamente, quanto mais acuradas são a avaliação e a melhoria de política, maior o custo computacional de cada iteração do algoritmo. Em compensação, isso também tende a diminuir o número total de iterações. Uma maneira de visualizar essa questão é comparar a Figura 2.2 com a Figura 2.3 abaixo. No primeiro caso tanto a avaliação quanto a melhoria de política são realizadas de maneira exata. Isso reduz o número total de iterações, mas aumenta o número de operações aritméticas realizadas em cada uma delas. A Figura 2.3 ilustra o caso em que a avaliação e a melhoria de política são realizadas em apenas um subconjunto de S a cada iteração. O resultado, como era de se esperar, é um número maior de iterações mais curtas. É razoável supor que exista um compromisso entre comprimento e número de iterações que minimize o número total de operações aritméticas realizadas pelo algoritmo. Embora isso seja verdade para um MDP específico, não parece haver uma resposta para essa questão que se aplique a todos os casos. Em outras palavras, o algoritmo que minimiza o número total de operações na busca por uma política ótima depende das características do processo de decisão de Markov [128, 22].

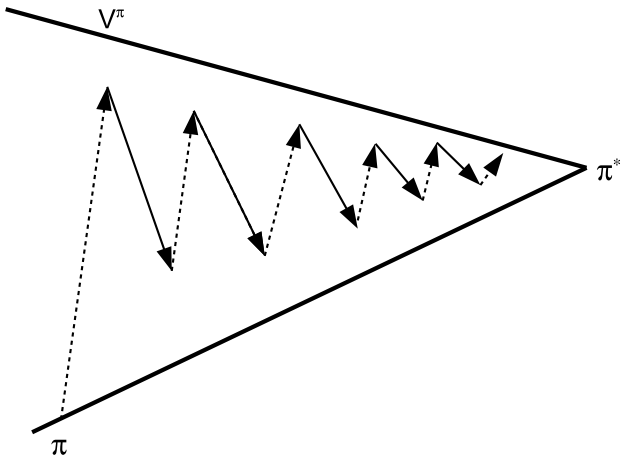


Figura 2.3: Iteração de política generalizada.

2.2.4 Iteração de valor

Suponha que na iteração de política generalizada a avaliação e a melhoria de política sejam sempre realizadas em todos os estados do MDP, ou seja, suponha que no Algoritmo 2.2 $S^v = S^\pi = S$ em todas as iterações. Nesse caso, fazendo $n_v = \infty$ tem-se de volta a iteração de política convencional. O outro extremo, quando $n_v = 1$, também corresponde a um conhecido algoritmo de programação dinâmica, chamado de *iteração de valor* [13]. A iteração de valor é mostrada em pseudo-código no Algoritmo 2.3.

Algoritmo 2.3 Iteração de valor

Requer $\mathbf{P}^a \in \mathbb{R}^{|S|\times|S|}$, $\mathbf{r}^a \in \mathbb{R}^{|S|}$ para cada $a \in A$, $\gamma \in [0,1)$, $\varepsilon \in \mathbb{R}_*^+$
Retorna $\mathbf{v} \approx \mathbf{v}^*$
  $\mathbf{v}'\leftarrow$ vetor aleatório em $\mathbb{R}^{|S|}$
  repita
    $\mathbf{v}\leftarrow\mathbf{v}'$
    para $i\leftarrow 1$ **até** $|S|$ **faça** $v_i'\leftarrow \max_{a\in A}\left(r_i^a + \gamma\sum_{j=1}^{|S|} p_{ij}^a v_j\right)$
  até $\| \mathbf{v}' - \mathbf{v} \| < \varepsilon$

Quando as operações (2.14) e (2.13) são aplicadas uma única vez de forma alternada, elas podem ser combinadas para formar uma única regra de atualização. Nesse caso, ao invés de realizar a melhoria de política de maneira explícita, basta aplicar (2.14) ao que seria o resultado da operação (2.13). Simbolicamente, isso significa que

$$V(s_i)\leftarrow r_i^{a^*} + \gamma\sum_{j=1}^{|S|} p_{ij}^{a^*} V(s_j), \quad \text{com} \quad a^* \in \operatorname{argmax}_{a\in A}\left[r_i^a + \gamma\sum_{j=1}^{|S|} p_{ij}^a V(s_j)\right]$$

equivale a

$$V(s_i)\leftarrow \max_{a\in A}\left[r_i^a + \gamma\sum_{j=1}^{|S|} p_{ij}^a V(s_j)\right].$$

A aplicação da regra de atualização acima a todos os estados de um MDP envolve $O(|S|^2|A|)$ operações. Algumas vezes é conveniente representar essa operação como um mapeamento definido como

$$\begin{aligned} \Upsilon:\ & \mathbb{R}^{|S|} \mapsto \mathbb{R}^{|S|} \\ & \Upsilon\mathbf{v} = \max_{a\in A}\left(\mathbf{r}^a + \gamma\mathbf{P}^a\mathbf{v}\right), \end{aligned} \tag{2.15}$$

em que a operação "max" deve ser aplicada elemento-por-elemento. Normalmente o mapeamento Υ é referenciado como o *operador da programação dinâmica*. Pode-se

mostrar que Υ é uma contração em $\mathbb{R}^{|S|}$ cujo ponto fixo é a função $\mathbf{v}^*$, o que implica

$$\lim_{n\to\infty} \Upsilon^n \mathbf{v} \to \mathbf{v}^*,$$

qualquer que seja $\mathbf{v}$ [80, 22]. Ou seja, partindo de qualquer vetor $\mathbf{v} \in \mathbb{R}^{|S|}$, a aplicação sucessiva do operador Υ origina uma seqüência que converge eventualmente para a função de valor ótima do MDP. Note que o resultado é válido mesmo se o vetor $\mathbf{v}$ não representar a função de valor de uma política π.

A partir da discussão acima, fica claro que a seqüência $\{\mathbf{v}_1, \mathbf{v}_2, ...\}$ gerada pelo algoritmo de iteração de valor converge assintoticamente para $\mathbf{v}^*$ (não custa lembrar que a partir de $\mathbf{v}^*$ é possível determinar todas as políticas ótimas de um MDP usando (2.13)). Evidentemente, em uma implementação real não se pode esperar um número infinito de iterações, e por isso é conveniente definir uma vizinhança em torno de $\mathbf{v}^*$ dentro da qual uma solução $\mathbf{v}$ é satisfatória. William e Baird [186] mostram que se o critério de parada do Algoritmo 2.3 for definido como

$$\| \mathbf{v} - \mathbf{v}' \|_\infty < \frac{(1-\gamma)}{2\gamma}\varepsilon, \tag{2.16}$$

onde $\| \cdot \|_\infty$ é a norma do máximo, a política π retornada pela iteração de valor é ε-ótima, ou seja,

$$\| \mathbf{v} - \mathbf{v}^* \|_\infty < \varepsilon.$$

Isso significa que ao seguir π ao invés de π^* o agente deixa de coletar, no máximo, ε unidades de recompensa, independentemente do estado onde ele iniciou a sua interação com o ambiente. No entanto, sabe-se que se $\mathbf{v}$ estiver suficientemente próxima de $\mathbf{v}^*$, a política derivada dessa função é ótima [20, 22]. Na prática, portanto, basta selecionar ε suficientemente pequeno para garantir que a política retornada pela iteração de valor colete o máximo possível de recompensas no MDP. Como a região em torno de $\mathbf{v}^*$ que origina políticas ótimas varia de caso para caso, pode-se combinar o Algoritmo 2.3 com técnicas de identificação de ações sub-ótimas para garantir a otimalidade da solução encontrada [128].

chamada de *síncrona*, porque o valor de todos os estados é atualizado com base em uma função de valor fixa. Em uma implementação em computador, isso significa que dois vetores $\mathbf{v}$ e $\mathbf{v}'$ são utilizados para representar duas versões sucessivas da função de valor.

É possível usar apenas um vetor $\mathbf{v}$, o que significa que valores recém-calculados estariam imediatamente disponíveis para atualizações posteriores. Esta seria a versão *assíncrona* do algoritmo [21]. Uma maneira de entender a diferença entre as versões síncrona e assíncrona da iteração de valor é comparar os métodos de Jacobi e Gauss-Seidel para a solução de sistemas lineares [127]. Assim como a iteração de valor original, a iteração de valor assíncrona converge para a função de valor ótima do MDP [22]. De fato, existe alguma evidência de que a atualização assíncrona dos estados acelera a convergência do algoritmo [161].

2.2.5 O espaço da programação dinâmica

A iteração de política generalizada é um algoritmo genérico que tem a iteração de valor e a iteração de política como casos particulares. Embora estes últimos sejam sem dúvida os algoritmos mais conhecidos da programação dinâmica, eles representam apenas duas das inúmeras configurações possíveis do Algoritmo 2.2. Uma maneira de visualizar essa questão é imaginar a iteração de política generalizada definindo um *espaço de algoritmos*, como mostra a Figura 2.4. Cada algoritmo é caracterizado por três variáveis: o número de estados envolvidos na melhoria de política, o número de estados envolvidos na avaliação de política e o número n_v de atualizações executadas nesta última. Note que o caso em que $n_v = \infty$ pode ser interpretado como a substituição de (2.14) por (2.11). Desde que todos os estados sejam selecionados um número infinito de vezes para a atualização, os algoritmos pertencentes ao espaço mostrado na figura convergem assintoticamente para a função de valor ótima V^* [22].

Como mostra a Figura 2.4, a iteração de política corresponde ao caso em que tanto a função V^π quanto a política π são atualizadas em todo o espaço de estados a cada iteração. Além disso, a atualização de V^π é feita de maneira exata ($n_v = \infty$). Se os estados forem atualizados com apenas uma aplicação de (2.14), ou seja, se $n_v = 1$, a iteração de política se transforma na iteração de valor síncrona. A versão assíncrona desse algoritmo corresponde ao caso em que apenas um estado do espaço S é atualizado a cada iteração. Nesse caso, as operações (2.14) e (2.13) são aplicadas de forma alternada a cada um dos estados $s_i \in S$. Pode-se pensar em outros algoritmos que representam configurações intermediá-

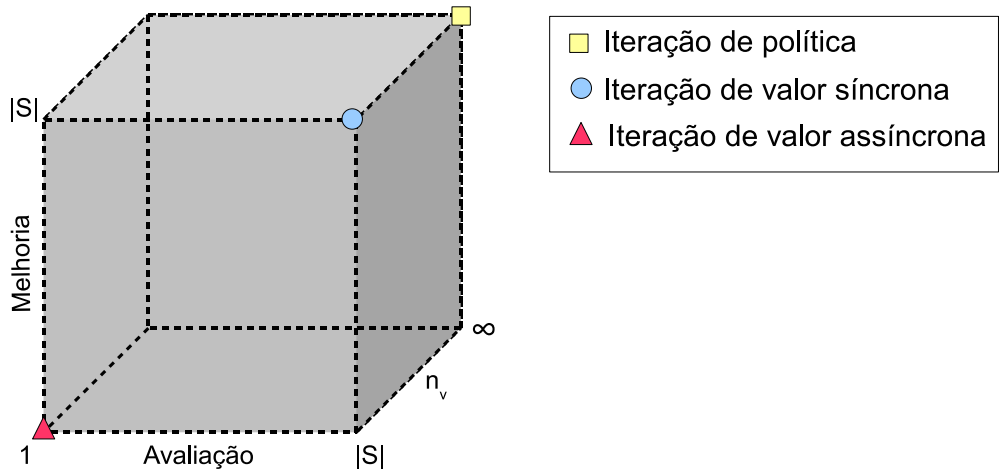


Figura 2.4: Espaço dos algoritmos de programação dinâmica. Como as três variáveis são discretas, pode-se imaginar esse espaço como uma malha tridimensional em que cada ponto representa um algoritmo. Contanto que as restrições teóricas sejam respeitadas, todos os algoritmos retornam a função de valor ótima de um MDP.

rias entre esses casos extremos. É possível, por exemplo, atualizar a política π em apenas um subconjunto de S e aplicar (2.14) prioritariamente aos estados mais afetados com essa mudança [105, 117]. Embora o número de estados envolvidos na avaliação de política não tenha que coincidir com a quantidade de estados em que π é atualizada, configurações em que existe uma grande discrepância entre esses dois valores fazem menos sentido (um exemplo de um algoritmo desse tipo foi dado no início da Seção 2.2.2).

Note que o critério usado para distinguir os algoritmos na Figura 2.4 é insensível a algumas de suas características. Por exemplo, na iteração de valor assíncrona os estados são submetidos um-a-um às operações (2.14) e (2.13). Pode-se pensar em uma versão desse algoritmo em que o estado a ser atualizado é *amostrado* de uma distribuição definida em S (contanto que todos os estados tenham uma probabilidade não-nula de serem selecionados, a convergência assintótica do algoritmo não é afetada). Na Figura 2.4 ambas as versões da iteração de valor seriam identificadas como o mesmo algoritmo. Além disso, é totalmente plausível pensar em mudar o número de estados atualizados em cada iteração de um determinado algoritmo, de acordo com o andamento do processo de aprendizagem. Nesse caso, cada *iteração* do algoritmo em questão seria representado por um ponto na Figura 2.4.

Técnicas avançadas

Todos os algoritmos discutidos até aqui podem ser considerados como "métodos padrões" para a solução de um MDP. No caso de aplicações reais, é aconselhável combiná-los com técnicas mais avançadas para diminuir o custo computacional por iteração ou acelerar a convergência. Por exemplo, Puterman [128] mostra uma versão modificada da iteração de valor que apresenta um desempenho significativamente superior ao da versão original do algoritmo. Na *iteração de valor relativa*, o vetor $\mathbf{v}$ é normalizado a cada iteração, da seguinte forma:

$$\mathbf{u} = \mathbf{v} - \bar{\mathbf{v}},$$

em que $\bar{v}_i = \frac{1}{|S|} \sum_{j=1}^{|S|} v_j$, para todo i. Além disso, o critério de parada (2.16) é substituído por

$$\mathrm{sp}(\mathbf{u} - \mathbf{u}') < \frac{(1-\gamma)}{\gamma}\varepsilon,$$

onde

$$\mathrm{sp}(\mathbf{u}) = \max_i u_i - \min_i u_i.$$

Puterman mostra que com essas modificações simples o Algoritmo 2.3 deixa de executar uma série de iterações desnecessárias, o que resulta em uma convergência mais rápida para uma política ε-ótima. Dependendo das características do MDP, a redução no número de iterações executadas pelo algoritmo pode ser dramática [128].

Outras técnicas podem ser utilizadas para melhorar o desempenho não somente da iteração de valor, como também dos demais algoritmos mostrados na Figura 2.4. Por exemplo, existem métodos que permitem a identificação e eliminação de ações sub-ótimas de um problema, o que diminui consideravelmente o custo computacional da busca por uma política ótima de um MDP [128]. Uma outra maneira de reduzir o número de operações executadas na solução de um MDP é durante a avaliação de política agregar dinamicamente os estados do problema com valores parecidos [18]. Finalmente, pode-se utilizar implementações paralelas dos algoritmos discutidos neste capítulo, o que reduz drasticamente o seu tempo de execução em ambientes com vários processadores [19, 21].

2.2.6 Um parágrafo de história

A origem da programação dinâmica é em geral associada com o trabalho de Bellman, que foi quem cunhou o nome da disciplina [13, 14, 15]. No entanto, Puterman [128] afirma que a programação dinâmica tem raízes mais antigas, com destaque para os trabalhos de Wald [171] e Massé (descrito em inglês por Gessford e Karlin [48]). Um marco para a área foi a publicação do livro de Howard [64], o primeiro autor a estudar em detalhes a formulação do problema com recompensas descontadas. Howard também propôs o algoritmo de iteração de política de forma independente de Bellman, que o chamava de "aproximação no espaço das políticas" [13]. Alguns anos mais tarde Puterman e Shin [129] apresentaram uma versão desse algoritmo chamada "iteração de política modificada," que está estreitamente relacionada com o algoritmo discutido na Seção 2.2.3.[6] Atualmente a quantidade de livros disponíveis sobre a programação dinâmica dificulta uma listagem exaustiva dos títulos. Dentre eles, os livros de Ross [133], Bertsekas [20] e Puterman [128] são freqüentemente citados como referências importantes da área. Puterman apresenta a programação dinâmica de uma maneira bastante genérica e discute algumas aplicações interessantes. De acordo com ele, muitas outras aplicações podem ser encontradas no trabalho de White [177, 178]. Bertsekas e Tsitsiklis [22] também apresentam uma introdução ao assunto, além de resultados teóricos importantes da área—embora a ênfase do seu livro seja o uso de aproximadores na programação dinâmica.

2.3 Aprendizagem por Reforço

Em poucas palavras, *aprendizagem por reforço* é a programação dinâmica sem um modelo. Ou seja, a aprendizagem por reforço engloba todas as técnicas desenvolvidas para encontrar a política ótima de um MDP quando não se conhece as funções P^a e r^a.

A primeira idéia que vem à cabeça quando não se tem um modelo do problema de tomada de decisão é construir um. No caso da aprendizagem por reforço, a construção de um MDP pode ser feita a partir de um conjunto de transições (s_i, a, r, s_j), onde s_j é um dos resultados possíveis da execução de a em s_i, e r é a recompensa associada à transição

[6] Bertsekas e Tsitsilis [22] chamam esse algoritmo de "iteração de política assíncrona."

$s_i \xrightarrow{a} s_j$. Suponha que exista uma amostra de transições S^a para cada ação $a \in A$, isso é, o conjunto S^a contém todas as transições em que a ação a foi executada. Com base nessa informação, pode-se facilmente construir uma aproximação da função de transição P^a. Seja $S_i^a \subset S^a$ o subconjunto de S^a formado por todas as transições iniciadas em s_i, e seja $S_{ij}^a \subset S_i^a$ o subconjunto deste último composto pelas transições terminadas em s_j. Nesse caso, a aproximação da função de transição seria dada por:

$$\tilde{P}^a(s_j|s_i) = \frac{|S_{ij}^a|}{|S_i^a|}. \tag{2.17}$$

Se os estados $s_j \in S_{ij}^a$ forem amostrados segundo $P(\cdot|s_i)$, não é difícil perceber que $\tilde{P}^a(s_j|s_i) \to P(s_j|s_i)$ quando $|S_i^a| \to \infty$ [85]. Raciocínio análogo vale para a aproximação das funções de recompensa r^a. Conclusão: pode-se construir uma aproximação de um MDP tão precisa quanto se queira, bastando para isso que cada ação $a \in A$ seja tentada um número suficientemente grande de vezes em todos os estados $s_i \in S$.

A abordagem acima é uma alternativa viável para a solução de problemas de tomada de decisão de pequeno porte. No entanto, à medida que o número de estados e ações do problema começa a crescer, a sua aplicação começa a ficar problemática. Imagine o caso do jogo de xadrez, que vem sendo usado como exemplo neste capítulo. Suponha que o objetivo fosse desenvolver um programa de computador para derrotar um jogador específico—o russo Kasparov, por exemplo. Nesse caso, cada estado do modelo corresponderia a uma configuração diferente do tabuleiro e as ações seriam os movimentos permitidos em cada posição. As probabilidades de transição entre os estados seriam derivadas da estratégia de jogo de Kasparov. Uma possibilidade para modelar essa estratégia seria apresentar ao jogador todas as configurações possíveis do tabuleiro e perguntar-lhe qual seria a jogada escolhida em cada situação. A partir dessa informação seria relativamente fácil determinar a dinâmica do MDP resultante usando a abordagem descrita acima. Note, no entanto, que embora $|S|$ e $|A|$ sejam números finitos nesse exemplo, eles são grandes demais para permitir a construção de um MDP dessa maneira. Para que se tenha uma idéia, em seu artigo clássico Shannon [146] estima que o número de configurações possíveis do tabuleiro de xadrez gira em torno de 10^{43}. Mesmo que Kasparov levasse apenas um segundo para avaliar cada posição, ainda assim seriam necessários 3.17×10^{35} anos para que todos os estados fossem avaliados.

Mas seria essa a melhor alternativa para se construir um MDP? No jogo de xadrez alguns estados são claramente mais importantes do que outros, não apenas por representarem situações mais críticas, como também por ocorrerem mais freqüentemente. Um exemplo extremo é a configuração inicial do tabuleiro, que ocorre necessariamente em *todos* os jogos. Da mesma forma, existem configurações do tabuleiro que, embora válidas, representam situações tão improváveis que ignorá-las completamente não traria maiores prejuízos. Seria natural, portanto, focar a construção do modelo nos estados que têm uma grande probabilidade de ocorrer em um jogo contra Kasparov. Note, no entanto, que a identificação precisa de tais estados depende do conhecimento da estratégia usada pelo jogador—exatamente aquilo que se está tentando modelar!

O cenário acima ilustra o paradoxo fundamental enfrentado pela aprendizagem por reforço: em geral deseja-se obter informação a respeito de um problema de forma inteligente, mas isso depende justamente da informação que se pretende obter. A solução é tornar a coleta de informação parte do problema.

2.3.1 O dilema entre exploração e perscrutação

Suponha que o objetivo seja calcular a função de valor de uma política π. Como discutido na Seção 2.2.3, uma das maneiras de fazê-lo é aplicar sucessivamente a regra de atualização (2.14) a todos os estados do espaço de estados. Também mencionado naquela seção é o fato de as atualizações não precisarem ser feitas de maneira síncrona; desde que todos os estados sejam atualizados um número infinito de vezes, o processo converge para a verdadeira função de valor V^π. Uma conseqüência disso é que, ao invés de atualizar os estados do MDP um-a-um, pode-se amostrá-los de uma distribuição de probabilidades definida sobre o espaço S. A única restrição em relação à distribuição é que todos os estados tenham uma probabilidade não-nula de serem selecionados.

A escolha mais óbvia seria amostrar os estados de uma distribuição uniforme em S. No entanto, essa abordagem não oferece nenhum benefício aparente em relação à estratégia de simplesmente "varrer" todo o espaço de estados a cada avaliação de política. Uma possibilidade mais atraente seria atribuir a cada estado s_i uma probabilidade de

ser selecionado proporcional à sua importância para o processo de decisão. No entanto, como discutido acima, a importância de cada estado não pode em geral ser determinada de antemão. Uma terceira possibilidade é amostrar os estados de acordo com uma *política de exploração*.

Uma política de exploração π_e define uma distribuição de probabilidades sobre os estados de S. Formalmente essa distribuição corresponde à *distribuição estacionária* da cadeia de Markov induzida por π_e no MDP [22], mas aqui basta saber o seguinte: se a política de exploração for tal que todos os estados possam ser alcançados pelo agente, então π_e pode ser usada para selecionar os estados que terão os seus valores atualizados. Basta iniciar o agente em um estado qualquer e aplicar (2.14) a cada transição executada por ele ao seguir π_e na sua interação com o MDP. Uma vantagem óbvia dessa estratégia é que a aprendizagem pode ocorrer em tempo real, ou seja, *enquanto o agente interage com o ambiente*. Um exemplo de como isso pode ser feito é dado no Algoritmo 2.4. Obviamente, esse algoritmo pode ser modificado para contemplar o caso em que a aprendizagem deve ocorrer *offline*: basta guardar a trajetória percorrida pelo agente e posteriormente aplicar (2.14) à seqüência de estados que compõem essa trajetória. É trivial também adequar o algoritmo a tarefas não-episódicas.

Algoritmo 2.4 Avaliação iterativa de política

Requer $\mathbf{P}^a \in \mathbb{R}^{|S|\times|S|}$, $\mathbf{r}^a \in \mathbb{R}^{|S|}$ para cada $a \in A$, $\gamma \in [0, 1)$, π, $\pi_e \in A^{|S|}$
Retorna $\mathbf{v}^\pi$
  $\mathbf{v}^\pi \leftarrow$ vetor aleatório em $\mathbb{R}^{|S|}$
  repita
    $i \leftarrow$ número aleatório em $\{1, 2, ..., |S|\}$
    repita
      $v_i^\pi \leftarrow r_i^\pi + \gamma \sum_{j=1}^{|S|} p_{ij}^\pi v_j^\pi$
      transição $s_i \xrightarrow{\pi_i^e} s_j$
      $i \leftarrow j$
    até s_i é um estado terminal
  até critério de parada satisfeito

Observe a existência de duas política distintas no Algoritmo 2.4: a política π, cuja função de valor pretende-se calcular, e π_e, que é a política de exploração. Embora o agente siga esta última, as atualizações (2.14) são feitas de acordo com a primeira. Uma pergunta

que surge naturalmente neste contexto é por que não fazer simplesmente $\pi = \pi_e$. De fato, isso pode ser feito, contanto que π resulte em uma probabilidade não-nula de o agente alcançar todos os estados do MDP. Se alguns estados forem ignorados, a função de valor retornada pela avaliação iterativa de política não refletirá o verdadeiro desempenho de π— o que pode distorcer a política que seria derivada de V^π através de (2.13). Como em geral a política π não apresenta as características necessárias a uma política de exploração, costuma-se definir esta última como uma versão ligeiramente modificada da primeira. Uma das estratégias mais simples é selecionar as ações de acordo com π na maioria das vezes, mas a cada transição escolher com probabilidade ω uma ação ao acaso [161, 71].

Note que se a política de exploração for baseada em π, a estratégia de amostragem usada no Algoritmo 2.4 pode ser vista como uma maneira indireta de selecionar os estados segundo a sua importância para o problema de tomada de decisão. Basta lembrar que o Algoritmo 2.4 está inserido no contexto de iteração de política generalizada (Algoritmo 2.2). Mais especificamente, esse algoritmo implementa a avaliação de política, que será seguida da fase de melhoria de política. Como a política π está sendo sucessivamente refinada, é razoável esperar que com o passar do tempo essa política passe a visitar apenas as áreas mais importantes do espaço de estados. Uma maneira de visualizar essa evolução é imaginar um aprendiz de xadrez, que aos poucos deixa de fazer jogadas improváveis e passa a encontrar apenas as configurações do tabuleiro com que normalmente se deparam jogadores razoáveis.

Observe, no entanto, que para melhorar o seu jogo o aprendiz precisa "arriscar" de vez em quando. No caso da aprendizagem por reforço, isso é representado pelo parâmetro de exploração ω. Quanto menor esse parâmetro, mais próxima é a política π_e de π. Por um lado, isso concentra a experiência do agente nas regiões mais importantes do espaço S, pelos motivos discutidos acima. No entanto, isso também impede que o agente descubra informações novas a respeito do problema, informações essas essenciais para o processo posterior de melhoria de política. Este é o famoso *dilema entre exploração e perscrutação* da aprendizagem por reforço. Infelizmente, ainda não há uma solução definitiva para esse dilema. Embora existam métodos de exploração bem fundamentados teoricamente que se aplicam a cenários específicos, a postura mais comum é usar heurísticas simples como a descrita acima [167, 70]

2.3.2 Diferenças temporais

O Algoritmo 2.4 permite que a avaliação de uma política seja feita *on-line*, ou seja, enquanto o agente interage com o ambiente seguindo uma política de exploração. No entanto, esse algoritmo pressupõe o conhecimento das funções P^a e r^a, o que viola a premissa básica da aprendizagem por reforço. Felizmente, uma análise mais cuidadosa revela que essas funções não são indispensáveis. Basta lembrar que $P^\pi(\cdot|s_i)$ é uma distribuição de probabilidades. Assim, ao invés de atualizar o valor de s_i com base em todas as transições possíveis a partir desse estado, pode-se fazê-lo considerando uma transição específica amostrada de $P^\pi(\cdot|s_i)$. Do ponto de vista prático, isso significa que a transição $s_i \xrightarrow{\pi(s_i)} s_j$ resulta na seguinte atualização:

$$V^\pi(s_i) \leftarrow V^\pi(s_i) + \alpha \left[r + \gamma V^\pi(s_j) - V^\pi(s_i) \right], \tag{2.18}$$

onde $\alpha \in [0, 1]$ é a *taxa de aprendizagem*. A atualização (2.18) pode ser vista como a versão incremental da regra (2.14). Uma maneira de entendê-la é imaginar que a cada atualização o valor de $V^\pi(s_i)$ é movido ligeiramente no sentido de $r + \gamma V^\pi(s_j)$. Como r e s_j são amostrados de acordo com π, as atualizações convergem para $E^\pi\{r + \gamma V^\pi(s_j)\}$, que é justamente o lado direito da regra (2.14). Note que o valor-alvo das atualizações só é conhecido um instante de tempo após a execução da ação $\pi(s_i)$. Por isso, convencionou-se chamar (2.18) de método das *diferenças temporais* (TD[†]) [157].

O método TD é considerado uma das maiores contribuições da comunidade de inteligência artificial ao estudo de problemas de tomada de decisão seqüencial. Ele permite que a aprendizagem ocorra em tempo real *sem a presença de um modelo*. Se todos os estados do MDP tiverem o seu valor atualizado um número infinito de vezes pela regra (2.18), o processo converge eventualmente para a verdadeira função de valor de π [157].

Como cada aplicação de (2.18) resulta em uma modificação pequena da função V^π, em alguns casos a convergência do método TD pode ser muito lenta. Uma maneira de aliviar esse problema é usar uma *atualização com rastro*[†]. A motivação da atualização com rastro vem da seguinte constatação: quando o estado s_i tem o seu valor modificado, o valor de todos os estados que levam a s_i também é alterado. Assim, ao invés de restringir

[†] *Temporal differences*

[†] *Eligibility traces*

a aplicação de (2.18) ao penúltimo estado visitado pelo agente, pode-se estendê-la a toda a trajetória executada até então. Evidentemente, a modificação deve ser mais intensa nos estados visitados mais recentemente. Uma analogia que pode ser esclarecedora é o processo usado pelas formigas para encontrar o caminho mais curto até uma fonte de comida, já explorada pela comunidade científica para resolver problemas de otimização [43]. Apenas para estimular a intuição, imagine que ao passar por um estado o agente libere uma substância que imediatamente começa a evaporar—o análogo ao *feromônio* liberado pelas formigas. Nesse caso, a quantidade de "feromônio" em cada estado determinaria a intensidade da modificação realizada por (2.18). Estados visitados recentemente teriam o seu valor modificado de forma mais intensa, porque o feromônio ainda não teria tido tempo de evaporar. Com o passar do tempo, no entanto, o processo de evaporação tornaria o efeito de (2.18) desprezível, a não ser que o agente voltasse a visitar o estado em questão.

Do ponto de vista prático, o feromônio pode ser representado por uma variável e_i associada a cada estado s_i do MDP. A cada instante de tempo os valores de todas as variáveis e_i sofrem um decréscimo, cuja magnitude é definida pelo parâmetro λ. Na analogia acima esse parâmetro seria a "taxa de evaporação" do rastro deixado pelo agente. A variável correspondente ao estado ocupado pelo agente tem uma unidade acrescida ao seu valor—o que pode ser interpretado como um novo depósito de feromônio. O Algoritmo 2.5 mostra como esses conceitos podem ser organizados para formar um método de avaliação de política conhecido como TD(λ).

Pode-se mostrar que o algoritmo TD(λ) converge para a função de valor V^π sob as mesmas condições do Algoritmo 2.4. Se for usada uma taxa de aprendizagem fixa, a convergência ocorre na média [157], mas se esse parâmetro decrescer a uma taxa adequada a convergência ocorre com probabilidade um [38, 39, 68, 168]. Note que com o algoritmo TD(λ) não faz muito sentido usar uma política de exploração, porque isso exigiria o conhecimento das funções P^a e r^a (além de dificultar a atualização com rastro). Em alguns casos é aconselhável calcular a função de valor de uma versão ligeiramente modificada de π que visite todos os estados do MDP. Contanto que o parâmetro de exploração ω seja suficientemente pequeno, essa alteração não traz maiores conseqüências para o processo de iteração de política generalizada [161].

Algoritmo 2.5 TD(λ)

Requer $\pi \in A^{|S|}$, $\gamma \in [0,1)$, $\lambda, \alpha \in [0,1]$
Retorna $\mathbf{v}^\pi$
  $\mathbf{v}^\pi \leftarrow$ vetor aleatório em $\mathbb{R}^{|S|}$
  $\mathbf{e} \leftarrow 0 \in \mathbb{R}^{|S|}$
  repita
    $i \leftarrow$ número aleatório em $\{1, 2, ..., |S|\}$
    repita
      execute a transição $s_i \xrightarrow{\pi_i} s_j$ e guarde a recompensa recebida r
      $\delta \leftarrow r + \gamma v_j^\pi - v_i^\pi$
      $e_i \leftarrow e_i + 1$
      para $k \leftarrow 1$ **até** $|S|$ **faça**
        $v_k^\pi \leftarrow v_k^\pi + \alpha \delta e_k$
        $e_k \leftarrow \gamma \lambda e_k$
      fim para
      $i \leftarrow j$
    até s_i é um estado terminal
  até critério de parada satisfeito

Se o agente visitar várias vezes o estado s_i em um pequeno intervalo de tempo, pode ocorrer de a variável e_i crescer demasiadamente, potencialmente prejudicando o processo de aprendizagem. Uma maneira de contornar esse problema é fazer $e_i = 1$ quando o agente passar por s_i [151]. Na analogia usada acima, isso corresponderia a limitar a quantidade de feromônio em cada estado. Embora seja extremamente simples, essa estratégia pode em alguns casos acelerar de forma significativa o cálculo da função de valor [161].

2.3.3 O valor de uma ação

O algoritmo TD(λ) permite que a avaliação de política seja feita sem um modelo do MDP, mas a busca por uma política ótima ainda depende de uma descrição explícita do processo de decisão de Markov. Isso porque a etapa de melhoria de política definida por (2.13) exige o conhecimento da família de funções P^a e r^a. Um conceito fundamental da aprendizagem por reforço que liberta a melhoria de política da necessidade de um modelo é o de uma *função de valor de ação*. A função $Q^\pi(s,a)$ representa o valor esperado das recompensas recebidas ao se executar a ação a no estado s_i e seguir π daí em

diante. Talvez a maneira mais fácil de entender essa função seja verificar como ela se enquadra na equação de Bellman de uma política π:

$$\begin{aligned} Q^{\pi}(s_i, a) &= r^a(s_i) + \gamma \sum_{j=1}^{|S|} P^a(s_j|s_i) V^{\pi}(s_j) \\ &= r^a(s_i) + \gamma \sum_{j=1}^{|S|} P^a(s_j|s_i) Q^{\pi}(s_j, \pi(s_j)). \end{aligned} \tag{2.19}$$

Observando (2.19) e (2.13), fica claro que o processo de melhoria de política pode ser reescrito como

$$\pi(s_i) \in \operatorname*{argmax}_{a \in A} Q^{\pi}(s_i, a). \tag{2.20}$$

Ou seja, uma vez que a função Q^{π} de uma política π seja conhecida, pode-se derivar uma política $\pi' \geq \pi$ sem o conhecimento das funções P^a e r^a.

O cálculo da função Q^{π} pode ser feito de maneira muito parecida com o da função de valor V^{π}. Por exemplo, o análogo da expressão (2.14) seria

$$Q^{\pi}(s_i, a) \leftarrow r^a(s_i) + \gamma \sum_{j=1}^{|S|} P^a(s_j|s_i) Q^{\pi}(s_j, \pi(s_j)), \tag{2.21}$$

e a regra (2.18) seria substituída por

$$Q^{\pi}(s_i, a) \leftarrow Q^{\pi}(s_i, a) + \alpha \left[r + \gamma Q^{\pi}(s_j, \pi(s_j)) - Q^{\pi}(s_i, a) \right]. \tag{2.22}$$

Com base nessas regras de atualização pode-se definir métodos de avaliação de política análogos aos Algoritmos 2.4 e 2.5. Da mesma forma, a iteração de política generalizada pode ser reescrita com Q^{π} no lugar de V^{π}. A vantagem nesse caso é que a melhoria de política seria implementada por (2.20) ao invés de (2.13).

Q-learning

Uma versão particularmente importante da iteração de política generalizada usando a função Q é aquela em que a aplicação de (2.22) é imediatamente seguida de (2.20). Assim como no caso da iteração de valor, essas duas operações podem ser combinadas para formar uma única regra de atualização:

$$Q(s_i, a) \leftarrow Q(s_i, a) + \alpha \left[r + \gamma \max_{a'} Q(s_j, a') - Q(s_i, a) \right]. \tag{2.23}$$

A regra acima dá origem ao conhecido algoritmo *Q-learning* [172, 173]. O *Q-learning* permite que a política ótima de um MDP seja determinada enquanto o agente interage

com o MDP seguindo uma política de exploração π_e qualquer. A única restrição em relação à política π_e é que todas as ações $a \in A$ sejam executadas um número infinito de vezes em cada estado do MDP. Se esse for o caso e se α decrescer a uma taxa adequada, a função Q calculada pelo algoritmo converge assintoticamente para Q^*, a função de valor de ação ótima do MDP. Com base nessa função, pode-se determinar todas as políticas ótimas do processo de decisão de Markov usando (2.20). O *Q-learning* é mostrado de forma esquemática no Algoritmo 2.6

Algoritmo 2.6 *Q-learning*

Requer $\pi_e \in A^{|S|}$, $\gamma \in [0, 1)$
Retorna Q^*
  $\mathbf{Q} \leftarrow$ matriz aleatória em $\mathbb{R}^{|S| \times |A|}$
  repita
    $i \leftarrow$ número aleatório em $\{1, 2, ..., |S|\}$
    repita
      $a \leftarrow \pi_i^e$
      execute a transição $s_i \xrightarrow{a} s_j$ e guarde a recompensa recebida r
      $k \leftarrow$ coluna de $\mathbf{Q}$ que corresponde à ação a
      $q_{ik} \leftarrow q_{ik} + \alpha \left[r + \gamma \max_l q_{jl} - q_{ik} \right]$
      $i \leftarrow j$
    até s_i é um estado terminal
  até critério de parada satisfeito

No algoritmo *Q-learning* a função de valor evolui independentemente da política de exploração escolhida. Isso fica claro quando se observa que a regra de atualização (2.23) não envolve as ações executadas pelo agente. O uso de uma política de exploração tem suas vantagens, claro, mas ele também dificulta o uso da atualização com rastro adotada no algoritmo TD(λ). Embora existam na literatura sugestões de como isso pode ser feito [172, 116, 118], parece mais natural usar um rastro quando a atualização é realizada pelo regra (2.22)—ou seja, de acordo com as transições efetivamente executadas pelo agente. Uma possibilidade nesse sentido é abandonar a idéia de uma política de exploração e coletar os dados de acordo com a política sendo refinada (ou uma versão ligeiramente modificada dela, como discutido anteriormente). Em outras palavras, ao invés de combinar as regras (2.22) e (2.20) em uma só, pode-se aplicá-las de forma alternada a uma política π. Essa estratégia dá origem ao algoritmo SARSA, cujo nome deriva da

seqüência $\{s_i, a_i, r, s_j, a_j\}$ que caracteriza uma transição do agente [134, 135, 160]. Assim como o Q-*learning*, o algoritmo SARSA converge para a política ótima de um MDP, contanto que algumas condições simples sejam atendidas [150].

2.3.4 Planejamento

A função de valor de ação combinada com o método das diferenças temporais permite que a busca por uma das políticas ótimas de um problema de tomada de decisão seja efetuada sem o conhecimento explícito do MDP. Note que quando esses dois conceitos são usados conjuntamente as funções P^a e r^a não são nem mesmo estimadas; a função ótima Q^* é calculada diretamente a partir de amostragem. Embora essa abordagem apresente vantagens óbvias, ela sofre de um sério inconveniente. Como o processo de refinamento da função Q^* precisa ser feito lentamente, é necessária uma grande quantidade de experiência do agente no ambiente. Em outras palavras, uma boa aproximação da função de valor requer um grande número de transições do tipo $s_i \xrightarrow{a} s_j$.

Uma solução potencial para esse problema seria usar as transições executadas pelo agente para estimar as funções P^a e r^a, mas, como discutido no início desta seção, tentar aproximar o MDP todo de uma vez pode não ser uma boa estratégia em problemas de grande porte. Uma solução intermediária é construir um modelo do MDP de forma gradativa, à medida que a informação for se tornando disponível. Nesse caso, ao invés de descartar as transições após cada atualização pelo método TD, pode-se usá-las para refinar uma aproximação das funções P^a e r^a. A vantagem em relação à construção de um modelo antes do início da aprendizagem é que nesse caso a coleta de dados faz parte do problema, ou seja, a política de exploração usada para gerar as transições é determinada levando em consideração a aproximação corrente do MDP.

A Figura 2.5 mostra de forma esquemática a abordagem descrita acima. Como ocorre na aprendizagem por reforço direta, cada transição executada pelo agente pode ser usada para atualizar a função de valor, através da regra (2.22), por exemplo. A diferença é que nesse caso a função de valor pode ser atualizada também com base em experiência simulada, ou seja, a partir de um modelo aproximado do MDP. Como discutido, esse modelo é

refinado gradativamente com base na experiência real do agente. Note que a atualização da função de valor com base em um modelo é um processo completamente "interno" ao agente. Não é de se espantar que esse processo seja chamado de *planejamento* [161].

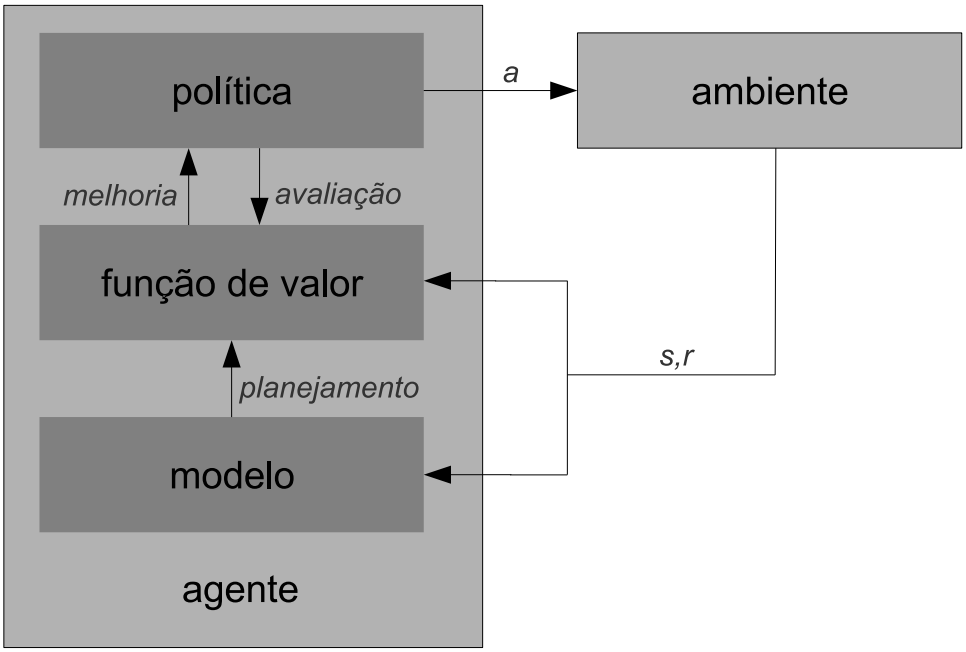


Figura 2.5: Aprendizagem por reforço com planejamento.

O esquema mostrado na Figura 2.5 confere grande flexibilidade ao processo de busca por uma das políticas ótimas de um MDP. Pode-se separar nitidamente as fases de coleta de dados e de planejamento. Nesse caso, as interações do agente com o ambiente são intercaladas por longos períodos de processamento, em que um dos algoritmos de programação dinâmica estudados na Seção 2.2 é usado para determinar a política ótima do MDP aproximado [73, 29]. É possível também deixar que a interação entre aprendizagem e planejamento ocorra em um nível maior de granularidade. Uma possibilidade nesse sentido é alternar uma aplicação de (2.22) baseada em uma transição real do agente com várias aplicações de (2.21) feitas com base no MDP aproximado [158, 159]. Para tornar o processo de aprendizagem mais eficiente, pode-se ordenar as atualizações segundo algum critério, priorizando por exemplo aquelas que promoverão as mudanças mais drásticas na função de valor [105, 117].

As vantagens e desvantagens em se usar um modelo aproximado de um MDP é tema de algum debate na comunidade de aprendizagem por reforço [4, 74]. De uma maneira

superficial, pode-se dizer o seguinte: como no caso da aprendizagem direta a função de valor é atualizada com base em informação "real," extraída diretamente do ambiente, as atualizações costumam ser mais efetivas—e portanto um número menor delas é necessário para a convergência. Quando se usa um modelo do MDP, é possível extrair muito mais informação de cada transição executada pelo agente, uma vez que os mesmos dados são usados em várias atualizações. Pelo mesmo motivo, o efeito de *cada* atualização tende a ser menor. Portanto, quando se usa um modelo a convergência ocorre em um número menor de transições, mas depende de um número maior de atualizações. Se a interação do agente com o ambiente é um processo "barato" (no sentido mais amplo do termo), a aprendizagem por reforço direta pode ser uma boa escolha. Nos casos em que os recursos computacionais tornam o processamento mais acessível do que a coleta direta de dados, o uso de um modelo é aconselhável [71].

2.3.5 Uma visão unificada

O algoritmo TD pode ser interpretado como a versão incremental da avaliação iterativa de política. Da mesma forma, o *Q-learning* pode ser visto como uma iteração de valor incremental. Ambos os casos se explicam pela diferença entre (2.21) e (2.22). Em se tratando da avaliação de política, a primeira dá origem ao Algoritmo 2.4, ao passo que a segunda origina o algoritmo TD. Quando (2.21) é combinada com a melhoria de política (2.20), surge a iteração de valor. Da mesma forma, o algoritmo *Q-learning* é o resultado da fusão de (2.22) com (2.20). Essa miscelânea de equações deixa claro que os algoritmos de aprendizagem por reforço nada mais são do que versões incrementais dos algoritmos tradicionais da programação dinâmica.

Uma maneira de visualizar essa questão é imaginar que o espaço representado na Figura 2.4 tem uma quarta dimensão, em que figura o número de estados sucessores de s_i envolvidos na sua atualização. Nessa imagem tanto a programação dinâmica quanto a aprendizagem por reforço representariam casos extremos: enquanto na primeira todos os sucessores de s_i são usados na atualização, na segunda apenas um estado s_j é considerado. Evidentemente, pode-se pensar em algoritmos intermediários, em que um subconjunto dos sucessores de s_i é levado em consideração. Esses algoritmos podem ser usados por

exemplo em MDPs em que o fator de ramificação é grande, ou seja, em que a execução de uma ação em um determinado estado pode levar a muitos estados diferentes. Como observam Sutton e Barto [161], nesses casos pode ser melhor fazer um grande número de atualizações incompletas do que apenas uma usando (2.21). Uma outra situação em que atualizações intermediárias podem ser úteis é quando se tem um modelo incompleto do MDP, em que se conhece o efeito de algumas ações apenas.

Essa visão unificada da programação dinâmica e da aprendizagem por reforço traz benefícios para ambas. Ao ser interpretada como uma versão incremental da primeira, a aprendizagem por reforço encontra ao seu dispor uma teoria madura, bem fundamentada matematicamente e consolidada ao longo de anos de estudo e pesquisa. De fato, os resultados mais importantes da área derivam diretamente da interpretação da aprendizagem por reforço como uma forma aproximada de programação dinâmica [172, 68, 168, 22]. Da mesma forma, a programação dinâmica se beneficia com os algoritmos desenvolvidos especificamente para o cenário considerado pela aprendizagem por reforço. Graças a conceitos desenvolvidos por pesquisadores da área (muitas vezes de forma intuitiva), atualmente é possível lidar com um problema de tomada de decisão seqüencial em tempo real e sem um modelo do MDP—algo impensável há alguns anos.

Neste trabalho adoto essa visão unificada, e muitas vezes uso o termo "programação dinâmica" de forma genérica, para me referir tanto às técnicas estudadas na Seção 2.2 quanto à aprendizagem por reforço. Nos contextos em que a distinção entre as duas disciplinas é crucial eu deixo isso claro.

2.3.6 Um parágrafo de história

A aprendizagem por reforço é o resultado da convergência de dois ramos de pesquisa que se desenvolveram de maneira independente até se encontrarem no final da década de oitenta. Um deles, claro, é a programação dinâmica—em particular os trabalhos de Bertsekas sobre algoritmos assíncronos [19, 21] e a programação dinâmica em tempo real, estudada por Witten [187, 188] e Werbos [175, 176, 174]. O segundo grande ramo de pesquisa que culminou na aprendizagem por reforço engloba pesquisadores de diversas

disciplinas, mas sem dúvida as contribuições mais importantes são provenientes da área de inteligência artificial. Uma idéia fundamental para o desenvolvimento da área é a de que a aprendizagem deve ocorrer enquanto o agente interage com o ambiente. De acordo com Sutton e Barto [161], Klopf [77, 78, 79] foi o grande responsável por enfatizar a importância desse aspecto da aprendizagem, que está ausente por exemplo no cenário estudado pela aprendizagem supervisionada. Um outro conceito que está no cerne da aprendizagem por reforço é o das diferenças temporais, que aparece inicialmente no trabalho de Barto, Anderson e Sutton [10] e que mais tarde foi formalizado por Sutton [157]. Em geral, atribui-se a primeira aplicação dessa idéia ao trabalho de Samuel [139, 140], que desenvolveu um sistema capaz de jogar o jogo de damas em um nível intermediário. O "sistema classificador" de Holland [63] também usava conceitos que podem ser claramente identificados com o método das diferenças temporais.[7] O algoritmo *Q-learning*, desenvolvido por Watkins [172, 173], é outro marco da área, por dispensar o uso de um modelo na busca pela política ótima de um MDP. Watkins também foi um dos primeiros a relacionar de maneira explícita a programação dinâmica com a aprendizagem por reforço. Um dos grandes responsáveis pela atenção que a aprendizagem por reforço vem recebendo desde a década de noventa é Tesauro [163, 164, 165], que criou um sistema baseado no algoritmo TD(λ) capaz de jogar gamão no nível dos melhores jogadores do mundo. Existem disponíveis vários artigos introdutórios sobre a aprendizagem por reforço [75, 71, 132, 8], além dos livros de Kaelbling [70] e Sutton e Barto [161]. Este último contém uma descrição detalhada da história da área, usada como base para esta seção. Embora não seja focado exclusivamente na aprendizagem por reforço, o livro de Bertsekas e Tsitsiklis [22] trata de muitos assuntos relevantes para a disciplina. O livro organizado por Si *et al.* [147] traz uma compilação do trabalho de vários autores e também serve como uma introdução ao assunto. Atualmente, grande parte das publicações da área se concentram no problema da instabilidade causada pelo uso de aproximadores na aprendizagem por reforço. Esse assunto será tratado na próxima seção.

[7] Holland ficou mais conhecido como o "pai" dos algoritmos genéticos, que eram um outro componente do seu sistema.

2.4 Programação Dinâmica Aproximada

Em princípio, a programação dinâmica e a aprendizagem por reforço podem ser aplicadas a qualquer problema de tomada de decisão seqüencial que possa ser modelado como um MDP finito. Quando a dinâmica do processo de decisão de Markov é conhecida pode-se adotar a primeira; nos casos em que P^a e r^a não estão disponíveis é preciso recorrer às técnicas de aprendizagem por reforço. Embora isso seja verdade em teoria, na prática os algoritmos provenientes de ambas as áreas têm sérios problemas quando o número de estados do MDP é muito grande. Isso porque os seus requerimentos de memória e processamento tornam-se proibitivos à medida que $|S| \rightarrow \infty$.

A questão da escalabilidade da programação dinâmica torna-se particularmente grave quando se considera que a cardinalidade de S cresce exponencialmente com a sua dimensão, um problema que ficou conhecido como a *maldição da dimensionalidade de Bellman* [15]. Para entender a maldição de Bellman, basta imaginar que cada variável do espaço S pode assumir um de n valores possíveis. Nesse caso, $|S| = n^{\dim(S)}$. Muitos dos problemas de tomada de decisão tratados pela programação dinâmica têm um espaço de estados que envolve naturalmente um grande número de variáveis. Por exemplo, Powell e Roy [147] discutem uma tarefa de alocação ótima de recursos em que a dimensão de S alcança a casa dos milhares. Imagine os recursos computacionais necessários para resolver um MDP dessa magnitude. Mesmo se a dimensão de S estiver dentro de limites tratáveis, pode acontecer de suas variáveis serem contínuas. Esse é o caso, por exemplo, de muitos problemas de controle [130].

Em casos como os citados acima os métodos convencionais da programação dinâmica e da aprendizagem por reforço não se aplicam. É preciso apelar para alguma forma de aproximação. Existem basicamente duas maneiras de introduzir a aproximação na programação dinâmica. Uma delas é criar um modelo aproximado do MDP, nos moldes da estratégia discutida na Seção 2.3.4. Em contraste com o que foi estudado, no entanto, nesse contexto o modelo deve conter um número reduzido de estados em relação ao MDP original. Uma outra estratégia é usar um aproximador que permita a generalização da função de valor sobre o espaço de estados S. Nas seções abaixo discuto cada uma dessas estratégias em detalhes.

2.4.1 Aproximação do modelo

A solução mais imediata no caso em que S é contínuo é simplesmente discretizá-lo e construir um MDP finito. Nesse caso, cada estado do modelo aproximado corresponde a um região do espaço original. A dinâmica do MDP pode ser determinada através de (2.17). Caso as funções P^a e r^a originais sejam Lipschitz contínuas [80], então para qualquer $\epsilon > 0$ é possível obter uma discretização do espaço de estados que resulte em uma função de valor $\tilde{V}$ tal que $\| V - \tilde{V} \|_\infty < \epsilon$ [33]. Existem na literatura limites bem definidos sobre qual deve ser a resolução da discretização de forma que essa precisão na aproximação seja alcançada [181, 182].

Apesar das garantias teóricas, a discretização do espaço de estados apresenta um inconveniente. Como o particionamento de S é feito antes do início da aprendizagem, é difícil determinar qual seria a resolução adequada para resolver o problema. Além disso, na falta de informação prévia recorre-se normalmente a uma discretização uniforme de S, o que em alguns casos pode representar um grande desperdício de recursos. Seria mais inteligente particionar o espaço de estados de acordo com a importância de cada região para o problema de tomada de decisão. Por exemplo, Barto *et al.* [10] usaram conhecimento prévio a respeito de um problema de controle para discretizar o seu espaço de estados de maneira mais eficiente. O objetivo do problema estudado pelos autores é manter um bastão equilibrado pelo máximo de tempo possível. O espaço de estados foi particionado de forma que a resolução da discretização fosse maior na região em que a velocidade dos componentes do sistema é menor. O problema de equilibrar um bastão será estudado em detalhes na Seção 4.4.

Baseado na constatação acima, Moore [104] propôs que a discretização fosse incorporada ao problema de aprendizagem, e apresentou um algoritmo capaz de particionar o espaço de estados de forma adaptativa. Em seu artigo o autor apresenta exemplos de problemas intratáveis com técnicas de discretização convencionais em que seu algoritmo é bem-sucedido. A desvantagem da abordagem proposta é que ela se restringe a problemas episódicos, além de exigir o conhecimento de uma trajetória razoável até um estado terminal. Mais tarde, Moore propôs ao lado de Atkenson [103] um outro algoritmo capaz de realizar a discretização de S durante o processo de aprendizagem. Trata-se de um método

extremamente rápido, capaz de encontrar uma política de decisão razoável em espaços de estados com até nove dimensões em menos de um minuto. No entanto, esse algoritmo exige que o agente tenha acesso a um "controlador local" capaz de guiá-lo até a célula mais próxima, além de se restringir a ambientes determinísticos.

Talvez a forma mais simples de lidar com espaços de estados grandes ou contínuos seja *agregar* os seus estados. A agregação de estados não é exatamente uma idéia nova, e é possível encontrá-la em alguns dos trabalhos pioneiros da área de programação dinâmica [18] e aprendizagem por reforço [99]. Atualmente, existem resultados teóricos que fornecem garantias em relação à convergência e ao desempenho dessa estratégia [169, 22]. De uma maneira geral, pode-se dizer que o sucesso da agregação depende da existência de grupos de estados com características similares (por "características similares" entenda-se funções de valor parecidas). No entanto, mesmo que existam tais grupos, é razoável esperar que o grau de similaridade entre eles não seja constante dentro de cada subconjunto. Por exemplo, podem existir estados que se encontram no "limite" entre dois grupos, e portanto não pertencem naturalmente a nenhum dos dois. O modelo gerado pela agregação de estados é incapaz de representar níveis intermediários de similaridade entre estados; dois estados ou pertencem ao mesmo grupo ou não. Isso pode originar grupos heterogêneos de estados que acabam por prejudicar a aproximação.

Para contornar essa limitação, Singh *et al.* [149] propuseram uma forma de agregação "suave," em que um estado está associado com vários grupos através de diferentes graus de pertinência. Em seu artigo os autores apresentam uma forma modificada da equação de Bellman em que cada estado corresponde a um grupo no MDP original, e mostram que o seu algoritmo converge para a solução dessa equação. Obviamente, a política derivada do modelo aproximado não apresentará um desempenho ótimo no problema original, mas em alguns casos essa estratégia permite obter soluções razoáveis dentro de um custo computacional aceitável. Na Seção 4.2 discuto uma estratégia de atribuição que se assemelha às idéias de Singh *et al.* e que é fundamental para a proposta deste trabalho.

Gordon [55, 54] trata do problema de escalabilidade da programação dinâmica de uma maneira particularmente interessante. Ele considera uma classe especial de aproximadores chamados *averagers*. Como mostra o autor, quando um *averager* é usado para

aproximar a função de valor, a solução aproximada de um MDP equivale à solução exata de um processo de decisão de Markov derivado do original. Com base nessa observação, Gordon prova que o seu esquema de aproximação é estável, no sentido em que ele sempre converge para uma solução. Rust [136] e Ormoneit e Sen [111] apresentam estratégias para aproximar as funções P^a e r^a de um MDP que podem ser consideradas extensões das idéias de Gordon para o caso de espaços de estados contínuos. A abordagem de Ormoneit e Sen serve como base para um dos algoritmos propostos neste trabalho, e será analisada em detalhes na Seção 4.3.

Uma estratégia para lidar com espaços de estados grandes que vem recebendo muita atenção nos últimos anos são os métodos hierárquicos [9, 71, 147]. Os métodos hierárquicos são um exemplo típico de uma técnica conhecida na ciência da computação como *divisão-e-conquista*. Nesse caso, o problema de tomada de decisão seqüencial é desmembrado em uma série de sub-tarefas mais simples organizadas hierarquicamente. Dessa forma, o agente lida com o problema em vários graus de abstração diferentes (na Seção 2.1 foi discutido como um método hierárquico poderia ser usado para ajudar um agente a dirigir um automóvel). Embora as políticas de decisão encontradas pelos algoritmos hierárquicos sejam em geral sub-ótimas, a simplificação da tarefa e conseqüente redução do custo computacional oferecidos por esses métodos podem torná-los alternativas atraentes em alguns casos. Isso é particularmente verdade em problemas em que a mesma seqüência de ações pode ser reutilizada em várias situações. Barto e Mahadevan [9] apresentam uma excelente revisão sobre os métodos hierárquicos, assim como uma proposta para unificar a terminologia ainda em evolução. Outros apanhados sobre o assunto podem ser encontrados nos artigos de Kaelbling *et al.* [71] e Ryan [147].

2.4.2 Aproximação da função de valor

Uma outra alternativa para lidar com espaços de estados contínuos ou muito grandes é aproximar a função de valor. Nesse caso, o objetivo passa a ser ajustar os parâmetros de um aproximador que computa um mapeamento $\tilde{V} : S \mapsto \mathbb{R}$ (ou $\tilde{Q} : S \times A \mapsto \mathbb{R}$). Como em geral o número de parâmetros livres do aproximador é muito menor do que $|S|$, não é possível armazenar o valor de cada estado explicitamente, o que significa que o agente

deve ser capaz de *generalizar*.

A generalização a partir de exemplos é o cenário estudado pela *aprendizagem supervisionada*, já citada no Capítulo 1 [102, 60, 144]. A aprendizagem supervisionada é uma matéria multidisciplinar que envolve pesquisadores de diversas áreas, entre elas a inteligência artificial e a estatística. Atualmente, a quantidade de resultados teóricos acumulados, sem falar nos achados empíricos, torna uma descrição detalhada da área uma tarefa hercúlea, que vai muito além do escopo deste trabalho. O leitor interessado no uso de aproximadores na programação dinâmica deve se dirigir ao livro de Bertsekas e Tsitsiklis [22], que apresentam um tratado minucioso sobre o assunto. Aqui, tentarei dar apenas uma visão histórica sobre os sucessos e fracassos das tentativas de se aproximar a função de valor.

A história da aprendizagem supervisionada se confunde com a de muitas outras áreas do conhecimento, e a programação dinâmica e a aprendizagem por reforço não são exceções. Na implementação do seu jogador de damas artificial Samuel [139, 140] já utilizava esquemas de aproximação que podem ser claramente identificados com técnicas modernas da aprendizagem supervisionada. Um exemplo mais atual é o sistema de Tesauro [164], que combinou o algoritmo TD(λ) com um *percéptron* de múltiplas camadas para aproximar a função de valor ótima do jogo de gamão. Não deixa de ser surpreendente que a política de decisão derivada desse aproximador é um dos melhores jogadores de gamão do mundo! Mas o sucesso de aproximadores na programação dinâmica não se restringe aos jogos de tabuleiro. Vários outros exemplos de aplicações bem-sucedidas podem ser citados, entre eles: a alocação dinâmica de canais de comunicação de telefones celulares [148], o agendamento de tarefas em missões da NASA [192], o controle de descarregadores de navios [142], o gerenciamento automático de elevadores [35] e a regulação de uma rede de irrigação [57]. Na robótica, algoritmos de aprendizagem por reforço combinados com aproximadores foram usados para a navegação [91, 42] e para ajudar robôs a andarem [16] e a jogarem futebol [155].

Apesar das histórias de sucesso, existem limitações fundamentais no uso de aproximadores com a programação dinâmica. Como visto, os cálculos realizados pelos algoritmos dessa disciplina têm uma natureza recorrente, uma vez que a atualização da função de

valor em um estado s_i é feita com base no valor de outros estados (*cf.* (2.21) e (2.22)). Assim, um erro de aproximação pode ser facilmente reabsorvido e potencializado pelo processo de aprendizagem, causando instabilidade ou até mesmo divergência dos algoritmos. Esse fenômeno foi estudado por vários autores [166, 137, 27, 169, 170].

O contraste entre os obstáculos teóricos e alguns sucessos práticos com o uso de aproximadores despertou grande interesse pelo assunto. Alguns dos trabalhos iniciais apresentaram resultados desencorajadores, com exemplos benignos em que algoritmos convencionais da programação dinâmica combinados com aproximadores simples falhavam acintosamente [27, 5, 55, 169]. Mais tarde, Sutton [160] mostrou que alguns desses fracassos poderiam ser sanados se fossem adotados aproximadores lineares e um esquema de exploração ativo, em que a atualização da função de valor é feita segundo a política em processo de refinamento. Baseados nessa observação, Tsitsiklis e Roy [170] provaram que o algoritmo TD(λ) sempre converge para uma solução quando usado com essa configuração. Esse resultado criou uma forte tendência à utilização de aproximadores lineares na programação dinâmica [162, 131], movimento este que culminou em diversas propostas de algoritmos aproximados estáveis [126, 143, 119].

Dentre as combinações estáveis de aproximadores lineares com a programação dinâmica, merecem destaque os algoritmos baseados no trabalho de Bradtke e Barto [28, 26, 190, 82]. Nesses algoritmos a avaliação de política se reduz a um problema de ajuste linear, em que o objetivo é minimizar uma função de custo quadrática. Esse é um problema de otimização bem conhecido, para o qual existem métodos de solução estáveis e muito eficientes [127, 50]. Comparados com o algoritmo TD(λ) original, os métodos baseados no trabalho de Bradtke e Barto fazem um uso muito mais eficiente dos dados, além de não requererem a definição de uma taxa de aprendizagem. Na Seção 4.4 um desses métodos é usado como referência na análise de desempenho de um algoritmo proposto neste trabalho.

2.5 Outras Soluções

É possível solucionar problemas de tomada de decisão seqüencial usando outras abordagens além da programação dinâmica e da aprendizagem por reforço. Nesta seção descrevo brevemente duas possibilidades e discuto por que no atual estágio de desenvolvimento teórico a programação dinâmica ainda parece ser a melhor alternativa.

2.5.1 Programação linear

A busca por uma das políticas ótimas de um processo de decisão de Markov pode ser interpretada como a procura pela solução de um sistema de equações não-lineares. Nesse caso, o objetivo é encontrar o vetor $\mathbf{v} \in \mathbb{R}^{|S|}$ cujos elementos satisfazem a relação

$$v_i = \max_{a \in A} \left(r_i^a + \gamma \sum_{j=1}^{|S|} p_{ij}^a v_j \right), \tag{2.24}$$

que é a equação de Bellman do MDP. Sabe-se que o único vetor que satisfaz a equação de Bellman é $\mathbf{v}^*$, de onde podem ser derivadas todas as políticas ótimas do processo de decisão de Markov [128, 22].

O sistema (2.24) pode ser reformulado como um problema de *programação linear* [40]. Assim, algoritmos como o *simplex* e as suas variantes podem ser usados para encontrar a função de valor ótima de um MDP [37]. A vantagem de se usar a programação linear em um problema de tomada de decisão seqüencial é que essa disciplina conta com uma base teórica bem consolidada, além de diversos algoritmos implementados disponíveis para uso. Adicionalmente, a formulação do problema usada pela programação linear facilita a inclusão de restrições ao sistema (2.24) [128]. No entanto, como a programação linear não explora a estrutura particular de um MDP—como o faz a programação dinâmica—, o seu desempenho em problemas de tomada de decisão é muito inferior ao dos algoritmos apresentados neste capítulo [92, 128, 161].

2.5.2 Busca no espaço de políticas

Os algoritmos de programação dinâmica fazem uma busca indireta por uma das políticas ótimas de um dado MDP, usando como suporte o conceito de função de valor. Uma alternativa a essa abordagem é realizar a busca diretamente no espaço das políticas de decisão. A forma mais natural de fazê-lo é usar um modelo paramétrico para representar as políticas candidatas, ou seja, a política π é substituída por um aproximador que computa um mapeamento $\tilde{\pi} : S \mapsto A$. O objetivo do problema passa a ser configurar os parâmetros desse aproximador de forma a maximizar as recompensas coletadas por $\tilde{\pi}$. Uma vantagem óbvia dessa abordagem é que ela pode ser naturalmente aplicada a espaços de estados grandes ou contínuos.

A configuração de $\tilde{\pi}$ é um problema de otimização não-linear. Portanto, qualquer algoritmo desenvolvido para resolver esse tipo de problema pode a princípio ser usado para encontrar uma política de decisão. Uma possibilidade é estimar o gradiente de (2.4) e atualizar os parâmetros de $\tilde{\pi}$ na direção que maximiza o ganho de recompensa [185, 11, 12, 156, 147]. Grande parte dos esforços despendidos na busca por uma solução desse tipo está no desenvolvimento de técnicas confiáveis e computacionalmente viáveis para se estimar o gradiente. Um dos desafios atuais é reduzir a quantidade de dados necessária para esse tipo de estimativa [147]. Um ponto favorável dessa abordagem é que quando é possível calcular o gradiente o algoritmo resultante converge garantidamente para uma solução. Um ponto negativo é que essa solução pode ser uma política sub-ótima.

Uma alternativa para tentar contornar essa limitação é adotar métodos de otimização global, dentre os quais os algoritmos evolucionários são provavelmente os mais conhecidos [49]. Os algoritmos evolucionários já foram usados em diversos problemas de tomada de decisão, com grande sucesso [96, 52]. De fato, não é raro encontrar na literatura relatos de experimentos computacionais em que esses métodos superaram de forma consistente algoritmos baseados na programação dinâmica [179, 51]. Pollack [122, 123] chega a sugerir que o sucesso do algoritmo de Tesauro [164], considerado um dos grandes feitos da aprendizagem por reforço, está mais relacionado com as características do jogo de gamão do que com o método de solução adotado. O autor mostra um algoritmo simples que consegue atingir rapidamente um nível de jogo similar ao obtido por Tesauro em

seus experimentos iniciais (embora ainda muito distante do desempenho alcançado mais tarde).

No entanto, existem limitações inerentes à interpretação do problema de tomada de decisão seqüencial como uma otimização convencional. Como se sabe, os algoritmos de otimização global seguem todos o mesmo princípio básico, que é a geração e a avaliação de soluções candidatas. Esse mecanismo parece ser adequado a problemas de decisão em que a aprendizagem se estende por um grande número de episódios curtos. No caso em que a interação do agente com o ambiente é contínua ou se estende por longos períodos de tempo, torna-se difícil para um método de otimização avaliar as soluções candidatas. Os métodos baseados na programação dinâmica, em especial nas diferenças temporais, podem lidar com esse tipo de problema porque são capazes de avaliar e refinar uma política de forma incremental, *a cada transição executada*. Pode-se dizer que esses algoritmos realizam uma otimização intra-episódio, ao passo que a otimização realizada por outros métodos ocorre nos intervalos entre as interações do agente com o ambiente.

Existe ainda uma outra situação em que métodos de otimização "inter-episódios" podem falhar. Em alguns problemas de tomada de decisão o término de um episódio depende diretamente da seqüência de ações executadas pelo agente. Ou seja, a interação do agente com o ambiente *pode* ser quebrada em episódios e estes *podem* ser curtos, mas ambos os fatos dependem de um bom desempenho da política de decisão. Políticas de má qualidade ficam "presas" no MDP, potencialmente por um tempo indeterminado (uma imagem que pode ajudar aqui é a de um labirinto). Nesse tipo de problema um método de otimização global precisa encerrar o episódio após um certo número de transições, sob o risco de ficar retido indefinidamente na avaliação de uma política candidata. Como as política fracassadas têm todas a mesma avaliação, o processo de otimização fica reduzido a uma busca cega até que uma solução razoável seja encontrada por sorte. Dependendo do nível de dificuldade do problema, isso pode demorar muito a acontecer. Para demonstrar essa questão, em um trabalho recente eu modifiquei ligeiramente uma tarefa de controle que já havia sido resolvida anteriormente por um algoritmo evolucionário [7]. Na versão modificada do problema, que é um pouco mais difícil do que a original, nenhum método evolucionário foi capaz de encontrar uma única solução para a tarefa em questão. Os métodos baseados nas diferenças temporais não tiveram problemas.

Capítulo 3

Fatoração Estocástica de um Processo de Decisão de Markov

"Todos os caminhos levam a Roma."

Dito popular

A busca por uma das políticas ótimas de um MDP pode ser formulada como um problema de programação linear, e portanto pode ser efetuada em tempo polinomial. Embora essa garantia teórica seja reconfortante, na prática sabe-se que espaços de estados razoavelmente grandes inviabilizam o uso tanto da programação linear quanto da programação dinâmica. Como discutido no final do capítulo anterior, pode-se recorrer a duas estratégias nesse caso. Uma delas é aproximar a função de valor. A outra é criar um modelo aproximado do MDP—um modelo do modelo, por assim dizer. A proposta que apresento neste capítulo se enquadra nessa segunda categoria. Especificamente, eu mostro como gerar um modelo reduzido de um processo de Markov cuja função de valor guarda uma relação de equivalência com a função do processo original. Por ora me concentro no caso em que o espaço de estados é finito e as funções de transição e recompensa do MDP são conhecidas. No próximo capítulo mostro como a estratégia apresentada pode ser estendida a espaços contínuos e ao cenário estudado pela aprendizagem por reforço.

Este capítulo está organizado da seguinte maneira: na Seção 3.1 apresento a idéia in-

tuitiva que deu origem a este trabalho. Como será visto, é possível reduzir a dimensão de um processo de Markov simplesmente redirecionando as sua transições para estados artificiais. Na seção seguinte formalizo essa idéia, e apresento uma proposição que constitui a base teórica de todo o trabalho. Nas Seções 3.3 e 3.4 mostro como esse resultado pode ser aplicado na prática, primeiro para calcular a função de valor de um processo de Markov e a seguir para encontrar uma das políticas ótimas de um MDP. Em ambos os casos a discussão é sustentada por análises teóricas e práticas. Finalmente, na Seção 3.5 apresento as principais conclusões a respeito do assunto tratado neste capítulo e discuto possibilidades de extensões da proposta para cenários mais realísticos.

3.1 Arquétipos

Nesta seção utilizo um exemplo bem simples para apresentar de maneira intuitiva a idéia fundamental deste trabalho, que mais tarde formalizo e generalizo. Apenas a título de ilustração, suponha que se queira modelar o jogo de golfe como um processo de decisão seqüencial. Considere que o campo desse jogo tenha sido dividido em 4 setores: longa distância, média distância, *green* e buraco. Para facilitar, suponha que o jogador já tenha definido a sua estratégia para o buraco atual, que chamarei de π, e o objetivo nesse caso seja simplesmente calcular a função de valor V^π correspondente. Suponha ainda que a cadeia de Markov originada por π seja caracterizada pelas seguintes probabilidades de transição:

$$\mathbf{P}^\pi = \begin{bmatrix} 0.10 & 0.70 & 0.15 & 0.05 \\ 0.08 & 0.58 & 0.24 & 0.10 \\ 0.00 & 0.10 & 0.60 & 0.30 \\ 0.00 & 0.00 & 0.00 & 1.00 \end{bmatrix}, \tag{3.1}$$

onde a primeira linha se refere ao setor de longa-distância, a segunda ao setor de média distância, a terceira ao *green* e a quarta ao buraco. Esses setores serão identificados como os estados s_1, s_2, s_3 e s_4, respectivamente. Finalmente, considere que o jogador receba uma recompensa de 1 quando acertar o buraco e de 0 em todas as outras tacadas. Isso

resulta no seguinte vetor de recompensas associado a π (veja expressão (2.1)):

$$\mathbf{r}^{\pi} = \begin{bmatrix} 0.05 \\ 0.10 \\ 0.30 \\ 0.00 \end{bmatrix}. \tag{3.2}$$

Note que o estado s_4 ("buraco") é terminal, e por isso tem uma recompensa associada de 0. Para um fator de desconto $\gamma = 0.9$, a função de valor de π seria

$$\mathbf{v}^{\pi} = \begin{bmatrix} 0.62 \\ 0.66 \\ 0.78 \\ 0.00 \end{bmatrix}.$$

Como era de se esperar, os valores dos estados crescem à medida que as suas distâncias em relação ao objetivo diminuem. Esse exemplo seria bastante desinteressante se não fosse uma peculiaridade que em geral passa despercebida. Observe em (3.1) como a segunda linha de $\mathbf{P}^{\pi}$ pode ser obtida como uma combinação convexa da primeira com a terceira linha dessa matriz. Especificamente,

$$\mathbf{p}_2^{\pi} = 0.8\mathbf{p}_1^{\pi} + 0.2\mathbf{p}_3^{\pi}, \tag{3.3}$$

o que corresponde a dizer que as probabilidades de transição a partir do setor de média distância são valores intermediários entre as transições relativas ao *green* e ao setor de longa distância—o que parece razoável. Observe que o mesmo acontece com os elementos do vetor $\mathbf{r}^{\pi}$:

$$r_2^{\pi} = 0.8r_1^{\pi} + 0.2r_3^{\pi}. \tag{3.4}$$

O leitor astuto não tardará em verificar se o mesmo ocorre com a função $\mathbf{v}^{\pi}$, e de fato a relação se mantém:

$$v_2^{\pi} = 0.8v_1^{\pi} + 0.2v_3^{\pi}. \tag{3.5}$$

Isso é verdade para qualquer fator de desconto $0 \leq \gamma < 1$. Por que isso acontece? Aqui a relação recursiva entre estados descrita pela equação de Bellman pode ser útil. Como se sabe, v_2^{π} pode ser escrito como

$$v_2^{\pi} = r_2^{\pi} + \gamma \sum_{i=1}^{4} p_{2i}^{\pi} v_i^{\pi}. \tag{3.6}$$

Observando as relações de dependência linear em (3.3) e (3.4), pode-se reescrever (3.6) da seguinte maneira:

$$\begin{aligned} v_2^\pi &= 0.8r_1^\pi + 0.2r_3^\pi + \gamma \sum_{i=1}^4 (0.8p_{1i}^\pi + 0.2p_{3i}^\pi)v_i^\pi \\ &= 0.8\left[r_1^\pi + \gamma \sum_{i=1}^4 p_{1i}^\pi v_i^\pi\right] + 0.2\left[r_3^\pi + \gamma \sum_{i=1}^4 p_{3i}^\pi v_i^\pi\right] \\ &= 0.8v_1^\pi + 0.2v_3^\pi. \end{aligned}$$

A constatação de que o valor do estado s_2 pode ser obtido como uma combinação dos valores de s_1 e s_3 sugere que o primeiro talvez não seja realmente necessário do ponto de vista da aprendizagem. Lembre-se que v_i^π nada mais é do que o total de recompensas acumulado pelo agente ao seguir π a partir de s_i. Imagine agora que todas as transições para o estado s_2 fossem "desviadas" para os estados s_1 e s_3 com probabilidades 0.8 e 0.2, respectivamente. Observando (3.4), não é difícil perceber que o total de recompensas coletadas pelo agente permaneceria exatamente o mesmo.

Pode-se, portanto, pensar em resolver o problema inicial em uma versão reduzida da cadeia de Markov em que o estado s_2 não esteja presente. Para tal, farei uso de uma matriz que chamarei de *matriz de desvio*. A matriz de desvio $\mathbf{D}^\pi$ é uma matriz linha-estocástica que tem como objetivo realocar as probabilidades de transição em $\mathbf{P}^\pi$. Em particular, a i-ésima linha de $\mathbf{D}^\pi$ contém as "direções" de redirecionamento referentes ao estado s_i. No exemplo dado, a matriz de desvio seria a seguinte:

$$\mathbf{D}^\pi = \begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.8 & 0.0 & 0.2 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}.$$

Note que a multiplicação $\mathbf{P}^\pi\mathbf{D}^\pi$ realoca as probabilidades de transição para o estado s_2 (ou seja, os elementos na segunda coluna de $\mathbf{P}^\pi$) para s_1 e s_3, de acordo com as proporções na segunda linha de $\mathbf{D}^\pi$—justamente o que era desejado. Nesse caso, a matriz resultante seria

$$\mathbf{P}^\pi\mathbf{D}^\pi = \begin{bmatrix} 0.66 & 0.0 & 0.29 & 0.05 \\ 0.54 & 0.0 & 0.36 & 0.10 \\ 0.08 & 0.0 & 0.62 & 0.30 \\ 0.00 & 0.0 & 0.00 & 1.00 \end{bmatrix}. \tag{3.7}$$

A matriz $\mathbf{P}^\pi\mathbf{D}^\pi$ representa a cadeia de Markov derivada da cadeia original (3.1) através do deslocamento para s_1 e s_3 das transições terminadas em s_2. Como essa realocação de transições foi feita respeitando a dependência linear entre os estados, a função de valor de π não se altera ao se substituir (3.1) por (3.7), pelos motivos discutidos acima. Note, porém, que existe uma vantagem em adotar a segunda matriz no lugar da primeira: como todos os elementos na segunda coluna de $\mathbf{P}^\pi\mathbf{D}^\pi$ são nulos, o agente nunca fará uma transição para o estado s_2. Logo, os valores dos estados s_1, s_3 e s_4 não dependerão de v_2^π, e s_2 pode ser simplesmente descartado do cálculo de $\mathbf{v}^\pi$—seja qual for o método utilizado para fazê-lo. Excluir o estado s_2 corresponde a eliminar a segunda linha e a segunda coluna de $\mathbf{P}^\pi\mathbf{D}^\pi$, bem como o segundo elemento de $\mathbf{r}^\pi$. O processo de Markov reduzido seria dado por:

$$\bar{\mathbf{P}}^\pi = \begin{bmatrix} 0.66 & 0.29 & 0.05 \\ 0.08 & 0.62 & 0.30 \\ 0.00 & 0.00 & 1.00 \end{bmatrix} \quad \text{e} \quad \bar{\mathbf{r}}^\pi = \begin{bmatrix} 0.05 \\ 0.30 \\ 0.00 \end{bmatrix}, \tag{3.8}$$

que é claramente mais simples do que aquele mostrado em (3.1) e (3.2).

Observe que, ao contrário da matriz de transições original $\mathbf{P}^\pi$, a matriz $\bar{\mathbf{P}}^\pi$ não descreve necessariamente a dinâmica de um sistema real. Nesse caso, por exemplo, as probabilidades de transição em $\bar{\mathbf{P}}^\pi$ não correspondem a nenhum setor do campo de golfe, mas a estados prototípicos hipotéticos. Chamarei esses estados de *arquétipos*.[1] Assim, a primeira, segunda e terceira linhas de $\bar{\mathbf{P}}^\pi$ se referem aos arquétipos q_1, q_2 e q_3, respectivamente. O que torna os arquétipos especiais é a relação que eles mantêm com os estados originais do problema. Nesse caso, por exemplo,

$$v_1^\pi = \bar{v}_1^\pi, \quad v_3^\pi = \bar{v}_2^\pi, \quad v_4^\pi = \bar{v}_3^\pi, \tag{3.9}$$

onde $\bar{\mathbf{v}}^\pi$ é a função de valor de (3.8). Resta ainda uma última questão: como calcular o valor de s_2 a partir dos recém-calculados valores de q_1, q_2 e q_3? Observando (3.5) e (3.9), fica fácil perceber que o vetor $\mathbf{v}^\pi$ referente ao problema original pode ser facilmente obtido como

$$\mathbf{v}^\pi = \mathbf{D}\bar{\mathbf{v}}^\pi,$$

[1] O termo "arquétipo" foi usado pela primeira vez por Cutler e Breinam [36], embora com um sentido ligeiramente diferente do adotado aqui.

onde $\mathbf{D}$ é definida como:

$$\mathbf{D} = \begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.8 & 0.2 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}.$$

Note como a partir de uma idéia simples, que é o desvio de transições, é possível reduzir a dimensão de um processo de Markov sem no entanto alterar a sua solução. Embora no exemplo acima apenas um estado tenha sido excluído do processo original, não há nada que impeça a generalização dessa idéia para um número arbitrário deles. Na próxima seção eu discuto formalmente quais são as condições necessárias para que isso seja possível.

3.2 Fatoração Estocástica

Nesta seção apresento uma proposição que servirá de base para todo o raciocínio a ser desenvolvido neste trabalho. Em particular, é a partir desse resultado que generalizo a idéia de reduzir a dimensão de um processo de Markov através do redirecionamento de transições para estados arquetípicos.

O desvio de transições pode ser visto como um tipo especial de fatoração da matriz de transições, que chamarei de *fatoração estocástica* de $\mathbf{P}^\pi$. Como o nome sugere, o que caracteriza uma fatoração estocástica do tipo $\mathbf{P}^\pi = \mathbf{DK}$ é o fato de as matrizes $\mathbf{D}$ e $\mathbf{K}$ serem também matrizes estocásticas, ao lado da matriz $\mathbf{P}^\pi$. O interessante desse tipo de fatoração é que a inversão de seus termos dá origem a uma matriz de transições válida. Como será visto, em alguns casos a função de valor do processo descrito por $\mathbf{P}^\pi$ e $\mathbf{r}^\pi$ pode ser facilmente obtida a partir da função de valor de um processo de Markov com matriz de transições $\bar{\mathbf{P}}^\pi = \mathbf{KD}$. Assim, dependendo das dimensões de $\bar{\mathbf{P}}^\pi$, pode-se reduzir drasticamente o custo computacional de se calcular $\mathbf{v}^\pi$.

Para os resultados deste capítulo considerarei que se está lidando com problemas de tomada de decisão seqüencial em que o fator de desconto $\gamma < 1$. Isso torna desnecessárias suposições extras que garantam a existência de uma solução única para o sistema asso-

ciado com um processo de Markov. É importante observar que essa não é uma hipótese muito restritiva, porque em geral um problema sem desconto pode ser reformulado como um problema em que $\gamma < 1$ [161]. Além disso, acredito que os resultados apresentados aqui possam ser estendidos para o caso em que $\gamma = 1$, contanto que as tecnicalidades sejam contornadas de maneira adequada. Dito isso, posso apresentar a proposição principal deste capítulo.

Proposição 3.1. *Dada uma política π definida sobre um MDP finito, sejam $\mathbf{P}^\pi$ e $\mathbf{r}^\pi$ a matriz de transições e o vetor de recompensas que descrevem o processo de Markov induzido por essa política. Sejam uma matriz não-negativa $\mathbf{D}$, uma matriz estocástica $\mathbf{K}$ e um vetor qualquer $\bar{\mathbf{r}}^\pi$ tais que*

$$\mathbf{DK} = \mathbf{P}^\pi \tag{3.10}$$

$$\mathbf{D}\bar{\mathbf{r}}^\pi = \mathbf{r}^\pi. \tag{3.11}$$

Então, $\bar{\mathbf{P}}^\pi = \mathbf{KD}$ é uma matriz de transições válida, e a função de valor $\bar{\mathbf{v}}^\pi$ do processo de Markov descrito por $\bar{\mathbf{P}}^\pi$ e $\bar{\mathbf{r}}^\pi$ atende à seguinte igualdade:

$$\mathbf{v}^\pi = \mathbf{D}\bar{\mathbf{v}}^\pi, \tag{3.12}$$

para qualquer fator de desconto $0 \leq \gamma < 1$.

Demonstração. Começo mostrando que $\bar{\mathbf{P}}^\pi$ é uma matriz de transições válida. Como $\mathbf{K}$ é uma matriz estocástica e $\mathbf{D}$ uma matriz não-negativa, a igualdade $\mathbf{DK} = \mathbf{P}^\pi$ implica que $\mathbf{D}$ também é uma matriz estocástica. Para enxergar isso, basta notar o seguinte:

$$\begin{aligned} 1 = \textstyle\sum_j p_{ij} &= \textstyle\sum_j \sum_k d_{ik} k_{kj} \\ &= \textstyle\sum_k d_{ik} \sum_j k_{kj} \\ &= \textstyle\sum_k d_{ik}. \end{aligned} \tag{3.13}$$

Como $\mathbf{K}$ e $\mathbf{D}$ são matrizes estocásticas, a matriz $\bar{\mathbf{P}}^\pi = \mathbf{KD}$ obviamente também o é. Portanto, como $\bar{\mathbf{P}}^\pi$ é quadrada, ela constitui uma matriz de transições válida. Essa matriz juntamente com o vetor $\bar{\mathbf{r}}^\pi$ definem um processo de Markov cuja função de valor é a única solução para

$$\bar{\mathbf{v}}^\pi = \bar{\mathbf{r}}^\pi + \gamma \bar{\mathbf{P}}^\pi \bar{\mathbf{v}}^\pi, \tag{3.14}$$

com $0 \leq \gamma < 1$. A existência e unicidade de uma solução para (3.14) são garantidas pela propriedade estocástica de $\bar{\mathbf{P}}^\pi$(veja, por exemplo, o Lema 2.3.3 de Golub e Van Loan [50]

ou a Proposição 2.6 de Bertsekas e Tsitsiklis [22]). Multiplicando ambos os membros de (3.14) por $\mathbf{D}$, resulta:

$$\mathbf{D}\bar{\mathbf{v}}^{\pi} = \mathbf{D}\bar{\mathbf{r}}^{\pi} + \gamma\mathbf{D}\bar{\mathbf{P}}^{\pi}\bar{\mathbf{v}}^{\pi}. \tag{3.15}$$

A partir de (3.10), (3.11) e da igualdade $\bar{\mathbf{P}}^{\pi} = \mathbf{KD}$, pode-se reescrever (3.15) como

$$\mathbf{D}\bar{\mathbf{v}}^{\pi} = \mathbf{r}^{\pi} + \gamma\mathbf{P}^{\pi}\mathbf{D}\bar{\mathbf{v}}^{\pi}, \tag{3.16}$$

o que implica que $\mathbf{D}\bar{\mathbf{v}}^{\pi} = \mathbf{v}^{\pi}$, uma vez que o vetor $\mathbf{v}^{\pi}$ é o único ponto fixo da equação acima [128, 22]. □

Observe que a Proposição 3.1 não faz qualquer menção às dimensões das matrizes $\mathbf{D}$ e $\mathbf{K}$. Obviamente, a condição (3.10) determina o número de linhas da primeira e de colunas da segunda, que correspondem ao número de estados $|S|$ do processo original. É a dimensão livre dessas matrizes, m, que define o número de estados-arquétipos do processo de Markov derivado. Se $m \ll |S|$, a resolução de (3.14) no lugar de (3.16) pode significar uma imensa economia computacional. Em particular, se os sistemas forem resolvidos de maneira exata, o custo cai de $O(|S|^3)$ para $O(m^3)$. É interessante ressaltar a importância da estocasticidade de $\bar{\mathbf{P}}^{\pi}$ na demonstração da proposição acima. É essa propriedade que garante a existência e unicidade da solução de (3.14). Aqui a estocasticidade de $\bar{\mathbf{P}}^{\pi}$ é uma conseqüência direta da mesma propriedade de $\mathbf{D}$ e $\mathbf{K}$. Fica claro, portanto, que a fatoração estocástica de $\mathbf{P}^{\pi}$ constitui o cerne da estratégia proposta para a redução de um processo de Markov.

Note que o exemplo da seção anterior é um caso especial da Proposição 3.1, em que $\mathbf{K} = \mathbf{P}^{\pi}$ e $\bar{\mathbf{r}}^{\pi} = \mathbf{r}^{\pi}$. Como discutido naquele caso, $\mathbf{D}$ pode ser vista como uma *matriz de desvio* que define as probabilidades de transição dos estados originais do problema para estados arquetípicos. De maneira análoga, os elementos da matriz $\mathbf{K}$ podem ser interpretados como as probabilidades de transição dos arquétipos de volta aos estados originais. Por isso, $\mathbf{K}$ será chamada de *matriz de retorno*.[2] Vistos sob essa perspectiva, a matriz $\bar{\mathbf{P}}^{\pi}$ e o vetor $\bar{\mathbf{r}}^{\pi}$ representam, respectivamente, as probabilidades de transição entre os arquétipos e as recompensas associadas a essas transições. Para entender isso, basta

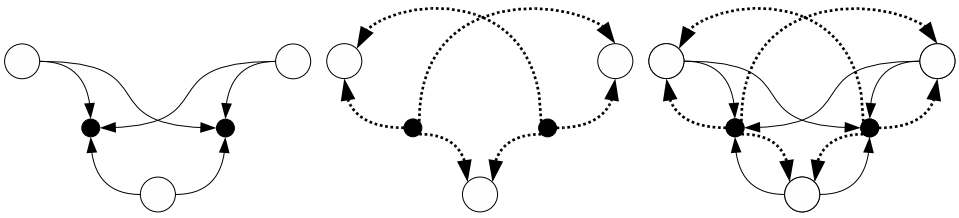
[2] O uso de "$\mathbf{K}$" ao invés de "$\mathbf{R}$"—que seria uma referência mais natural ao nome adotado para a matriz—visa evitar confusões com a matriz de recompensas, cuja notação já está bastante estabelecida na programação dinâmica.

lembrar que $\bar{\mathbf{P}}^{\pi} = \mathbf{KD}$, o que significa que o elemento $\bar{p}^{\pi}_{ij}$ é o produto escalar entre a i-ésima linha de $\mathbf{K}$ e a j-ésima coluna de $\mathbf{D}$:

$$\bar{p}^{\pi}_{ij} = \sum_{k} k_{ik} d_{kj}.$$

A expressão acima pode ser lida como: a probabilidade de transição do arquétipo q_i para o arquétipo q_j é dada pela soma das probabilidades de transição de q_i para cada estado original do problema s_k multiplicadas pelas probabilidades de transição desses últimos de volta ao arquétipo q_j.

A Figura 3.1 traz uma representação gráfica das transições entre estados e arquétipos para o caso em que o processo original tem três estados que podem ser reduzidos a dois arquétipos. As Figuras 3.1a e 3.1b mostram as transições na matriz de desvio e de retorno, respectivamente. Note que, embora essas matrizes tenham o mesmo número de elementos, a sua forma é diferente: no caso representado pela Figura 3.1, a matriz $\mathbf{D}$ seria 3×2 e a matriz $\mathbf{K}$ seria 2×3. É interessante notar também que essas matrizes sozinhas não representam cadeias de Markov, porque elas só contêm transições em um sentido: de estados para arquétipos em $\mathbf{D}$ e de arquétipos para estados em $\mathbf{K}$. A Figura 3.1c mostra as transições resultantes da composição de $\mathbf{D}$ e $\mathbf{K}$, que aí sim representam cadeias de Markov válidas. Para interpretar o desenho basta aplicar uma regra simples: as transições entre os estados originais são representadas por uma seta contínua seguida de uma pontilhada, ao passo que as transições entre arquétipos seguem o padrão contrário. A composição de uma seta contínua com uma pontilhada corresponde à multiplicação de seus valores (na figura não mostro probabilidades associadas a cada transição para deixar os desenhos mais claros). Assim, a probabilidade de transição de um estado para ele mesmo, por exemplo, seria a soma de duas setas contínuo-pontilhadas, cada uma delas conectando o estado com um dos arquétipos e depois de volta para o estado original. Seguindo esse raciocínio simples, fica fácil derivar duas cadeias de Markov—uma formada pelos estados originais e outra composta pelos arquétipos—simplesmente combinando todas as setas com o mesmo padrão cuja origem e destino coincidam.



(a) Matriz de desvio $\mathbf{D}$ (b) Matriz de retorno $\mathbf{K}$ (c) $\mathbf{P}^\pi = \mathbf{DK}$ e $\bar{\mathbf{P}}^\pi = \mathbf{KD}$

Figura 3.1: Representação das diferentes matrizes envolvidas na fatoração estocástica. Os círculos brancos correspondem aos estados originais do problema e os círculos pretos são os arquétipos. As setas contínuas são as transições no sentido estados → arquétipos e as setas pontilhadas representam as transições no sentido oposto. Por extensão, as transições do tipo estado → estado são representadas pela composição de uma seta contínua e uma pontilhada, e as do tipo arquétipo → arquétipo seguem o padrão contrário.

3.2.1 Interpretação geométrica

A redução de um processo de Markov através do desvio de transições baseia-se fortemente na fatoração estocástica

$$\mathbf{P}^\pi = \mathbf{DK}, \tag{3.17}$$

com $\mathbf{P}^\pi \in \mathbb{R}^{|S|\times|S|}$, $\mathbf{D} \in \mathbb{R}^{|S|\times m}$ e $\mathbf{K} \in \mathbb{R}^{m\times|S|}$. É fundamental, portanto, analisar cuidadosamente essa relação. Em primeiro lugar, é preciso dizer que a fatoração estocástica de $\mathbf{P}^\pi$ é sempre possível; uma solução trivial é fazer

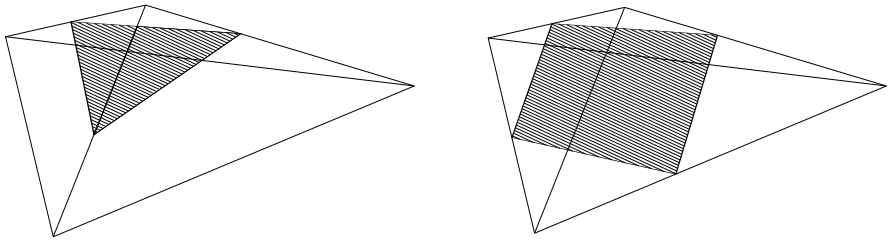
$$\mathbf{D} = \mathbf{P}^\pi \quad \text{e} \quad \mathbf{K} = \mathbf{I}, \tag{3.18}$$

onde $\mathbf{I}$ é a matriz-identidade de dimensão $|S|$ (e portanto uma matriz estocástica). Note que (3.17) significa que as linhas de $\mathbf{P}^\pi$ podem ser obtidas como combinações convexas das linhas de $\mathbf{K}$. Geometricamente, portanto, a solução (3.18) significa que todos os vetores estocásticos em $\mathbb{R}^{|S|}$ estão contidos no *simplex canônico* de dimensão $|S|-1$, ou seja, no simplex cujos vértices são a base canônica desse espaço. Por exemplo, qualquer vetor estocástico tridimensional pode ser representado como uma combinação convexa de $[0,0,1]$, $[0,1,0]$ e $[1,0,0]$, o que constitui um simplex de duas dimensões (ou um 2-simplex). Como os vetores na base canônica de $\mathbb{R}^{|S|}$ são eles mesmos estocásticos, fica claro que o conjunto formado por todos os vetores estocásticos $|S|$-dimensionais coincide com o simplex canônico de dimensão $|S|-1$. Denotarei esse conjunto por $\Delta^{|S|-1}$.

Os vetores-linha de $\mathbf{P}^\pi$ pertencem a $\Delta^{|S|-1}$, mas este não é necessariamente o simplex estocástico de menor dimensão que os contém. Pense no caso em que $\mathbf{P}^\pi$ é uma cadeia de Markov determinística, por exemplo. Nesse contexto o simplex definido pelos vetores $\mathbf{p}_i^\pi$ coincide com $\Delta^{|S|-1}$ apenas se $\mathbf{P}^\pi$ for uma matriz de permutação. O que determina a dimensão do menor simplex estocástico que contém as linhas de uma matriz de transições determinística é o número de estados distintos que recebem pelo menos uma transição, ou o número de colunas não-nulas em $\mathbf{P}^\pi$. Nesse caso a fatoração estocástica de $\mathbf{P}^\pi$ pode ser obtida simplesmente retendo em $\mathbf{K}$ as linhas distintas de $\mathbf{P}^\pi$ e definindo $\mathbf{D}$ de acordo: o elemento d_{ij} é 1 se e somente se a i-ésima linha de $\mathbf{P}^\pi$ corresponde à j-ésima linha de $\mathbf{K}$. Fica claro, portanto, que cadeias com uma arquitetura "em estrela"—em que um pequeno subconjunto de S recebe todas as transições—são mais suscetíveis à redução do que cadeias em forma de fila ou anel.

No caso de cadeias não-determinísticas a fatoração estocástica não tem uma interpretação tão trivial. Para ilustrar, suponha que $|S| = 4$. Nesse caso as linhas de $\mathbf{P}^\pi$ definem um tetraedro circunscrito no "tetraedro canônico" Δ^3. No entanto, se os vetores $\mathbf{p}_i^\pi$ forem colineares, eles pertencem a pelo menos um outro simplex estocástico: o segmento de reta cujos extremos são as interseções da reta que contém as linhas de $\mathbf{P}^\pi$ com as faces de Δ^3. Logo, os vetores-linha $\mathbf{p}_i^\pi$ podem ser representados como a combinação convexa de dois vetores estocásticos apenas, ou seja, existem matrizes estocásticas $\mathbf{D} \in \mathbb{R}^{4\times 2}$ e $\mathbf{K} \in \mathbb{R}^{2\times 4}$ tais que $\mathbf{DK} = \mathbf{P}^\pi$. É tentador generalizar esse raciocínio e dizer que o que determina o número m de arquétipos necessários para a fatoração estocástica de $\mathbf{P}^\pi$ é o número de vértices do envoltório convexo definido pelas suas linhas. Entretanto, esse nem sempre é o caso. Suponha que no exemplo anterior os vetores-linha de $\mathbf{P}^\pi$ não pertencessem à mesma reta, mas ainda fossem coplanares. Nesse caso, se o plano que contém as linhas de $\mathbf{P}^\pi$ separa um dos vértices de Δ^3 dos demais, então a interseção do plano com o tetraedro é um 2-simplex, como mostra a Figura 3.2a. Isso significa que é possível fatorar estocasticamente $\mathbf{P}^\pi$ com $m = 3$, mesmo que o envoltório convexo definido no plano pelos seus vetores-linha seja um quadrilátero. É natural supor então que o que determina m é a *dimensão* do envoltório convexo definido pelas linhas $\mathbf{p}_i^\pi$. Mais uma vez isso não é verdade. Considere que no cenário anterior o plano que contém os vetores $\mathbf{p}_i^\pi$ separe os vértices de Δ^3 dois a dois, como ilustra a Figura 3.2b. Pela figura fica óbvio que a inter-

seção do plano com o tetraedro não é um simplex, e portanto a fatoração estocástica de $\mathbf{P}^\pi$ pode não ser possível com apenas 3 arquétipos. Para enxergar isso, basta imaginar na Figura 3.2b que os quatro vetores-linha de $\mathbf{P}^\pi$ são as interseções do plano com as arestas do tetraedro Δ^3.



(a) A interseção é um simplex de dimensão 2 (b) A interseção não é um simplex

Figura 3.2: Distribuição dos vetores $\mathbf{p}_i^\pi$ no simplex Δ^3. A figura mostra as duas situações possíveis quando $|S| = 4$ e os vetores-linha de $\mathbf{P}^\pi$ são coplanares. No primeiro caso a interseção do plano que contém as linhas de $\mathbf{P}^\pi$ com o tetraedro Δ^3 é um triângulo (ou seja, um simplex de dimensão 2). No segundo caso a interseção não é um simplex.

O que determina o número mínimo de arquétipos necessários para a fatoração estocástica de uma matriz $\mathbf{P}^\pi$ é o número de vértices do menor simplex estocástico que contém as suas linhas. Chamarei esse número de *posto estocástico* de $\mathbf{P}^\pi$, e o denotarei por $\hat{\varrho}(\mathbf{P}^\pi)$. No caso de matrizes determinísticas o posto estocástico de $\mathbf{P}^\pi$ coincide com o seu posto convencional, que corresponde ao número de linhas distintas dessa matriz. No caso em que $\mathbf{P}^\pi$ é uma matriz estocástica qualquer essa relação não é tão clara. Chamando o posto de $\mathbf{P}^\pi$ de $\varrho(\mathbf{P}^\pi)$, pode-se dizer apenas que $\varrho(\mathbf{P}^\pi) \leq \hat{\varrho}(\mathbf{P}^\pi) \leq |S|$. Por definição, uma fatoração estocástica de $\mathbf{P}^\pi$ usando m arquétipos só é possível quando $m \geq \hat{\varrho}(\mathbf{P}^\pi)$.[3]

[3] A relação entre $\varrho(\mathbf{P}^\pi)$ e $\hat{\varrho}(\mathbf{P}^\pi)$ é uma questão interessante, e talvez seja possível vincular esses dois valores mais estreitamente. Eis algumas idéias preliminares nesse sentido: se $\varrho(\mathbf{P}^\pi) = n$, então existe uma matriz $\mathbf{A} \in \mathbb{R}^{|S| \times n}$ e uma matriz estocástica $\mathbf{B} \in \mathbb{R}^{n \times |S|}$ tais que $\mathbf{AB} = \mathbf{P}^\pi$. A partir de (3.13), pode-se afirmar que $\sum_j a_{ij} = 1$ para todo i. Então, se $\mathbf{A}$ é uma matriz não-negativa, ela é também estocástica. Portanto, uma possibilidade seria tentar obter a partir de $\mathbf{A}$ e $\mathbf{B}$ uma matriz não-negativa $\mathbf{D}$ e uma matriz estocástica $\mathbf{K}$ de forma que $\mathbf{DK} = \mathbf{P}^\pi$. Se isso for possível, então $\varrho(\mathbf{P}^\pi) = \hat{\varrho}(\mathbf{P}^\pi)$. Infelizmente não consegui demonstrar essa relação, e tampouco fui capaz de encontrar uma exceção (ou seja, um exemplo em que $\hat{\varrho}(\mathbf{P}^\pi) > \varrho(\mathbf{P}^\pi)$).

3.3 Redução de um Processo de Markov

A fatoração estocástica da matriz de transições $\mathbf{P}^\pi$ com $m < |S|$ depende de uma distribuição não trivial dos vetores-linha dessa matriz. Mesmo que esse seja o caso, para aplicar a Proposição 3.1 é necessário encontrar um vetor $\bar{\mathbf{r}}^\pi \in \mathbb{R}^m$ tal que $\mathbf{D}\bar{\mathbf{r}}^\pi = \mathbf{r}^\pi$, o que nem sempre é possível. Essas duas condições são muito restritivas, o que acaba por limitar a aplicabilidade da fatoração estocástica a um pequeno conjunto de processos de Markov. É natural portanto pensar em relaxá-las. Nesse caso, o problema passaria a ser encontrar matrizes estocásticas $\mathbf{D}$ e $\mathbf{K}$ e um vetor $\bar{\mathbf{r}}^\pi$ tais que

$$\mathbf{DK} \approx \mathbf{P}^\pi, \tag{3.19}$$

$$\mathbf{D}\bar{\mathbf{r}}^\pi \approx \mathbf{r}^\pi. \tag{3.20}$$

Note que a restrição de estocasticidade sobre $\mathbf{D}$ e $\mathbf{K}$ continua valendo—é ela que garante que $\mathbf{KD}$ seja uma cadeia de Markov válida—e por isso a expressão (3.19) pode ser chamada de *fatoração estocástica aproximada* de $\mathbf{P}^\pi$. Na seqüência do texto, usarei indiscriminadamente o termo "fatoração estocástica" para me referir tanto à versão exata quanto à versão aproximada dessa fatoração. Nos casos em que a distinção for fundamental e ela não estiver clara pelo contexto usarei os termos apropriados.

O significado exato das expressões (3.19) e (3.20) depende das funções de custo adotadas como medida de dissimilaridade entre as soluções encontradas e o objetivo. Pode-se, por exemplo, utilizar alguma noção de distância definida nos espaços métricos correspondentes [80]. Uma outra possibilidade é adotar uma medida mais específica para o problema. Quando se compara vetores estocásticos, como em (3.19), uma escolha comum é a divergência generalizada de Kullback-Leibler [81]. Qualquer que seja a função de custo escolhida para avaliar a qualidade das aproximações, a substituição de (3.10) e (3.11) pelas expressões acima tem uma conseqüência inevitável: a relação (3.12) também deixa de ser atendida de maneira estrita, ou seja, agora

$$\mathbf{v}^\pi \approx \mathbf{D}\bar{\mathbf{v}}^\pi.$$

Isso não é necessariamente um problema. Como discutido anteriormente, na programação dinâmica o cálculo da função de valor de uma política π não precisa ser exato. Contanto

que a política derivada da função de valor $\mathbf{v}^\pi$ seja a mesma gerada pela sua aproximação $\mathbf{D}\bar{\mathbf{v}}^\pi$, o uso da primeira ou da segunda é irrelevante do ponto de vista da aprendizagem.

3.3.1 Análise teórica

Exigir que a política gerada pela aproximação de $\mathbf{v}^\pi$ seja a mesma gerada pela função original ainda é muito restritivo. Mais interessante seria avaliar o impacto de uma aproximação qualquer sobre o processo de aprendizagem. Existem diversos resultados na literatura que permitem uma estimativa desse impacto [20, 186, 152, 22]. Na seção 3.4 um desses resultados é discutido em detalhes, mas por enquanto basta saber que ele se baseia em

$$\| \mathbf{v}^\pi - \tilde{\mathbf{v}}^\pi \|_\infty, \tag{3.21}$$

onde $\tilde{\mathbf{v}}^\pi = \mathbf{D}\bar{\mathbf{v}}^\pi$. Obviamente, quando se utiliza a fatoração estocástica para reduzir um processo de Markov não se conhece a função $\mathbf{v}^\pi$. Portanto, para avaliar o impacto da fatoração sobre a aprendizagem é necessário limitar (3.21) com base nos erros de aproximação de $\mathbf{P}^\pi$ e $\mathbf{r}^\pi$. Especificamente, o que procuro nesta seção é encontrar uma cota superior para (3.21) baseada em

$$\| \mathbf{P}^\pi - \tilde{\mathbf{P}}^\pi \|_\infty \quad \text{e} \quad \| \mathbf{r}^\pi - \tilde{\mathbf{r}}^\pi \|_\infty,$$

onde $\tilde{\mathbf{P}}^\pi = \mathbf{DK}$ e $\tilde{\mathbf{r}}^\pi = \mathbf{D}\bar{\mathbf{r}}^\pi$. Para tal, farei uso do fato de $\mathbf{v}^\pi$ e $\tilde{\mathbf{v}}^\pi$ serem soluções de sistemas de equações lineares. Inicio apresentando um teorema de Golub e Van Loan [50] que trata justamente dessa situação:

Teorema 3.1. *Seja* $\mathbf{A}$ *inversível e* $\mathbf{x}$ *a solução de* $\mathbf{Ax} = \mathbf{b}$. *Seja* $\tilde{\mathbf{A}}$ *uma aproximação da matriz de coeficientes* $\mathbf{A}$ *e* $\tilde{\mathbf{b}}$ *uma aproximação do vetor* $\mathbf{b}$ *tais que*

$$\frac{\| \mathbf{A} - \tilde{\mathbf{A}} \|}{\| \mathbf{A} \|} \leq \epsilon \quad e \quad \frac{\| \mathbf{b} - \tilde{\mathbf{b}} \|}{\| \mathbf{b} \|} \leq \epsilon.$$

Se $\epsilon < 1/\mathrm{cond}(\mathbf{A})$, *onde* $\mathrm{cond}(\mathbf{A}) = \| \mathbf{A} \| \| \mathbf{A}^{-1} \|$, *então a matriz* $\tilde{\mathbf{A}}$ *é inversível e* $\tilde{\mathbf{x}} = \tilde{\mathbf{A}}^{-1}\tilde{\mathbf{b}}$ *atende a*

$$\frac{\| \mathbf{x} - \tilde{\mathbf{x}} \|}{\| \mathbf{x} \|} \leq \frac{2\epsilon}{1 - \mathrm{cond}(\mathbf{A})\epsilon} \mathrm{cond}(\mathbf{A}). \tag{3.22}$$

Demonstração. Golub e Van Loan [50], Teorema 2.7.2, p. 83. □

Note que o Teorema 3.1 vale para qualquer norma, contanto que a norma matricial seja aquela induzida pela norma vetorial adotada. No caso de $\| \cdot \|_\infty$, isso significa que

$$\| \mathbf{A} \|_\infty = \max_i \sum_j |a_{ij}|. \tag{3.23}$$

A norma acima é particularmente fácil de ser calculada para as matrizes envolvidas no problema de redução de um processo de Markov. Isso porque todas elas têm por definição vetores-linha com uma soma fixa, o que torna (3.23) uma operação trivial. A princípio, o que poderia dificultar a aplicação de (3.22) na prática é a necessidade de se determinar a inversa da matriz de coeficientes $\mathbf{A}$ (afinal de contas, o objetivo de se fatorar o processo de Markov é justamente evitar essa inversão). Felizmente, quando as linhas de uma matriz $\mathbf{A}$ têm todas a mesma soma não é preciso invertê-la para calcular o seu número de condição $\mathrm{cond}(\mathbf{A})$, como mostra o lema abaixo.

Lema 3.1. *Se todas as linhas de uma matriz inversível* $\mathbf{A}$ *têm soma* β*, então as linhas de sua inversa* $\mathbf{A}^{-1}$ *têm soma* $1/\beta$*.*

Demonstração. Para facilitar a notação, seja $\mathbf{B} = \mathbf{A}^{-1}$. Sabe-se que $\mathbf{BA} = \mathbf{I}$, o que implica que

$$\begin{aligned} 1 &= \textstyle\sum_j \sum_k b_{ik} a_{kj} \\ &= \textstyle\sum_k b_{ik} \sum_j a_{kj} \quad \text{para todo } i. \end{aligned}$$

Como $\sum_j a_{kj} = \beta$, tem-se $\sum_k b_{ik} = 1/\beta$ ($\beta \neq 0$ porque $\mathbf{A}$ é inversível). □

Note que do lema acima segue que, sob a norma $\| \cdot \|_\infty$, o número de condição de uma matriz cujas linhas têm a mesma soma é 1. Essa é a melhor situação possível, uma vez que $\mathrm{cond}(\mathbf{A}) \geq 1$ quando se adota essa norma (e, como se sabe, quanto maior $\mathrm{cond}(\mathbf{A})$ mais mal-condicionada é a matriz [50]). Além disso, a conclusão acima simplifica consideravelmente a expressão (3.22) para o caso estudado aqui, o que me permite enunciar a seguinte proposição:

Proposição 3.2. *Sejam* $\mathbf{P}^\pi \in \mathbb{R}^{|S|\times|S|}$ *e* $\mathbf{r}^\pi \in \mathbb{R}^{|S|}$ *a matriz de transições e o vetor de recompensas de um processo de Markov induzido por uma política* π*. Sejam* $\tilde{\mathbf{P}}^\pi \in \mathbb{R}^{|S|\times|S|}$ *uma matriz estocástica e* $\tilde{\mathbf{r}}^\pi \in \mathbb{R}^{|S|}$ *um vetor qualquer. Defina*

$$\epsilon = \max\left(\frac{\gamma}{1-\gamma} \| \mathbf{P}^\pi - \tilde{\mathbf{P}}^\pi \|_\infty, \frac{\| \mathbf{r}^\pi - \tilde{\mathbf{r}}^\pi \|_\infty}{\| \mathbf{r}^\pi \|_\infty} \right), \tag{3.24}$$

com $\gamma \in [0, 1)$. *Se* $\epsilon < 1$, *então*

$$\frac{\| \mathbf{v}^\pi - \tilde{\mathbf{v}}^\pi \|_\infty}{\| \mathbf{v}^\pi \|_\infty} \leq \frac{2\epsilon}{1-\epsilon}, \tag{3.25}$$

onde $\mathbf{v}^\pi$ *é a função de valor do processo de Markov original e* $\tilde{\mathbf{v}}^\pi$ *é a função de valor do processo definido por* $\tilde{\mathbf{P}}^\pi$ *e* $\tilde{\mathbf{r}}^\pi$, *ambas calculadas com um fator de desconto* γ.

Demonstração. Sabe-se que

$$(\mathbf{I} - \gamma \mathbf{P}^\pi)\mathbf{v}^\pi = \mathbf{r}^\pi,$$

e, como $\tilde{\mathbf{P}}^\pi$ é uma matriz quadrada estocástica, sabe-se que

$$(\mathbf{I} - \gamma \tilde{\mathbf{P}}^\pi)\tilde{\mathbf{v}}^\pi = \tilde{\mathbf{r}}^\pi \tag{3.26}$$

tem uma solução única para $0 \leq \gamma < 1$ [128, 22]. Tem-se, portanto, dois sistemas lineares com o mesmo número de equações e incógnitas. Para limitar a diferença entre as suas respectivas soluções recorro ao Teorema 3.1. Nesse caso tem-se

$$\begin{aligned} \mathbf{A} &= (\mathbf{I} - \gamma \mathbf{P}^\pi), \quad \mathbf{b} = \mathbf{r}^\pi, \quad \mathbf{x} = \mathbf{v}^\pi, \\ \tilde{\mathbf{A}} &= (\mathbf{I} - \gamma \tilde{\mathbf{P}}^\pi), \quad \tilde{\mathbf{b}} = \tilde{\mathbf{r}}^\pi, \quad \tilde{\mathbf{x}} = \tilde{\mathbf{v}}^\pi. \end{aligned} \tag{3.27}$$

Como $\mathbf{P}^\pi$ é uma matriz estocástica, pode-se escrever para toda linha i de $\mathbf{A}$:

$$\begin{aligned} \textstyle\sum_j a_{ij} &= 1 - \gamma p_{ii}^\pi - \gamma \textstyle\sum_{j \neq i} p_{ij}^\pi \\ &= 1 - \gamma p_{ii}^\pi - \gamma(1 - p_{ii}^\pi) \\ &= 1 - \gamma. \end{aligned}$$

Logo,

$$\| \mathbf{A} \|_\infty = 1 - \gamma, \quad \| \mathbf{A}^{-1} \|_\infty = \tfrac{1}{1-\gamma} \quad \text{e} \quad \text{cond}(\mathbf{A}) = 1, \tag{3.28}$$

em que o segundo resultado é uma aplicação direta do Lema 3.1. Substituindo (3.27) e (3.28) em (3.22), após algumas manipulações algébricas simples chega-se à expressão (3.25). □

A expressão (3.25) fornece uma cota superior para o erro relativo na aproximação de $\mathbf{v}^\pi$. Para derivar um limite para o erro absoluto a partir dessa expressão basta encontrar uma cota superior para $\| \mathbf{v}^\pi \|_\infty$. Na falta de informação adicional sobre o problema, pode-se usar o fato de

$$\| \mathbf{v}^\pi \|_\infty = \| (\mathbf{I} - \gamma \mathbf{P}^\pi)^{-1} \mathbf{r}^\pi \|_\infty \leq \frac{1}{1-\gamma} \| \mathbf{r}^\pi \|_\infty, \tag{3.29}$$

que, substituindo em (3.25), leva a

$$\| \mathbf{v}^\pi - \tilde{\mathbf{v}}^\pi \|_\infty \leq \frac{2\epsilon}{(1-\epsilon)(1-\gamma)} \| \mathbf{r}^\pi \|_\infty, \quad \epsilon < 1. \tag{3.30}$$

Note, no entanto, que (3.25) é mais justa do que (3.30).

A condição $\epsilon < 1$ no enunciado da Proposição 3.2 implica que

$$\| \mathbf{P}^\pi - \tilde{\mathbf{P}}^\pi \|_\infty < \frac{1-\gamma}{\gamma},$$

o que pode significar uma restrição forte quando o fator de desconto γ se aproxima de 1. Por exemplo, um desconto de $\gamma = 0.9$ implica que a cota (3.25) só pode ser usada se $\sum_j |p_{ij}^\pi - \tilde{p}_{ij}^\pi| < 0.12$ para todo i. Isso claramente limita o número de casos em que a proposição pode ser utilizada. Sendo assim, apresento abaixo uma maneira alternativa de se calcular uma cota superior para $\| \mathbf{v}^\pi - \tilde{\mathbf{v}}^\pi \|_\infty$ que não impõe nenhuma restrição sobre a qualidade das aproximações de $\mathbf{P}^\pi$ ou $\mathbf{r}^\pi$.

Proposição 3.3. *Sejam $\mathbf{P}^\pi$, $\mathbf{r}^\pi$, $\tilde{\mathbf{P}}^\pi$ e $\tilde{\mathbf{r}}^\pi$ como na Proposição 3.2. Então*

$$\| \mathbf{v}^\pi - \tilde{\mathbf{v}}^\pi \|_\infty \leq \frac{1}{1-\gamma} \left(\| \mathbf{r}^\pi - \tilde{\mathbf{r}}^\pi \|_\infty + \frac{\gamma}{1-\gamma} \| \mathbf{P}^\pi - \tilde{\mathbf{P}}^\pi \|_\infty \| \mathbf{r}^\pi \|_\infty \right). \tag{3.31}$$

Demonstração. Sejam $\mathbf{A}$ e $\tilde{\mathbf{A}}$ como em (3.27) e defina

$$\Delta\mathbf{A} = \tilde{\mathbf{A}} - \mathbf{A}, \quad \Delta\mathbf{r}^\pi = \tilde{\mathbf{r}}^\pi - \mathbf{r}^\pi \quad \text{e} \quad \Delta\mathbf{v}^\pi = \tilde{\mathbf{v}}^\pi - \mathbf{v}^\pi.$$

Então

$$\begin{aligned} \mathbf{r}^\pi + \Delta\mathbf{r}^\pi &= (\mathbf{A} + \Delta\mathbf{A})(\mathbf{v}^\pi + \Delta\mathbf{v}^\pi) \\ &= \mathbf{A}\mathbf{v}^\pi + \mathbf{A}\Delta\mathbf{v}^\pi + \Delta\mathbf{A}\mathbf{v}^\pi + \Delta\mathbf{A}\Delta\mathbf{v}^\pi. \end{aligned}$$

Lembrando que $\mathbf{r}^\pi = \mathbf{A}\mathbf{v}^\pi$, após algumas manipulações algébricas tem-se

$$\Delta\mathbf{v}^\pi = \tilde{\mathbf{A}}^{-1}(\Delta\mathbf{r}^\pi - \Delta\mathbf{A}\mathbf{v}^\pi),$$

o que implica que

$$\begin{aligned} \| \mathbf{v}^\pi - \tilde{\mathbf{v}}^\pi \|_\infty &\leq \| \tilde{\mathbf{A}}^{-1} \|_\infty \| \Delta\mathbf{r}^\pi - \Delta\mathbf{A}\mathbf{v}^\pi \|_\infty \\ &\leq \| \tilde{\mathbf{A}}^{-1} \|_\infty (\| \Delta\mathbf{r}^\pi \|_\infty + \| \Delta\mathbf{A} \|_\infty \| \mathbf{v}^\pi \|_\infty). \end{aligned} \tag{3.32}$$

Como $\tilde{\mathbf{P}}^\pi$ é uma matriz estocástica, basta usar o Lema 3.1 para concluir que

$$\| \tilde{\mathbf{A}}^{-1} \|_\infty = \frac{1}{1-\gamma}, \tag{3.33}$$

como feito na demonstração da Proposição 3.2. Substituindo (3.33) e (3.29) em (3.32), chega-se a (3.31). □

A cota (3.31) é mais justa do que (3.30). Para enxergar isso basta lembrar a definição de ϵ em (3.24) e notar o seguinte:

$$\frac{2\epsilon\| \mathbf{r}^\pi \|_\infty}{(1-\epsilon)(1-\gamma)} \geq \frac{1}{(1-\epsilon)(1-\gamma)} \left(\| \mathbf{r}^\pi - \tilde{\mathbf{r}}^\pi \|_\infty + \frac{\gamma}{1-\gamma} \| \mathbf{P}^\pi - \tilde{\mathbf{P}}^\pi \|_\infty \| \mathbf{r}^\pi \|_\infty \right).$$

Comparando o lado direito da expressão acima com a cota superior em (3.31) fica claro que (3.30) é pelo menos $(1-\epsilon)^{-1}$ vezes maior do que esta última. Isso não significa que a Proposição 3.2 não tenha importância. Quando se deseja uma estimativa do *erro relativo* na aproximação de $\mathbf{v}^\pi$, a cota (3.25) pode ser mais justa do que (3.31), dependendo do valor das variáveis envolvidas nessas expressões. Nesse caso, uma alternativa seria calcular o valor de ambas as cotas e adotar a menor delas como teto para o erro de aproximação.

Dadas aproximações $\tilde{\mathbf{P}}^\pi$ e $\tilde{\mathbf{r}}^\pi$, o lado direito das expressões (3.25) e (3.31) pode ser facilmente calculado. No caso da fatoração estocástica, isso significa que basta fazer $\tilde{\mathbf{P}}^\pi = \mathbf{DK}$ e $\tilde{\mathbf{r}}^\pi = \mathbf{D}\bar{\mathbf{r}}^\pi$ para se ter uma estimativa do erro introduzido no cálculo de $\mathbf{v}^\pi$ pela redução do processo de Markov, ou seja, uma cota superior para $\| \mathbf{v}^\pi - \mathbf{D}\bar{\mathbf{v}}^\pi \|_\infty$. Como as expressões (3.25) e (3.31) podem ser calculadas eficientemente, pode-se pensar em adotá-las na prática como um critério que garanta a qualidade da aproximação de $\mathbf{v}^\pi$. Por exemplo, em princípio é possível definir uma tolerância máxima para (3.21) e usar (3.31) como critério de parada para um algoritmo iterativo que refine sucessivamente as aproximações de $\mathbf{P}^\pi$ e $\mathbf{r}^\pi$ (na próxima seção apresento um algoritmo desse tipo). É concebível também usar (3.25) ou (3.31) para determinar o número mínimo de arquétipos necessários para se alcançar um determinado grau de precisão na aproximação de $\mathbf{v}^\pi$. Note, no entanto, que (3.25) e (3.31) são limites teóricos, que em alguns casos representam apenas uma superestimativa grosseira do erro real de aproximação. Na Seção 3.3.2 eu analiso as situações em que esses limites constituem valores com alguma relevância prática.

Independentemente do seu uso na prática, a definição de uma cota superior para o erro na aproximação da função de valor é de extrema importância para embasar teoricamente a fatoração estocástica aproximada. Isso porque as expressões para o cálculo dessa cota deixam claro quais são as variáveis que influenciam a qualidade da aproximação. Além disso, essas expressões fornecem uma indicação do comportamento do erro de aproximação em relação aos diferentes aspectos do problema. Por exemplo, observando (3.25)

e (3.31) fica claro que a tendência do erro é crescer com um aumento do fator de desconto γ. Na próxima seção essa e outras questões são investigadas empiricamente em uma uma série de experimentos computacionais.

3.3.2 Análise empírica

Na seção anterior discuti de um ponto de vista teórico as conseqüências de se usar aproximações da matriz de transições e do vetor de recompensas no cálculo da função de valor de um processo de Markov. Nesta seção abordo o mesmo assunto sob uma perspectiva mais prática, através de uma série de experimentos computacionais em que analiso as peculiaridades do problema de aproximação em questão. Em particular, avalio como as características da matriz de transições $\mathbf{P}^{\pi}$ e do vetor-recompensa $\mathbf{r}^{\pi}$ influenciam na qualidade da aproximação de um processo de Markov, e como esta última se reflete no cálculo da função de valor $\tilde{\mathbf{v}}^{\pi}$.

Para a análise empírica desta seção interpreto a fatoração estocástica de $\mathbf{P}^{\pi}$ como um tipo especial de fatoração de uma matriz em que todos os elementos são maiores ou iguais a zero. Além de ser natural, essa formulação do problema torna disponível uma série de métodos desenvolvidos especificamente para resolvê-lo. Note portanto que a escolha dessa abordagem é puramente uma questão de conveniência, e não significa que eu esteja sugerindo que esta seja a melhor alternativa para lidar com o problema. Como será visto à frente, existem maneiras mais eficientes de se fatorar um processo de Markov. No início desta seção, porém, a ferramenta fica em segundo plano, e o foco da análise são as características do problema de aproximação.

Fatoração não-negativa de uma matriz

Vou me concentrar por um instante na fatoração estocástica de $\mathbf{P}^{\pi}$ isoladamente. Suponha que o objetivo seja minimizar os resíduos quadrados deixados pela diferença $\mathbf{P}^{\pi} - \mathbf{DK}$. Então, o problema pode ser escrito de forma sucinta como:

$$\underset{\mathbf{D},\mathbf{K}\geq 0}{\text{minimizar}}\, \xi(\mathbf{P}^{\pi}, \mathbf{DK}) = ||\mathbf{P}^{\pi} - \mathbf{DK}||_{\mathrm{F}}^{2} \tag{3.34}$$

em relação à $\mathbf{D}$ e $\mathbf{K}$ sujeito a

$$\sum_{j=1}^{m} d_{ij} = 1 \quad \text{para } i = 1, 2, ...|S|, \tag{3.35}$$

$$\sum_{j=1}^{|S|} k_{ij} = 1 \quad \text{para } i = 1, 2, ...m, \tag{3.36}$$

onde $||\cdot||_{\mathrm{F}}$ é a norma de Frobenius [50].

A expressão (3.34) é um problema conhecido na literatura como *fatoração não-negativa de uma matriz*. Embora a sua versão exata já venha sendo estudado há algum tempo pela álgebra linear [94, 32, 120], foi apenas recentemente que a versão aproximada do problema passou a ser considerada como uma abordagem para a análise de dados [114, 87]. A vantagem da fatoração não-negativa em relação a técnicas de redução de dimensionalidade convencionais, como a análise de componentes principais, é a possibilidade de interpretação dos vetores-base em aplicações em que valores negativos não encontram correspondência no mundo real. Por exemplo, ao se fatorar uma matriz representando uma imagem em escalas de cinza, valores não-negativos podem ser interpretados como as tonalidades dos *pixels*. Dessa forma, um objeto pode ser decomposto em suas "partes constituintes." Uma boa ilustração desse conceito é dada por Lee e Seung [87], que utilizam a fatoração não-negativa para decompor imagens faciais em caracteres identificáveis como olhos, bocas, *etc*.

A origem do estudo da fatoração não-negativa aproximada é em geral atribuída à Paatero [114, 112, 113], mas Lee e Seung foram provavelmente os grandes responsáveis pela sua popularização, ao interpretarem a técnica como uma forma de análise de dados genérica [87] e apresentarem algoritmos simples para resolver o problema [88]. Embora seja um assunto relativamente novo, a fatoração não-negativa já é tema de uma enorme quantidade de trabalhos, que discutem desde aspectos teóricos do problema de otimização [41] até aplicações práticas do método [87, 189], passando por questões técnicas de implementação [90, 53]. Uma tentativa de revisão exaustiva da literatura seria portanto uma empreitada fadada ao fracasso, e neste contexto faria pouco mais do que desviar o

leitor da discussão principal. Assim, limito-me a citar algumas referências relevantes no fluxo do texto. Para uma pesquisa mais extensiva sobre o assunto, sugiro a leitura do trabalho de Sra e Dhillon [153], que contém uma revisão histórica e inúmeras referências para trabalhos sobre o tema.

Embora o problema (3.34) seja convexo em relação a $\mathbf{D}$ e a $\mathbf{K}$ isoladamente, ele não o é em relação às duas variáveis conjuntamente [88, 90, 62]. Assim, o máximo que um algoritmo de fatoração não-negativa pode garantir é um mínimo local para o problema. Os diferentes métodos existentes na literatura para a solução de (3.34) variam bastante em relação à técnica de otimização adotada, mas em geral eles seguem a filosofia de uma *minimização alternada*: primeiro fixa-se $\mathbf{K}$ e otimiza-se $\mathbf{D}$, e depois fixa-se $\mathbf{D}$ e otimiza-se $\mathbf{K}$. Nesta seção apresento o algoritmo de Lee e Seung [88], que é sem dúvida o representante mais popular dessa categoria. É importante observar de antemão que existem na literatura diversas propostas de algoritmos que supostamente funcionariam melhor do que este que será apresentado [53, 90, 76]. No entanto, esses algoritmos são em geral mais complicados e dependem da definição certa de seus parâmetros (em contraste com o algoritmo de Lee e Seung, que não depende de nenhum parâmetro). Como a ênfase aqui não está no desempenho do método de aproximação, o ganho eventual na precisão das soluções encontradas dificilmente justificaria o trabalho extra em apresentar e implementar esses algoritmos mais avançados.

O algoritmo de Lee e Seung pode ser visto como uma descida pelo gradiente em que cada elemento do vetor de atualização é escalado individualmente. Note que fixando $\mathbf{D}$ ou $\mathbf{K}$ a expressão (3.34) torna-se uma minimização dos mínimos quadrados linear, cujos gradientes são dados por

$$\frac{\partial \xi}{\partial \mathbf{D}} = \mathbf{P}^{\pi}\mathbf{K}^{\mathrm{t}} - \mathbf{D}\mathbf{K}\mathbf{K}^{\mathrm{t}}, \tag{3.37}$$

$$\frac{\partial \xi}{\partial \mathbf{K}} = \mathbf{D}^{\mathrm{t}}\mathbf{P}^{\pi} - \mathbf{D}^{\mathrm{t}}\mathbf{D}\mathbf{K}, \tag{3.38}$$

onde $\mathbf{K}^{\mathrm{t}}$ é a matriz transposta de $\mathbf{K}$. No algoritmo de Lee e Seung os elementos das matrizes acima são reescalados de maneira independente, ou seja, as equações de atualização são

$$\mathbf{D} \leftarrow \mathbf{D} - \mathbf{E} \times \frac{\partial \xi}{\partial \mathbf{D}},$$

$$\mathbf{K} \leftarrow \mathbf{K} - \mathbf{L} \times \frac{\partial \xi}{\partial \mathbf{K}},$$

em que "$\times$" denota o produto de Hadamard (elemento-por-elemento). As matrizes $\mathbf{E}$ e $\mathbf{L}$ são definidas de forma a aumentar o decréscimo da função de custo ξ a cada iteração, sem no entanto violar a restrição de não-negatividade. Especificamente, Lee e Seung [88] mostram que fazendo

$$\mathbf{E} = \frac{\mathbf{D}}{\mathbf{DKK}^{\mathrm{t}}},$$
$$\mathbf{L} = \frac{\mathbf{K}}{\mathbf{D}^{\mathrm{t}}\mathbf{DK}},$$

a função de custo (3.34) decresce monotonicamente a cada iteração (a divisão de matrizes aqui também é elemento-por-elemento). Assim, as equações de atualização do algoritmo ficam reduzidas a duas regras multiplicativas simples:

$$\mathbf{D} \leftarrow \mathbf{D} \times \frac{\mathbf{P}^{\pi}\mathbf{K}^{\mathrm{t}}}{\mathbf{DKK}^{\mathrm{t}}}, \tag{3.39}$$

$$\mathbf{K} \leftarrow \mathbf{K} \times \frac{\mathbf{D}^{\mathrm{t}}\mathbf{P}^{\pi}}{\mathbf{D}^{\mathrm{t}}\mathbf{DK}}. \tag{3.40}$$

Observe que o algoritmo acima foi desenvolvido para a versão original do problema de fatoração não-negativa de uma matriz. Aqui, no entanto, só interessa o caso em que $\mathbf{P}^{\pi}$ é aproximada por duas matrizes estocásticas, como determinam as restrições (3.35) e (3.36). Uma maneira de fatorar estocasticamente $\mathbf{P}^{\pi}$ seria simplesmente adicionar um termo de penalização a (3.34) que forçasse a soma dos elementos em uma linha de $\mathbf{D}$ e $\mathbf{K}$ a ser um. Essa possibilidade já foi levantada na literatura relacionada à fatoração não-negativa de matrizes [76, 153] e aparece em outros contextos como uma solução para problemas com restrições de convexidade muito parecidas com (3.35) e (3.36) [36, 86]. No entanto, neste trabalho adoto uma abordagem diferente. Como notado anteriormente, o algoritmo de Lee e Seung nada mais é do que um tipo especial de descida pelo gradiente. Uma das maneiras de lidar com problemas de otimização com restrições é utilizar o método da projeção do gradiente, em que as soluções intermediárias são atualizadas da maneira convencional e depois projetadas de volta no espaço factível [17]. Aqui os espaços factíveis referentes às linhas de $\mathbf{D}$ e de $\mathbf{K}$ são os simplexes Δ^{m-1} e $\Delta^{|S|-1}$, respectivamente. Portanto, pode-se após cada aplicação de (3.39) e (3.40) projetar as linhas de $\mathbf{D}$ e $\mathbf{K}$ nesses espaços. Note que essa projeção é um pouco mais complicada do que simplesmente normalizar os vetores de maneira que os seus elementos tenham soma um. Para fazê-lo adotei um engenhoso algoritmo apresentado por Nascimento *et al.* [108],

cujos detalhes serão omitidos em nome do fluxo da apresentação. Por ora basta saber que dada uma matriz qualquer $\mathbf{A}$ com linhas em $\mathbb{R}^n$, a função $\varphi(\mathbf{A})$ retorna a matriz estocástica formada pela projeção ortogonal dos vetores-linha de $\mathbf{A}$ em Δ^{n-1}.[4]

O Algoritmo 3.1 traz um pseudo-código descrevendo o método utilizado nesta seção para realizar a fatoração estocástica de $\mathbf{P}^\pi$. Note que o algoritmo é simplesmente o método de Lee e Seung seguido da projeção de Nascimento *et al.* É importante observar, porém, que com essa pequena alteração o algoritmo de Lee e Seung perde a garantia de decréscimo da função de custo (3.34) a cada iteração. Nos casos em que um acréscimo eventual de (3.34) é intolerável, pode-se garantir o seu decréscimo monotônico projetando os gradientes originais (3.37) e (3.38) ao invés de suas versões reescaladas, (3.39) e (3.40) [17, 90, 108] . Note, porém, que essa decisão envolve a definição de uma taxa de aprendizagem adequada que garanta a monotonicidade a cada iteração.

Algoritmo 3.1 Fatoração estocástica aproximada de uma matriz

Requer $\mathbf{P}^\pi \in \mathbb{R}^{|S|\times|S|}$, $m \in [1, |S|]$

Retorna $\mathbf{D} \in \mathbb{R}^{|S|\times m}$ e $\mathbf{K} \in \mathbb{R}^{m\times|S|}$ estocásticas tais que $\mathbf{DK} \approx \mathbf{P}^\pi$

$\mathbf{D} \leftarrow$ matriz estocástica $|S| \times m$ aleatória

$\mathbf{K} \leftarrow$ matriz estocástica $m \times |S|$ aleatória

repita

$$\mathbf{D} \leftarrow \varphi\left(\mathbf{D} \times \frac{\mathbf{P}^\pi\mathbf{K}^\intercal}{\mathbf{DKK}^\intercal}\right) \qquad \triangleright \textit{ Atualização (3.39) seguida de projeção}$$

$$\mathbf{K} \leftarrow \varphi\left(\mathbf{K} \times \frac{\mathbf{D}^\intercal\mathbf{P}^\pi}{\mathbf{D}^\intercal\mathbf{DK}}\right) \qquad \triangleright \textit{ Atualização (3.40) seguida de projeção}$$

até critério de parada satisfeito

Antes de encerrar esta seção, vale fazer uma observação. A estrutura do Algoritmo 3.1 pode dar a falsa impressão de que existe uma solução trivial para o problema de otimização alternada: ao invés de atualizar as matrizes $\mathbf{D}$ e $\mathbf{K}$ através das equações (3.39) e (3.40), por que não encontrar a solução exata para cada sub-problema antes de fazer a

[4] Uma solução alternativa para lidar com a restrição de estocasticidade de $\mathbf{D}$ e $\mathbf{K}$ seria adotar como função de custo a divergência generalizada de Kullback-Leibler [88]. O que torna essa função atraente para a aproximação de $\mathbf{P}^\pi$ é o fato de todos os seus pontos estacionários serem também matrizes estocásticas. Além disso, dado um ponto estacionário $\mathbf{P}^\pi \approx \mathbf{BC}$, pode-se facilmente obter a partir de $\mathbf{B}$ e $\mathbf{C}$ matrizes $\mathbf{D}$ e $\mathbf{K}$ que são linha-estocásticas, como mostra o Corolário 2 de Ho e Dooren [62]. Embora essa seja uma maneira elegante de lidar com as restrições (3.35) e (3.36), preferi me ater à função de custo (3.34), porque a partir dela pode-se facilmente desenvolver heurísticas com um custo computacional muito inferior ao do Algoritmo 3.1, como será visto à frente.

projeção? Nesse caso, a atualização de $\mathbf{D}$ seria

$$\mathbf{D}^{\mathrm{t}} \leftarrow \varphi\left(\left(\mathbf{K}\mathbf{K}^{\mathrm{t}}\right)^{-1}\mathbf{K}\mathbf{P}^{\pi\,\mathrm{t}}\right), \tag{3.41}$$

e a atualização de $\mathbf{K}$ poderia ser derivada de maneira análoga. Embora a princípio essa abordagem pareça mais bem fundamentada teoricamente, ocorre justamente o contrário. O algoritmo acima usado para a fatoração não-negativa pode divergir até mesmo no caso em que não se impõe as restrições de estocasticidade sobre $\mathbf{D}$ e $\mathbf{K}$ [76].

Aproximação da matriz de transições

Existem sempre dois tipos de erro envolvidos na aproximação de uma matriz $\mathbf{P}^{\pi}$ que representa as transições de um processo de Markov induzido por uma política π. Um se refere às características da própria matriz, e é inerente ao processo. O outro representa as limitações do algoritmo adotado para o problema de aproximação. Como discutido na Seção 3.2.1, o que determina o número mínimo de arquétipos necessários para a fatoração estocástica de uma matriz $\mathbf{P}^{\pi}$ é a dimensão do menor simplex estocástico que contém as suas linhas. Se $\hat{\varrho}(\mathbf{P}^{\pi}) = n$, existe um erro intrínseco na aproximação $\mathbf{DK} \approx \mathbf{P}^{\pi}$ com $m < n$ que independe do método de aproximação utilizado. Por outro lado, mesmo que uma fatoração exata seja possível é provável que a função de custo (3.34) nunca seja zerada na prática, por questões inerentes ao método adotado para a aproximação. Por exemplo, o resíduo na função de custo pode refletir o fato de a solução encontrada ser apenas um mínimo local para o problema. É fundamental que se distingua esses dois tipos de erro, porque eles são indicativos de duas limitações diferentes da proposta deste trabalho. Enquanto o primeiro representa um limite teórico, o segundo é uma medida da imperfeição da ferramenta usada para implementá-la. Nesta seção me concentro na análise do erro do primeiro tipo. As dificuldades de se resolver o problema de aproximação na prática são tratadas mais à frente.

De um ponto de vista computacional, o posto estocástico de uma matriz não é o único fator relevante para a sua fatoração estocástica. Quando se adota um método como o Algoritmo 3.1 para fatorar estocasticamente $\mathbf{P}^{\pi}$, existem outras características dessa matriz que podem facilitar ou dificultar o problema. Suponha que $\hat{\varrho}(\mathbf{P}^{\pi}_1) = \hat{\varrho}(\mathbf{P}^{\pi}_2) = m$. O

fato de uma fatoração usando m arquétipos ser possível em ambos os casos não significa que os dois problemas tenham o mesmo nível de dificuldade. Em particular, se o $(m-1)$-simplex definido pelas linhas de $\mathbf{P}_1^\pi$ está contido naquele definido pelas linhas de $\mathbf{P}_2^\pi$, então o conjunto dos simplexes que representam esta última de maneira exata está contido no conjunto dos simplexes que representam a primeira. Em outras palavras: uma matriz $\mathbf{K}$ que seja uma solução para a fatoração estocástica de $\mathbf{P}_2^\pi$ é também uma solução para a fatoração de $\mathbf{P}_1^\pi$, mas a recíproca não é necessariamente verdadeira—$\mathbf{P}_1^\pi$ é "mais fatorizável" do que $\mathbf{P}_2^\pi$.

Intuitivamente, a dificuldade em se fatorar estocasticamente uma matriz está relacionada com o hiper-volume do envoltório convexo definido pelas suas linhas. Em geral, quanto menor esse volume mais fácil é a fatoração. Para visualizar essa questão, suponha que $\mathbf{P}^\pi$ seja a matriz-identidade de dimensão $|S|$. Nesse caso os vetores-linha dessa matriz são coincidentes com os vértices do simplex $\Delta^{|S|-1}$, e uma fatoração exata só é possível quando se utiliza os próprios vértices desse simplex (o que equivale a dizer que $\mathbf{K} = \mathbf{L}\mathbf{P}^\pi$, onde $\mathbf{L}$ é uma matriz de permutação). Imagine agora que as linhas da matriz $\mathbf{P}^\pi$ tivessem um alto grau de estocasticidade—ou seja, que a diferença de magnitude dos seus elementos não fosse muito grande. Nesse caso, os vetores-linha estariam na "região central" de $\Delta^{|S|-1}$, o que resultaria em uma diminuição do hiper-volume definido pelo seu envoltório convexo. Isso claramente aumentaria as possibilidades de fatoração da matriz $\mathbf{P}^\pi$. Especificamente, qualquer simplex em $\Delta^{|S|-1}$ cujo j-ésimo vértice pertencesse a $[\max_i p_{ij}^\pi, 1]$ seria uma solução exata para o problema. Estendendo um pouco o raciocínio, não é difícil imaginar situações em que uma representação exata de $\mathbf{P}^\pi$ seria possível utilizando-se menos de $|S|$ arquétipos. Isso ocorre por exemplo quando uma das linhas de $\mathbf{P}^\pi$ representa um ponto em $\Delta^{|S|-1}$ que é interno ao envoltório convexo formado pelas demais. Mesmo que uma fatoração exata não seja possível, a expectativa é que o resíduo deixado por uma fatoração estocástica aproximada diminua à medida que o hiper-volume definido pelas linhas de $\mathbf{P}^\pi$ diminui.

Para verificar essas afirmações, apresento nesta seção uma série de experimentos em que o Algoritmo 3.1 é usado para aproximar matrizes de transições com diferentes características. Para tal, desenvolvi um método para gerar os vetores-linha de $\mathbf{P}^\pi$ que permite configurações representativas dos vários cenários descritos acima. A estratégia consiste

em interpretar cada linha $\mathbf{p}_i^\pi$ como uma distribuição de probabilidades paramétrica, de forma que seja possível controlar as suas características através do ajuste de seus parâmetros. A distribuição escolhida nesse caso foi a distribuição normal, que além de simples constitui uma hipótese razoável para modelar muitos processos reais [85]. Nesse caso, cada elemento de $\mathbf{P}^\pi$ é dado por[5]

$$p_{ij}^\pi = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(j-\mu_i)^2}{2\sigma^2}\right), \tag{3.42}$$

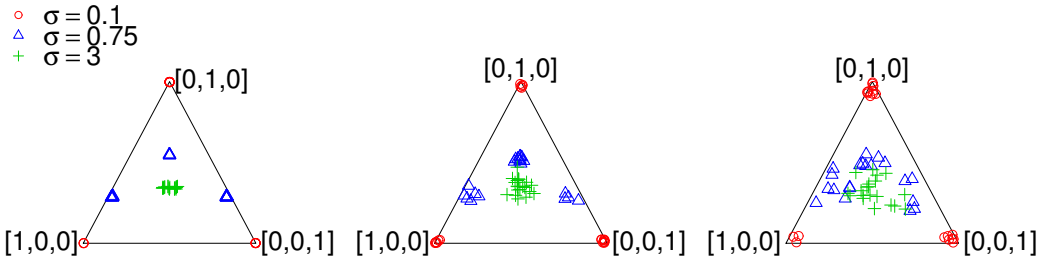
onde μ_i é o "centro" da distribuição, sorteado em um subconjunto de $\{1, 2, ..., |S|\}$ para cada linha $\mathbf{p}_i^\pi$. A cardinalidade desse subconjunto, que chamarei de ϑ, está claramente relacionada com o posto estocástico $\hat{\varrho}(\mathbf{P}^\pi)$ das matrizes geradas. O parâmetro σ é o desvio-padrão da curva normal, que nesse caso define o "nível de estocasticidade" de $\mathbf{P}^\pi$. Quanto maior esse parâmetro, mais próximas as linhas $\mathbf{p}_i^\pi$ são de uma distribuição uniforme em $\{1, 2, ..., |S|\}$. Um inconveniente de se usar a equação (3.42) para definir as linhas de $\mathbf{P}^\pi$ é que se o mesmo centro μ for sorteado para duas linhas $\mathbf{p}_i^\pi$ e $\mathbf{p}_j^\pi$, os elementos dessas linhas serão idênticos. Embora essa situação não seja particularmente inverossímil, seria interessante definir um mecanismo que permitisse controlar o nível de similaridade entre as linhas de $\mathbf{P}^\pi$ com mesmo centro μ. Assim, pode-se somar a cada elemento p_{ij}^π um pequeno ruído amostrado de uma distribuição uniforme em $[0, \eta]$, onde η é um parâmetro que controla o nível de similaridade entre as linhas de $\mathbf{P}^\pi$.

Explicitamente, o processo de construção de uma matriz $\mathbf{P}^\pi$ se dá da seguinte maneira: primeiro, um subconjunto de $\{1, 2, ..., |S|\}$ com $\vartheta \leq |S|$ elementos é gerado ao acaso. A seguir, o i-ésimo elemento desse conjunto é selecionado como o centro μ_i da linha $\mathbf{p}_i^\pi$. Os centros das $|S| - \vartheta$ linhas restantes de $\mathbf{P}^\pi$ são amostrados de maneira independente e uniforme do subconjunto definido no passo anterior. Finalmente, os elementos p_{ij}^π são calculados através de (3.42), e um ruído em $[0, \eta]$ é adicionado a cada um deles. É este último passo que diferencia as linhas $\mathbf{p}_i^\pi$ com mesmo centro μ. Depois de todo esse processo os elementos de cada linha $\mathbf{p}_i^\pi$ são normalizados de forma a garantir que sua soma seja 1.[6]

[5]Note uma pequena inconsistência na equação (3.42): o símbolo "π" é utilizado tanto para denotar a política π quanto a dízima $3,1416...$.

[6]Uma outra opção para simular uma distribuição normal, provavelmente mais correta do ponto de vista formal, seria definir o elemento p_{ij}^π como a área delimitada pela função gaussiana sobre um intervalo em

O procedimento de geração de uma matriz $\mathbf{P}^\pi$ descrito acima pode ser visto como uma amostragem de $|S|$ vetores estocásticos de uma distribuição definida em $\Delta^{|S|-1}$ através dos parâmetros ϑ, σ e η. Para que o leitor tenha uma idéia das características dessa distribuição, foram geradas amostras de 20 vetores em Δ^2 usando diferentes combinações de valores para σ e η (como o efeito de ϑ é fácil de entender, o valor desse parâmetro foi fixado em $\vartheta = 3$). O resultado dessa simulação pode ser visto na Figura 3.3. Como fica claro na figura, as distribuições apresentam 3 "focos" correspondendo aos 3 vértices do simplex Δ^2. O que define o foco do vetor $\mathbf{p}_i^\pi$ é o parâmetro μ_i, e portanto o que determina o número de focos da distribuição é ϑ. Note que quando $\eta = 0$ esse parâmetro define o posto estocástico da matriz $\mathbf{P}^\pi$. Observe também que o efeito de um aumento no desvio-padrão σ é deslocar os focos das distribuições das extremidades de Δ^2 para o seu centro. Se o efeito de um aumento de σ é deslocar os focos da distribuição, o de η é aumentar o seu raio.



(a) $\eta = 0.01$ (b) $\eta = 0.1$ (c) $\eta = 0.3$

Figura 3.3: Distribuição dos vetores estocásticos gerados em Δ^2 usando a estratégia descrita no texto com $\vartheta = 3$. Foram gerados 20 vetores para cada valor do desvio-padrão σ e do ruído η.

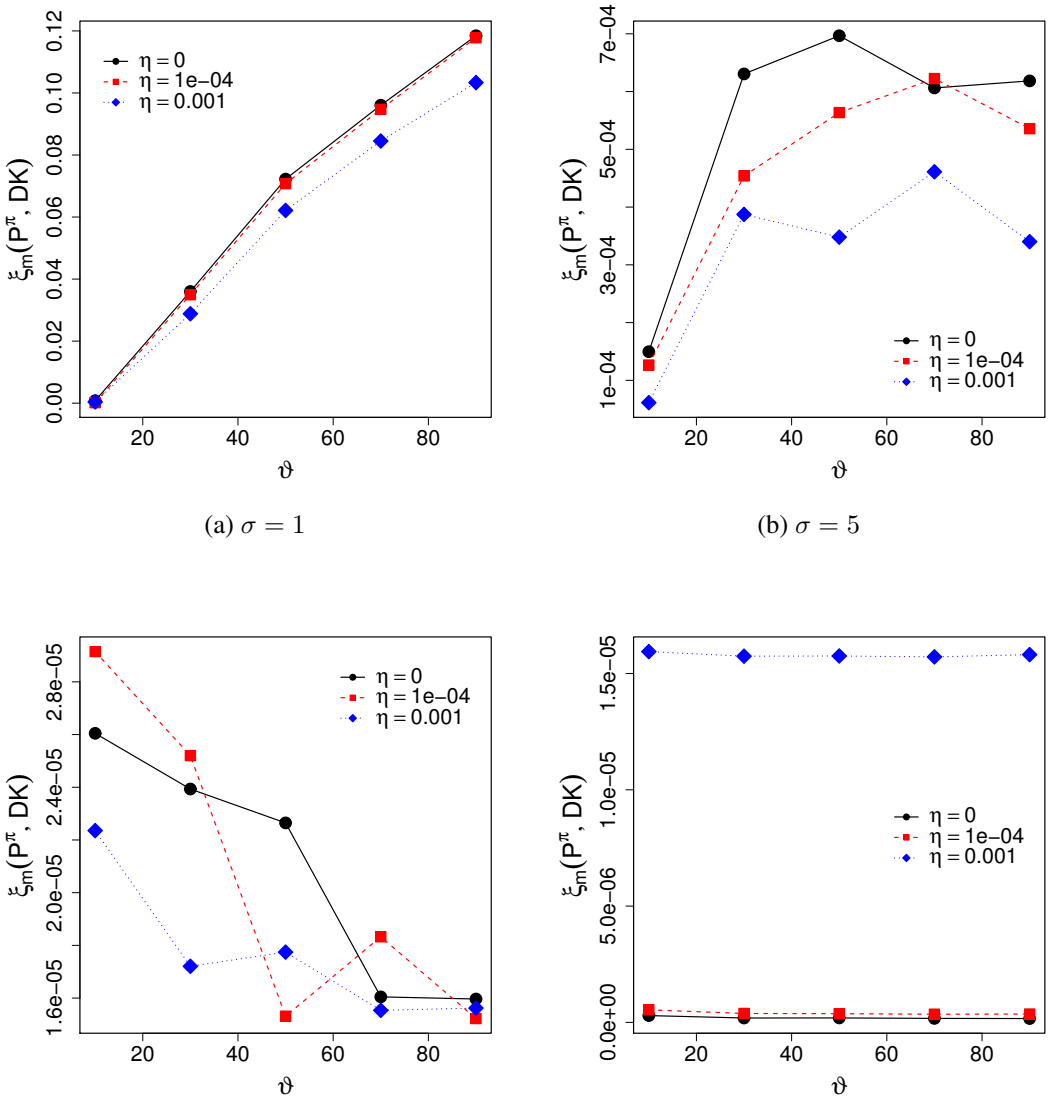
A estratégia descrita acima permite a geração de matrizes $\mathbf{P}^\pi$ bastante diferentes entre si. Antes de relatar os experimentos, porém, é preciso justificar os valores adotados para os diferentes parâmetros, o primeiro deles sendo a dimensão $|S|$ das matrizes a serem aproximadas. Após uma bateria preliminar de experimentos, ficou claro que os resultados do Algoritmo 3.1 seguem um padrão que independe da cardinalidade de S (contanto, é

torno de j. Por exemplo, p_{ij}^π poderia ser calculado como a integral de (3.42) em $[j - 0.5, j + 0.5]$. No entanto, como aqui estou interessado apenas na característica qualitativa da curva que descreve as transições, a distorção causada pelo uso de uma estratégia simplificada não tem maior importância.

claro, que um número suficientemente grande de iterações seja executado). Assim, um valor de $|S| = 100$ me pareceu um bom compromisso entre a dificuldade do problema e o custo computacional dos experimentos. O próximo passo foi determinar o número m de arquétipos usados nas aproximações. Como o objetivo aqui é estudar o efeito das características de $\mathbf{P}^\pi$ sobre a sua fatoração estocástica, esta decisão me pareceu menos importante, e um valor de $m = 20$ foi adotado de maneira mais ou menos arbitrária (à frente analiso detalhadamente o efeito de m sobre a qualidade das aproximações).

Há que se discutir ainda os valores de ϑ, σ e η usados para definir as distribuições de onde foram amostradas as linhas de $\mathbf{P}^\pi$. Para o parâmetro ϑ foram selecionados valores representativos do conjunto de valores possíveis, que nesse caso é $\{1, 2, ..., 100\}$. Especificamente, ϑ assumiu nos experimentos os valores 10, 30, 50, 70 e 90. Os desvios-padrões σ foram determinados de forma a originar matrizes de transições com características bem diferentes. Para $|S| = 100$, valores de σ no conjunto $\{1, 5, 10, 50\}$ resultam em matrizes $\mathbf{P}^\pi$ que variam desde o caso em que as transições de uma linha ficam concentradas em torno de um único elemento até o caso em que os vetores $\mathbf{p}_i^\pi$ aproximam-se de uma distribuição uniforme em $\{1, 2, ..., 100\}$. Finalmente, ficou constatado que um valor de 10^{-3} para o parâmetro η descaracterizava bastante as distribuições originais em que $\eta = 0$. Para dar uma idéia do efeito de um aumento gradativo desse parâmetro, além desses dois extremos η assumiu nos experimentos o valor intermediário de 10^{-4}. Todas as combinações possíveis dos valores dos parâmetros ϑ, σ e η deram origem a 60 distribuições de vetores estocásticos em Δ^{99}. Para cada distribuição foram geradas 50 matrizes $\mathbf{P}^\pi$ diferentes. Os resultados do Algoritmo 3.1 na aproximação dessas matrizes são mostrados na Figura 3.4. A função de custo adotada é o erro médio por estado: $\xi_m = \xi/|S|$.

A análise dos resultados mostrados na Figura 3.4 deixa claro que a distribuição dos vetores-linha de $\mathbf{P}^\pi$ em Δ^{99} tem, de fato, um enorme impacto sobre a qualidade da fatoração estocástica dessa matriz. Para entender o efeito dos parâmetros ϑ, σ e η, basta analisar a sua influência sobre o simplex definido por $\mathbf{P}^\pi$. Um aumento do desvio-padrão σ tende a concentrar os vetores-linha de $\mathbf{P}^\pi$ na região central de $\Delta^{|S|-1}$, o que corresponde a uma diminuição do volume do simplex definido pelas linhas dessa matriz. O efeito esperado de um aumento de σ é, portanto, uma diminuição no erro de aproximação. Observando a Figura 3.4 fica evidente que isso é de fato o que acontece, independentemente dos valores



(a) $\sigma = 1$

(b) $\sigma = 5$

(c) $\sigma = 10$

(d) $\sigma = 50$

Figura 3.4: Resultado da aproximação de matrizes de transições $\mathbf{P}^{\pi}$ com diferentes características. Para cada combinação dos parâmetros ϑ, σ e η foram geradas 50 matrizes 100×100. Os valores acima se referem ao resultado médio do Algoritmo 3.1 nesses 50 casos após 100 iterações usando $m = 20$ arquétipos.

de ϑ e η (note a diferença de escala nos eixos das ordenadas). Em contraste, o efeito do parâmetro η depende do desvio-padrão σ. Quando σ é pequeno, um aumento de η tende a deslocar os vetores para o centro de $\Delta^{|S|-1}$, diminuindo o simplex definido pelos vetores-linha de $\mathbf{P}^{\pi}$. Por outro lado, quando σ é grande os vetores $\mathbf{p}_i^{\pi}$ já se encontram na região central do simplex $\Delta^{|S|-1}$, e o acréscimo de um ruído η aumenta o volume do simplex definido por $\mathbf{P}^{\pi}$ (veja a Figura 3.3). Observe como na Figura 3.4a o erro de aproximação diminui com o aumento de η, enquanto na Figura 3.4d ocorre justamente o contrário. As outras duas figuras representam situações intermediárias.

O parâmetro ϑ também parece interagir com σ, embora de uma maneira mais complexa. O efeito dessa variável é fácil de entender quando σ é pequeno. Nesse caso, os vetores $\mathbf{p}_i^{\pi}$ se concentram nas vizinhanças de alguns dos vértices de Δ^{99}. O que determina o vértice de $\mathbf{p}_i^{\pi}$ é o seu centro μ_i. Quanto maior o número de centros μ_i distintos, portanto, maior o volume do simplex que contém $\mathbf{P}^{\pi}$. É por isso que na Figura 3.4a o erro de aproximação cresce monotonicamente com o parâmetro ϑ. Por outro lado, à medida que $\sigma \to \infty$, os vetores $\mathbf{p}_i^{\pi}$ deixam de "pertencer" a um dos vértices do simplex Δ^{99}, e o efeito de ϑ passa a ser mais difícil de explicar. Em particular, na Figura 3.4c o aumento desse parâmetro parece facilitar a fatoração estocástica de $\mathbf{P}^{\pi}$. Aqui posso fazer pouco mais do que especular sobre as causas desse fenômeno. A hipótese que tenho é que o decréscimo de ξ_m na Figura 3.4c está relacionado com a natureza do método usado para a aproximação. Note que a diminuição do parâmetro ϑ tem dois efeitos contraditórios. Se por um lado ele diminui o volume do simplex definido pelas linhas de $\mathbf{P}^{\pi}$, por outro ele deixa a sua fatoração estocástica mais sensível à escolha dos valores iniciais para $\mathbf{D}$ e $\mathbf{K}$ (imagine, por exemplo, que os vetores-linha da matriz $\mathbf{K}$ inicial estejam "distantes" dos vetores $\mathbf{p}_i^{\pi}$). Quando σ é suficientemente pequeno é razoável esperar que o primeiro efeito seja preponderante, mas à medida que essa variável aumenta pode haver uma inversão. Se esse for mesmo o caso, os erros de aproximação da Figura 3.4c podem ser conseqüência da estratégia de inicialização adotada no Algoritmo 3.1, em que $\mathbf{D}$ e $\mathbf{K}$ são geradas de maneira aleatória. Qualquer que seja a explicação para esse fenômeno, é importante notar que as diferenças nos erros de aproximação mostrados na Figura 3.4c são muito pequenas, quase insignificantes quando comparadas com aquelas mostradas na Figura 3.4a, que é o cenário mais fácil de entender.

A fatoração estocástica de matrizes de transições $\mathbf{P}^\pi$ geradas com diferentes valores de ϑ, σ e η dá uma boa idéia de quais são as características dessas matrizes que determinam o sucesso da sua aproximação. É evidente que se pode pensar em outras distribuições de vetores em $\Delta^{|S|-1}$ que dariam origem a matrizes de transições com características diferentes. Qual seria, por exemplo, a conseqüência de se adotar um desvio-padrão σ_i diferente na geração de cada linha $\mathbf{p}_i^\pi$ de $\mathbf{P}^\pi$? Como existe uma quantidade inesgotável de configurações possíveis da matriz $\mathbf{P}^\pi$, é importante extrair padrões que permitam uma generalização para casos não vistos. As conclusões desta seção são um passo nesse sentido. Aparentemente, a dificuldade de se fatorar uma matriz estocástica é diretamente proporcional ao volume do envoltório convexo definido pelos seus vetores-linha. Isso significa que à medida que a distância média $\| \mathbf{p}_i^\pi - \mathbf{p}_j^\pi \|$ diminui, a tendência é que a qualidade de uma eventual fatoração estocástica de $\mathbf{P}^\pi$ aumente. No caso em que as linhas de $\mathbf{P}^\pi$ são aproximadamente uma distribuição normal, duas variáveis influenciam a distância média entre elas. Uma é o número de estados em torno dos quais as transições se concentram, ou o número de centros distintos ϑ. Esse número aproxima-se de $\hat{\varrho}(\mathbf{P}^\pi)$ à medida que $\mathbf{P}^\pi$ torna-se mais determinística. Um outro fator que influencia a qualidade da fatoração estocástica das matrizes é o seu grau de estocasticidade. Em geral, matrizes $\mathbf{P}^\pi$ em que as linhas $\mathbf{p}_i^\pi$ são próximas de uma distribuição uniforme são mais fáceis de se aproximar do que aquelas em que as probabilidades de transição são muito desbalanceadas.

Adicionando recompensas às transições

Agora que se tem uma idéia de quais são as características de $\mathbf{P}^\pi$ que facilitam ou dificultam a sua aproximação, é natural perguntar qual seria o efeito de se incluir o vetor $\mathbf{r}^\pi$ ao problema. Existem pelo menos duas maneiras de se fatorar um processo de Markov completo, ou seja, de se resolver (3.19) e (3.20) conjuntamente. Supondo que a função de custo usada para avaliar (3.20) seja a mesma adotada em (3.34), uma solução simples seria submeter a aproximação de $\mathbf{r}^\pi$ à de $\mathbf{P}^\pi$. Isso é, dadas as soluções $\mathbf{D}$ e $\mathbf{K}$ para (3.34), o vetor $\bar{\mathbf{r}}^\pi$ poderia ser calculado como

$$\bar{\mathbf{r}}^\pi = (\mathbf{D}^\mathrm{t}\mathbf{D})^{-1}\mathbf{D}^\mathrm{t}\mathbf{r}^\pi, \tag{3.43}$$

que corresponde à projeção ortogonal de $\mathbf{r}^\pi$ no espaço alcançado pelos vetores em $\mathbf{D}$. Note, no entanto, que nesse caso a otimização de $\mathbf{D}$ seria independente de $\mathbf{r}^\pi$, o que limitaria a qualidade das soluções encontradas. Uma alternativa mais interessante é concatenar o vetor $\mathbf{r}^\pi$ à matriz $\mathbf{P}^\pi$ e considerar a otimização de ambos conjuntamente. Ou seja, dada

$$\mathbf{M}^\pi = \begin{bmatrix} r^\pi_1 & p^\pi_{11} & p^\pi_{12} & \cdots & p^\pi_{1|S|} \\ r^\pi_2 & p^\pi_{21} & p^\pi_{22} & \cdots & p^\pi_{2|S|} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ r^\pi_{|S|} & p^\pi_{|S|1} & p^\pi_{|S|2} & \cdots & p^\pi_{|S||S|} \end{bmatrix}, \tag{3.44}$$

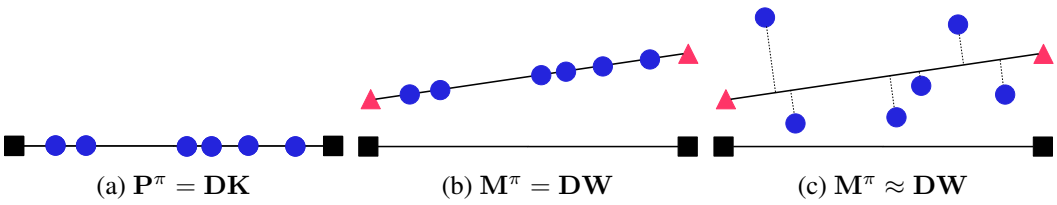
o problema passa a ser encontrar um vetor $\overline{\mathbf{r}}^\pi$ e matrizes estocásticas $\mathbf{D}$ e $\mathbf{K}$ tais que $\xi(\mathbf{M}^\pi, \mathbf{DW})$ seja minimizada, onde $\mathbf{W}$ é a matriz formada pela concatenação de $\overline{\mathbf{r}}^\pi$ à esquerda de $\mathbf{K}$, de maneira análoga a (3.44).

Note que como $\mathbf{W} = [\overline{\mathbf{r}}^\pi, \mathbf{K}]$, a restrição de estocasticidade de $\mathbf{K}$ recai sobre $\mathbf{W}$, que poderia ser chamada de "parcialmente estocástica"—os elementos em cada uma de suas linhas, *a partir da segunda coluna*, são não-negativos e têm soma 1. Uma maneira simples de lidar com essa restrição é adaptar o Algoritmo 3.1 de forma a garantir que a primeira coluna de $\mathbf{W}$ permaneça livre. Nesse caso, as equações de atualização (3.39) e (3.40) seriam idênticas, com $\mathbf{W}$ no lugar de $\mathbf{K}$, mas a projeção posterior de $\mathbf{W}$ seria ligeiramente diferente: apenas os $|S|$ últimos elementos de cada linha dessa matriz seriam projetados em $\Delta^{|S|-1}$. Embora esta solução *ad-hoc* não seja muito elegante, ela é suficiente para os propósitos ilustrativos desta seção.[7]

Geometricamente, a inclusão de $\mathbf{r}^\pi$ ao problema pode ser vista como a adição de uma dimensão extra ao simplex $\Delta^{|S|-1}$, que deixa de ter apenas vértices estocásticos. Isso pode dificultar consideravelmente a aproximação. Suponha, por exemplo, que as linhas de $\mathbf{P}^\pi$ estejam confinadas em um simplex de dimensão $m-1$ cujos vértices sejam as linhas da matriz $\mathbf{K}$. Nesse caso, uma fatoração exata $\mathbf{P}^\pi = \mathbf{DK}$ é possível. No entanto, a qualidade da aproximação do processo de Markov $\mathbf{M}^\pi$ fica sujeita à existência de um vetor $\overline{\mathbf{r}}^\pi \in \mathbb{R}^m$ tal que $\mathbf{r}^\pi \approx \mathbf{D}\overline{\mathbf{r}}^\pi$. Dependendo do vetor $\mathbf{r}^\pi$, a aproximação $\mathbf{M}^\pi \approx \mathbf{DW}$ pode

[7] Ao adotar o Algoritmo 3.1 para resolver o problema estou assumindo implicitamente que $\overline{\mathbf{r}}^\pi \geq 0$. Se esse não for o caso, pode-se usar (3.43) ou reformular o problema de maneira que essa restrição seja atendida [161].

ser muito ruim, mesmo que uma fatoração exata de $\mathbf{P}^\pi$ exista. A Figura 3.5 ilustra esse fenômeno para o caso em que $m = 2$. A Figura 3.5a mostra a situação em que apenas a matriz $\mathbf{P}^\pi$ está sendo aproximada. Nesse caso, uma representação exata é possível através da combinação convexa das linhas de $\mathbf{K}$. A Figura 3.5b mostra um exemplo benigno da inclusão do vetor $\mathbf{r}^\pi$ ao problema: embora as linhas de $\mathbf{M}^\pi$ tenham uma terceira dimensão, ainda é possível representá-las de maneira exata com os vetores-linha de $\mathbf{W}$. Finalmente, na Figura 3.5c é possível ver uma situação em que a inclusão de $\mathbf{r}^\pi$ dificultou consideravelmente o problema. Nesse caso, existe um erro intrínseco na aproximação de $\mathbf{M}^\pi$ com apenas 2 vetores, qualquer que seja o método utilizado para fazê-lo.



(a) $\mathbf{P}^\pi = \mathbf{DK}$ (b) $\mathbf{M}^\pi = \mathbf{DW}$ (c) $\mathbf{M}^\pi \approx \mathbf{DW}$

Figura 3.5: Representação gráfica da inclusão do vetor-recompensa $\mathbf{r}^\pi$ ao problema de aproximação para o caso em que $|S| = 6$ e $m = 2$. Os quadrados representam os vetores-linha da matriz $\mathbf{K}$ e os triângulos as linhas de $\mathbf{W}$. Na primeira figura os círculos são as linhas de $\mathbf{P}^\pi$ e nas demais eles representam as linhas de $\mathbf{M}^\pi$.

Neste ponto deve estar claro que só faz sentido falar em dificuldade de aproximação de $\mathbf{r}^\pi$ se a matriz $\mathbf{P}^\pi$ correspondente for levada em consideração. Em princípio, seria concebível desenvolver uma estratégia para gerar os vetores $\mathbf{r}^\pi$ que levasse em conta as características de $\mathbf{P}^\pi$, de forma que fosse possível controlar o nível de dificuldade de fatoração de $\mathbf{M}^\pi$ nos experimentos. No entanto, adotei aqui uma postura bem mais simples: a cada processo de Markov gerado, os elementos de $\mathbf{r}^\pi$ foram sorteados de maneira independente em uma distribuição uniforme em $[0, 1]$. Assim, é possível ter uma idéia da dificuldade adicional de se incluir $\mathbf{r}^\pi$ ao problema sem que nenhuma forma particular de $\mathbf{P}^\pi$ seja favorecida.

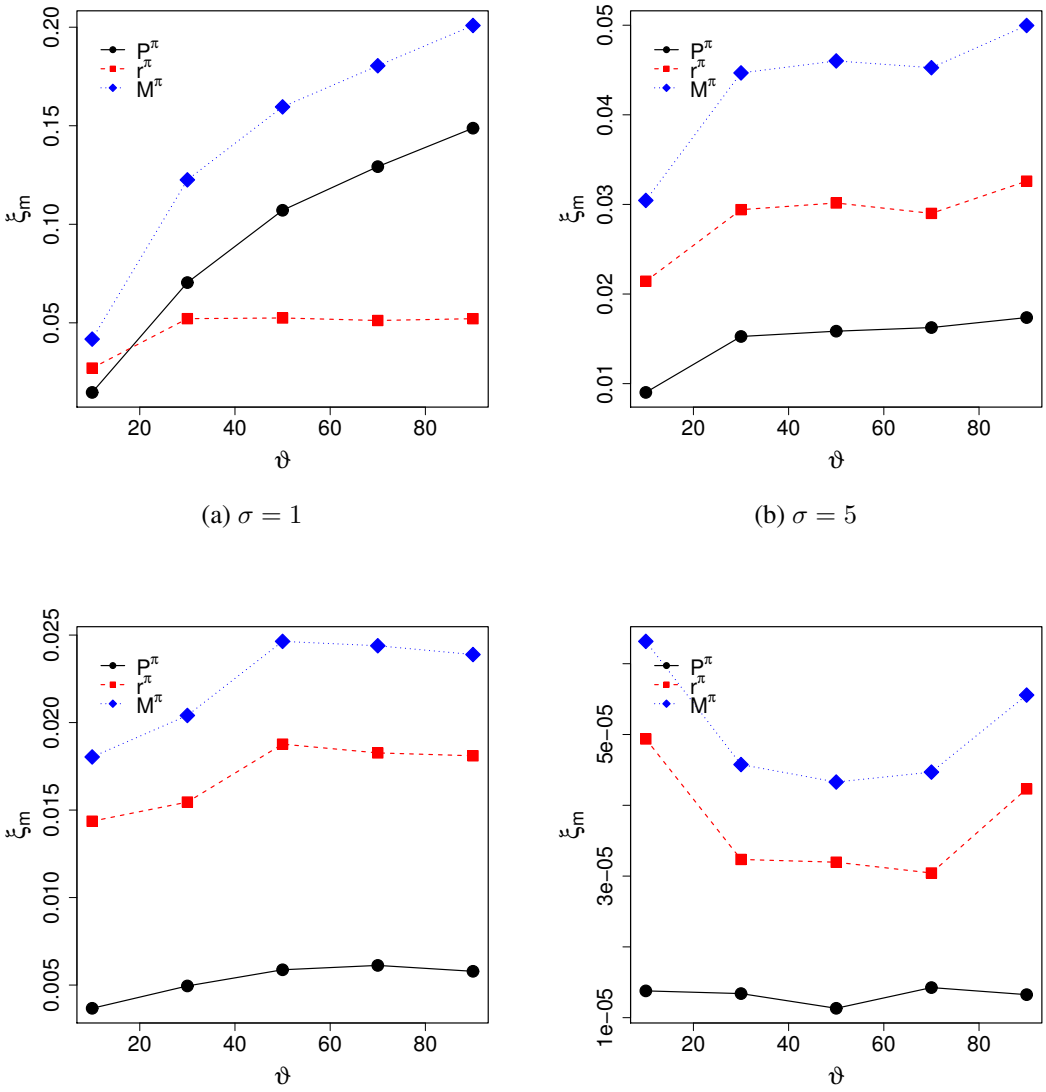
A Figura 3.6 mostra a contribuição de $\mathbf{P}^\pi$ e $\mathbf{r}^\pi$ para o erro total de aproximação de $\mathbf{M}^\pi$. Note que a inclusão de $\mathbf{r}^\pi$ ao problema resulta em um aumento do erro médio de aproximação de $\mathbf{P}^\pi$ em relação aos valores mostrados na Figura 3.4. Isso não chega a ser

uma surpresa, um vez que aqui a otimização de $\mathbf{D}$ leva também em consideração o vetor $\mathbf{r}^\pi$. A otimização conjunta de $\bar{\mathbf{r}}^\pi$ e $\mathbf{K}$ explica também porque ϑ e σ têm um impacto sobre os erros de aproximação de $\mathbf{r}^\pi$, cujos elementos são gerados de maneira completamente independente desses parâmetros. Note como em geral os parâmetros ϑ e σ têm efeitos sobre a função de custo $\xi_m(\mathbf{r}^\pi, \mathbf{D}\bar{\mathbf{r}}^\pi)$ que reproduzem os padrões detectados na seção anterior para a aproximação de $\mathbf{P}^\pi$ sozinha. No entanto, o desvio-padrão σ parece ter um impacto mais forte sobre esta última do que sobre a aproximação de $\mathbf{r}^\pi$. Isso fica claro quando se observa que $\xi_m(\mathbf{r}^\pi, \mathbf{D}\bar{\mathbf{r}}^\pi) > \xi_m(\mathbf{P}^\pi, \mathbf{DK})$ quando $\sigma \geq 5$, independentemente do valor de ϑ. Esse fenômeno é um forte indício de que em algumas situações a interação do vetor $\mathbf{r}^\pi$ com a matriz $\mathbf{P}^\pi$ tem um impacto maior sobre a fatoração do processo de Markov do que as características de $\mathbf{P}^\pi$ isoladamente.

É interessante notar, porém, que em um certo sentido a estratégia adotada aqui para gerar os vetores $\mathbf{r}^\pi$ representa o pior cenário. Em aplicações reais, é comum que todos os elementos de $\mathbf{r}^\pi$ tenham o mesmo valor ou valores parecidos. Esse é o caso, por exemplo, de problemas de menor caminho, em que em geral apenas as recompensas associadas com o objetivo final são diferentes das demais. Nesses casos a aproximação de $\mathbf{M}^\pi$ ficaria reduzida à fatoração estocástica de $\mathbf{P}^\pi$. De qualquer forma, o importante nessa seção é que se tenha uma idéia de como as características da matriz de transições $\mathbf{P}^\pi$ e do vetor de recompensas $\mathbf{r}^\pi$ influenciam a fatoração estocástica de um processo de Markov $\mathbf{M}^\pi$. Na próxima seção discuto o impacto dos erros de aproximação de $\mathbf{M}^\pi$ no cálculo de sua função de valor $\mathbf{v}^\pi$.

Impacto sobre a função de valor

Do ponto de vista da programação dinâmica, a única relevância da qualidade das aproximações de $\mathbf{P}^\pi$e $\mathbf{r}^\pi$ é o seu impacto sobre o cálculo da função de valor do processo de Markov. Colocado de uma maneira simples, pode-se dizer que a "verdadeira" medida de qualidade de uma aproximação $\tilde{\mathbf{M}}^\pi$ é a similaridade entre $\tilde{\mathbf{v}}^\pi$ e $\mathbf{v}^\pi$. Em particular, dadas duas aproximações $\tilde{\mathbf{M}}_1^\pi$ e $\tilde{\mathbf{M}}_2^\pi$ de um processo de Markov $\mathbf{M}^\pi$, pode-se dizer que a primeira é melhor do que a segunda se $\| \mathbf{v}^\pi - \tilde{\mathbf{v}}_1^\pi \| < \| \mathbf{v}^\pi - \tilde{\mathbf{v}}_2^\pi \|$, *mesmo que* $\| \mathbf{M}^\pi - \tilde{\mathbf{M}}_1^\pi \| > \| \mathbf{M}^\pi - \tilde{\mathbf{M}}_2^\pi \|$ (aqui $\| \cdot \|$ é uma norma qualquer). Torna-se portanto



(a) $\sigma = 1$

(b) $\sigma = 5$

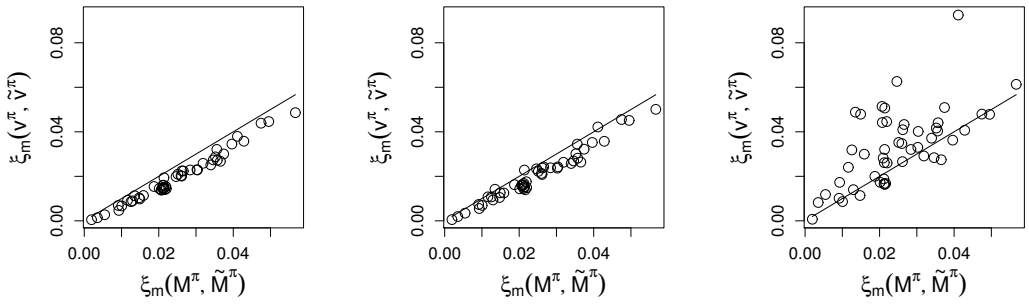
(c) $\sigma = 10$

(d) $\sigma = 50$

Figura 3.6: Contribuição de $\xi_m(\mathbf{P}^\pi, \mathbf{DK})$ e $\xi_m(\mathbf{r}^\pi, \mathbf{D}\bar{\mathbf{r}}^\pi)$ para o erro total de aproximação $\xi_m(\mathbf{M}^\pi, \mathbf{DW})$, para vários valores dos parâmetros ϑ e σ. As matrizes $\mathbf{P}^\pi$ foram geradas com $\eta = 0$. Os elementos dos vetores-recompensa $\mathbf{r}^\pi$ foram sorteados de uma distribuição uniforme em $[0, 1]$. Os processos de Markov são a concatenação de $\mathbf{r}^\pi$ e $\mathbf{P}^\pi$, ou seja, $\mathbf{M}^\pi = [\mathbf{r}^\pi, \mathbf{P}^\pi]$. Os valores se referem aos resultados médios do Algoritmo 3.1 em 50 casos gerados de maneira independente, após 100 iterações, usando $m = 20$.

fundamental verificar se esse tipo de inversão é possível, o que equivaleria a dizer que o efeito de um erro de aproximação não é determinado unicamente pela sua magnitude.

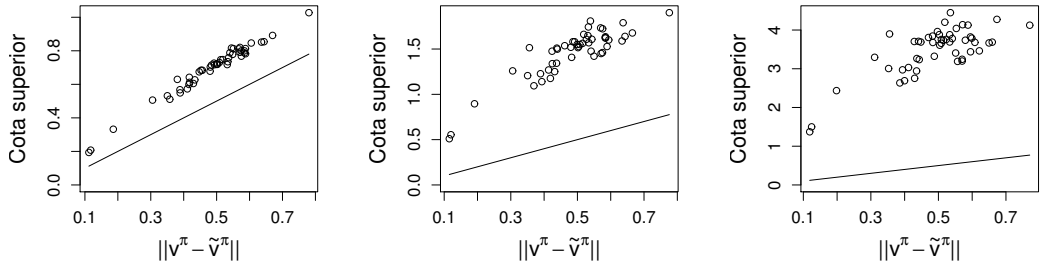
Intuitivamente, parece provável que erros com a mesma magnitude possam ter efeitos diferentes sobre o cálculo de $\tilde{\mathbf{v}}^{\pi}$. Um erro na aproximação de um elemento de $\mathbf{P}^{\pi}$, por exemplo, tende a ser mais danoso se a recompensa associada for grande. Para ilustrar esse fenômeno na prática, eu mostro na Figura 3.7 os erros no cálculo da função de valor como uma função do erro de aproximação de $\mathbf{M}^{\pi}$. Nesse caso, se o erro no cálculo de $\mathbf{v}^{\pi}$ fosse totalmente determinado por $\| \mathbf{M}^{\pi} - \tilde{\mathbf{M}}^{\pi} \|$, os pontos da Figura 3.7 formariam uma curva monotonicamente crescente em relação a essa diferença. Confirmando as suspeitas, isso não é o que ocorre. Observe como erros de aproximação de $\mathbf{M}^{\pi}$ com aproximadamente a mesma magnitude podem resultar em funções de valor de diferentes qualidades. Esse fenômeno tende a se agravar com o aumento do fator de desconto γ, provavelmente porque os resíduos deixados pelas aproximações de $\mathbf{P}^{\pi}$ e $\mathbf{r}^{\pi}$ acumulam com maior peso no cálculo de $\mathbf{v}^{\pi}$.



(a) $\gamma = 0.5$ ($r = 0.99$) (b) $\gamma = 0.7$ ($r = 0.98$) (c) $\gamma = 0.9$ ($r = 0.64$)

Figura 3.7: Correlação entre o erro de aproximação de um processo de Markov e o erro no cálculo de sua função de valor. As funções de valor $\tilde{\mathbf{v}}^{\pi}$ foram calculadas a partir das aproximações $\tilde{\mathbf{M}}^{\pi}$ do experimento mostrado na Figura 3.6, para o caso em que $\vartheta = 50$, $\sigma = 10$ e $\eta = 0$. Cada um dos 50 pontos nas figuras acima corresponde a uma execução do Algoritmo 3.1 por 100 iterações usando $m = 20$ arquétipos. A variável r mostrada entre parênteses é o coeficiente de correlação de Pearson [47].

Note que os dados da Figura 3.7 estão de acordo com as expressões que fornecem as cotas para o erro de aproximação de $\mathbf{v}^{\pi}$ derivadas na Seção 3.3.1. Como o aumento de γ torna o impacto de um erro de aproximação mais imprevisível, é natural que a cota



(a) $\gamma = 0.1$ (b) $\gamma = 0.3$ (c) $\gamma = 0.5$

Figura 3.8: Comparação entre a cota superior prevista pela expressão (3.31) e o erro $\| \mathbf{v}^\pi - \tilde{\mathbf{v}}^\pi \|_\infty$ efetivamente encontrado nas aproximações. A reta $x = y$ representa o que seria uma cota perfeita. Os processos são os mesmos usados no experimento mostrado na Figura 3.7. Cada um dos 50 pontos nas figuras acima corresponde a uma execução do Algoritmo 3.1 por 100 iterações usando $m = 20$ arquétipos.

superior para $\| \mathbf{v}^\pi - \tilde{\mathbf{v}}^\pi \|_\infty$ cresça à medida que $\gamma \to 1$. Além disso, como a derivação de um limite teórico leva em consideração o pior caso, a expectativa é que a sua relevância na prática diminua com o aumento do fator de desconto. Para mostrar que isso é de fato o que ocorre, comparo na Figura 3.8 os erros de aproximação da figura anterior, agora medidos com a norma do máximo, com a cota superior determinada pela expressão (3.31). Observe na Figura 3.8a como a cota é relativamente justa quando $\gamma = 0.1$. No entanto, à medida que esse parâmetro cresce os limites previstos pela expressão (3.31) tornam-se superestimativas com pouca ou nenhuma utilidade prática. No caso da expressão (3.25) esse fenômeno manifesta-se de uma maneira diferente: com um aumento de γ, a condição $\epsilon < 1$ deixa de ser atendida, e portanto a cota deixa de ser válida. É importante ressaltar que a causa da degeneração desses limites teóricos, que é a presença do termo $(1 - \gamma)^{-1}$ nas expressões para calculá-los, é uma constante em resultados semelhantes na área [22, 55], e reflete uma dificuldade inerente ao tipo de problema estudado pela programação dinâmica.

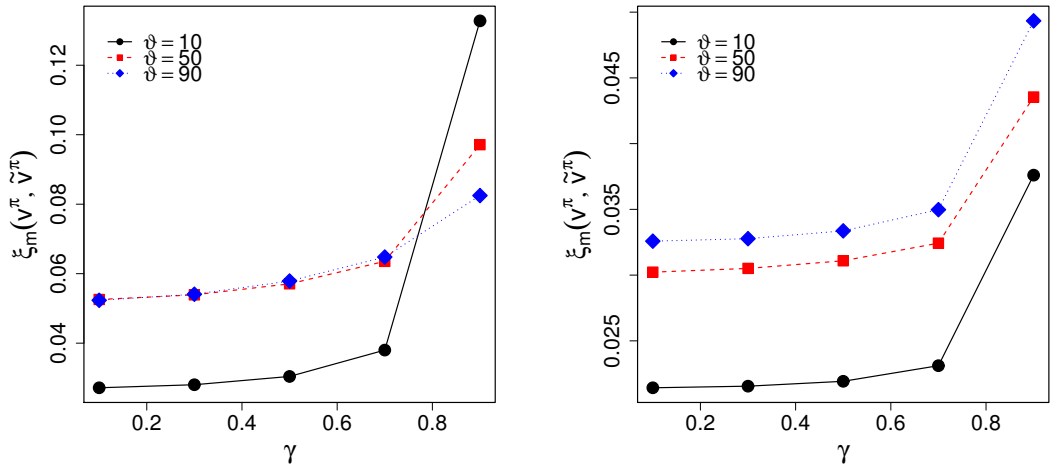
Como o erro de aproximação de um processo de Markov pode ter um efeito bastante imprevisível sobre o cálculo da sua função de valor, é natural a pergunta sobre qual seria a qualidade das funções $\tilde{\mathbf{v}}^\pi$ geradas a partir das aproximações $\tilde{\mathbf{M}}^\pi$ obtidas na seção anterior. Para responder a esta pergunta, calculei a função $\tilde{\mathbf{v}}^\pi$ de cada processo $\tilde{\mathbf{M}}^\pi$ ge-

rado no experimento anterior e comparei os resultados com as funções originais $\mathbf{v}^\pi$. Esse procedimento foi repetido para diferentes valores do fator de desconto γ. Um resumo dessa experiência pode ser visto na Figura 3.9. Observe que o efeito de ϑ e σ sobre a aproximação de $\mathbf{v}^\pi$ reproduz o impacto desses parâmetros sobre a fatoração de $\mathbf{P}^\pi$—o que sugere que as características dessa matriz são um fator determinante da qualidade da aproximação da função de valor de um processo de Markov. A variável nova aqui é o fator de desconto γ, cujo aumento tende a degenerar a aproximação $\tilde{\mathbf{v}}^\pi$. Isso está de acordo tanto com a Figura 3.7 quanto com as expressões (3.25) e (3.31).

É tentador fazer algum comentário a respeito da magnitude dos erros de aproximação mostrados na Figura 3.9. Note, no entanto, que no contexto da programação dinâmica é difícil definir o que seria um erro tolerável na aproximação da função de valor. Assim como um pequeno resíduo na aproximação de $\mathbf{M}^\pi$ pode ter efeitos catastróficos no cálculo de $\mathbf{v}^\pi$, um pequeno erro na aproximação desta última também pode ter efeitos inesperados sobre a política derivada dessa função. Como em geral a análise desses efeitos é difícil, adota-se $\| \mathbf{v}^\pi - \tilde{\mathbf{v}}^\pi \|$ como critério para avaliar a qualidade de uma aproximação. Em última análise, no entanto, o que determina o sucesso de uma aproximação no contexto da programação dinâmica é a qualidade da política encontrada para o problema de tomada de decisão seqüencial. Essa questão é investigada em detalhes na Seção 3.4.

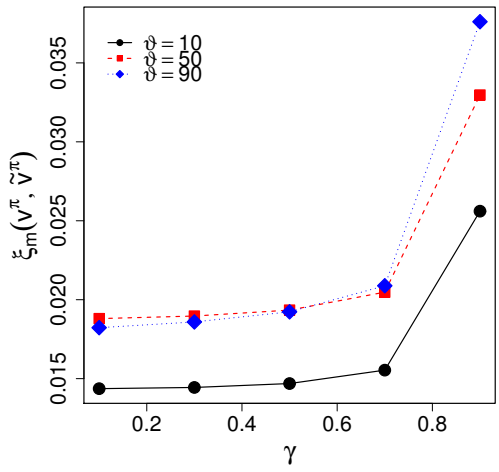
3.3.3 Soluções alternativas

Como mencionado anteriormente, o algoritmo de Lee e Seung não é a única alternativa para se realizar a fatoração estocástica de um processo de Markov. Além de vários outros algoritmos desenvolvidos para a fatoração não-negativa de uma matriz [53, 90, 76]—que a princípio podem também ser adaptados para a fatoração estocástica—, é possível adotar abordagens com filosofias completamente diferentes. Uma idéia nesse sentido seria tentar identificar o envoltório convexo formado pelos vetores $\mathbf{p}_i^\pi$. Embora em geral esse envoltório não coincida com o menor simplex estocástico que contém as linhas de $\mathbf{P}^\pi$ (Figura 3.2), ele certamente é uma solução para o problema. Nesse caso, bastaria definir as linhas de $\mathbf{W}$ como os vértices do envoltório e a matriz $\mathbf{D}$ poderia ser obtida de maneira análoga à expressão (3.41). Note que com essa estratégia o número de arquétipos

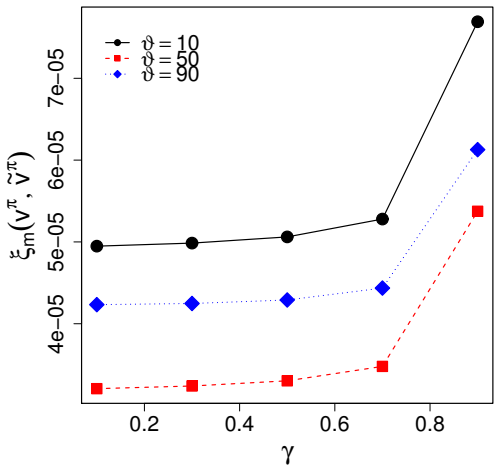


(a) $\sigma = 1$

(b) $\sigma = 5$



(c) $\sigma = 10$



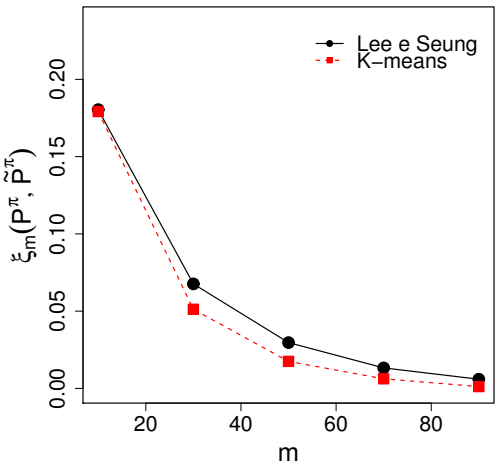
(d) $\sigma = 50$

Figura 3.9: Erro médio no cálculo da função de valor $\mathbf{v}^\pi$ para vários fatores de desconto γ. As funções $\tilde{\mathbf{v}}^\pi$ foram calculadas a partir das aproximações obtidas no experimento mostrado na Figura 3.6, e portanto os valores se referem a uma média de 50 casos.

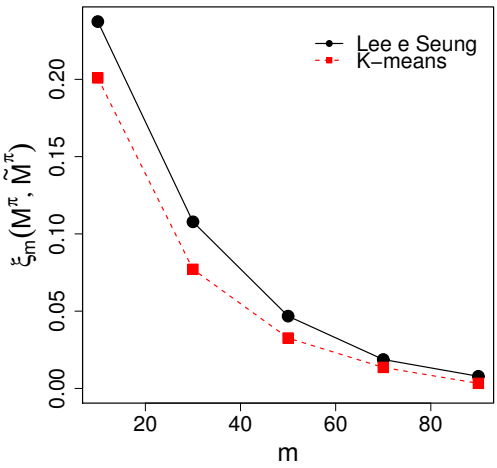
m seria definido de maneira automática. No entanto, mesmo quando m é fixado *a priori*, o envoltório convexo formado pelos vetores $\mathbf{p}_i^\pi$ pode ser útil: Cutler e Breinam [36] mostram que para $m > 1$ existe um conjunto de m arquétipos na "superfície" desse envoltório que minimiza (3.34). Existem na literatura diversos algoritmos para se determinar o envoltório convexo formado por um conjunto de pontos em um espaço de alta dimensionalidade [31, 145, 6, 34]. No caso em que se deseja apenas uma fatoração estocástica aproximada de $\mathbf{M}^\pi$, pode-se pensar em alterar esse algoritmos de forma que um determinado ponto só seja acrescentado ao envoltório se a sua distância à face mais próxima estiver acima de um limiar pré-estabelecido. No entanto, tanto a versão exata quanto a versão aproximada dessa abordagem ainda precisam ser avaliadas na prática.

Uma outra maneira de realizar a fatoração estocástica seria interpretá-la como um problema de otimização combinatória. Nesse caso o objetivo seria encontrar um subconjunto de m linhas de $\mathbf{M}^\pi$ para compor a matriz $\mathbf{W}$ que minimizassem uma função de custo adequada. Como essa formulação do problema é mais específica do que a original, é razoável supor que ela possa ser resolvida mais eficientemente. Uma outra idéia nessa mesma linha seria restringir o espaço factível de $\mathbf{D}$. Note que se as linhas dessa matriz estiverem restritas aos vértices de Δ^{m-1}, a fatoração estocástica fica reduzida ao conhecido problema de *agrupamento* ou *clusterização*, amplamente estudado na área de análise estatística de dados [58, 95, 72]. A única diferença em relação à formulação original do problema de clusterização é a restrição de "semi-estocasticidade" sobre os centros dos *clusters*, que nesse caso são as linhas da matriz $\mathbf{W}$. Note entretanto que o uso de (3.34) como função de custo equivale à minimização da distância euclidiana no agrupamento de dados, e portanto quando esse é o critério de dissimilaridade adotado a restrição de estocasticidade é naturalmente atendida por qualquer mínimo local do problema (já que os centros são internos ao envoltório convexo formado pelos pontos). Existem inúmeros algoritmos disponíveis na literatura para lidar com o problema de clusterização. Para ilustrar o potencial dessa abordagem, eu implementei o que talvez seja o mais conhecido deles, o algoritmo *k-means* [59]. Usando esse método e o Algoritmo 3.1, repeti as experiências das seções anteriores, agora variando o parâmetro m. A comparação dos dois pode ser vista na Figura 3.10.

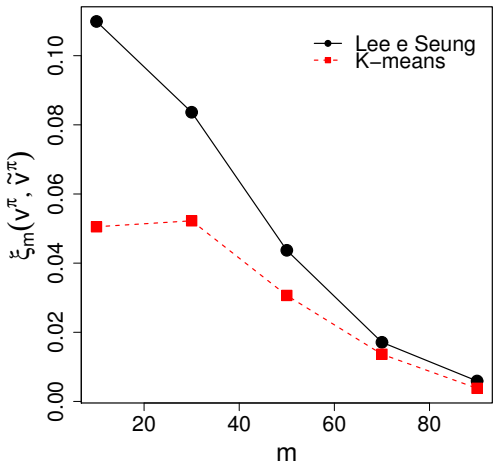
A primeira observação em relação à Figura 3.10 se refere a uma análise que ainda



(a) $\xi_m(\mathbf{P}^\pi, \mathbf{DK})$



(b) $\xi_m(\mathbf{M}^\pi, \mathbf{DW})$



(c) $\xi_m(\mathbf{v}^\pi, \mathbf{D}\bar{\mathbf{v}}^\pi)$

Figura 3.10: Erro médio na aproximação de $\mathbf{P}^\pi$, $\mathbf{M}^\pi$ e $\mathbf{v}^\pi$ pelos algoritmos de Lee e Seung (Algoritmo 3.1) e k-means para vários valores do parâmetro m. Os processos de Markov são os mesmos usados nos experimentos anteriores com $\vartheta = 50$, $\sigma = 1$ e $\eta = 0$. As funções de valor $\mathbf{v}^\pi$ foram calculadas com $\gamma = 0.9$. Os resultados do algoritmo de Lee e Seung foram obtidos após 100 iterações. O algoritmo k-means foi executado por 10 iterações apenas. Os valores se referem a uma média de 50 casos.

não havia sido feita até aqui. Note como tanto o algoritmo de Lee e Seung quanto o algoritmo k-means se comportam de acordo com o esperado quando se altera o parâmetro m. Em particular, um acréscimo no número de arquétipos sempre resulta em uma diminuição do erro de aproximação, seja da matriz $\mathbf{P}^\pi$ou do processo de Markov $\mathbf{M}^\pi$. Mais surpreendente talvez seja a constatação de que o algoritmo k-means, embora executando apenas 10% do número de iterações do Algoritmo 3.1, obteve resultados melhores do que este último, tanto em relação ao erro médio de aproximação quanto em relação ao desvio-padrão dos resultados nas 50 execuções (não mostrados na figura). Na aproximação da matriz de transições $\mathbf{P}^\pi$ a diferença nos resíduos deixados pelos dois algoritmos é pequena, mas o acréscimo do vetor $\mathbf{r}^\pi$ ao problema parece favorecer ligeiramente o algoritmo de clusterização—e essa vantagem se reflete na função de valor $\mathbf{v}^\pi$. Uma possível explicação para isso é a natureza das atualizações realizadas pelos dois métodos: como no algoritmo k-means a atualização de uma coluna da matriz $\mathbf{W}$ não influencia a atualização das demais, esse algoritmo tende a ser menos sensível à diferença de magnitude das colunas de $\mathbf{P}^\pi$ em relação a $\mathbf{r}^\pi$. Uma questão interessante que fica em aberto é se a clusterização de dados seria competitiva com um método mais geral, como o Algoritmo 3.1, caso outras métricas fossem adotadas no lugar de (3.34). Nos casos em que isso for verdade, o uso de algoritmos de agrupamento pode significar uma redução significativa no tempo de processamento. Vale observar que se um algoritmo hierárquico for adotado para a clusterização, é possível determinar de forma automática o número de arquétipos m necessários para uma boa fatoração estocástica de $\mathbf{M}^\pi$ [72].

3.3.4 A maldição do crescimento superficial

A motivação para se reduzir as dimensões de um processo de Markov é obter alguma economia de recursos computacionais, seja em termos de memória (no número de *bits* necessários para armazenar uma descrição do processo) ou em termos de tempo de execução (no número de operações aritméticas necessárias para calcular a sua função de valor). A economia no espaço de armazenamento obtida com a redução de um processo de Markov é evidente. A memória requerida para armazenar $\mathbf{P}^\pi$ e $\mathbf{r}^\pi$é $O(|S|^2)$, e portanto uma diminuição do número de estados do processo pode ter um grande impacto sobre a quantidade

de *bits* necessários para descrevê-lo. Só para se ter uma idéia, o uso de $m = 20$ arquétipos nos experimentos da seção anterior resulta em uma diminuição de aproximadamente 92.5% no uso de memória. Quando se está lidando com processos de Markov grandes, em que o número de estados alcança a casa dos milhões ou dos bilhões, essa economia pode se tornar significativa. Um exemplo interessante nesse sentido é dado por Ho e Dooren [62]. Grande parte do sucesso do Google, a mais bem-sucedida ferramenta de busca na internet da atualidade, se deve à estratégia usada por esse serviço para determinar a relevância de uma página da rede, o chamado *page-rank* [30]. Basicamente, o algoritmo responsável por essa classificação representa a estrutura da internet como uma enorme cadeia de Markov, em que cada uma das bilhões de páginas existentes corresponde a um estado. A importância das páginas fica determinada pela distribuição estacionária da cadeia, o que, intuitivamente, corresponde a medir a relevância de uma página como a probabilidade de uma pessoa seguindo *links* ao acaso acabar por visitá-la [30]. Imagine o ganho potencial de uma fatoração estocástica bem-sucedida nesse caso.

No entanto, neste trabalho estou interessado na solução de processos de Markov no contexto da programação dinâmica, e portanto faz mais sentido falar em redução do tempo de processamento do que em economia de armazenamento. Infelizmente, nesse quesito a fatoração estocástica de um processo de Markov não tem muito a oferecer. De acordo com Lin [90], o custo computacional de uma iteração do algoritmo de Lee e Seung é $O(|S|^2 m)$. Portanto, se o número de iterações necessárias para a fatoração estocástica de um processo $\mathbf{M}^\pi$ crescer pelo menos linearmente com $|S|$ (o que parece bastante provável), o custo computacional do Algoritmo 3.1 pode se tornar superior ao custo de se resolver o sistema original diretamente. Além disso, não se pode esquecer que após a fatoração estocástica de $\mathbf{M}^\pi$ é necessário calcular a matriz $\bar{\mathbf{P}}^\pi$, o que envolve $O(|S|m^2)$ operações aritméticas, e resolver o processo de Markov reduzido, o que toma um tempo na ordem de m^3.

Infelizmente, o uso do algoritmo k-means para efetuar a fatoração estocástica não é garantia de solução do problema. Como o custo por iteração desse algoritmo também é da ordem de $|S|^2 m$, assintoticamente ele só significaria uma economia de tempo de processamento em relação à solução direta do sistema linear se o número de iterações executadas por ele crescesse sub-linearmente com $|S|$. Embora exista alguma evidência empírica na literatura de que isso é de fato o que ocorre em alguns casos [46], Arthur

e Vassilvitskii [3] mostraram recentemente que para algumas distribuições de dados o algoritmo k-means requer um tempo super-polinomial para convergir.[8]

Por que exatamente ocorre essa explosão do custo computacional em ambos os casos acima? Esse fenômeno é uma conseqüência do fato de $\mathbf{M}^{\pi}$ crescer "nas duas dimensões" quando $|S|$ tende ao infinito, ou seja, um aumento no número de linhas de $\mathbf{M}^{\pi}$ vem necessariamente acompanhado de um aumento equivalente no seu número de colunas. Chamarei esse fenômeno de *crescimento superficial*, em referência ao aumento quadrático da área de uma superfície em relação ao aumento linear das suas duas dimensões.[9] Em problemas convencionais de análise de dados o crescimento superficial não acontece: em geral, o número de variáveis (colunas) permanece fixo mesmo quando o número de pontos (linhas) aumenta. Isso explica a boa escalabilidade de algoritmos como o k-means e derivados nesse tipo de problema [72, 109]. Se a dimensão do espaço aumenta a cada inclusão de um novo ponto, há um crescimento do custo computacional que é intrínseco ao problema, independentemente do algoritmo adotado para resolvê-lo. Em termos simples: um aumento no número de pontos a serem manipulados acarreta um aumento do custo computacional dessa manipulação. Em uma referência à Bellman [15], chamarei esse fenômeno de *maldição do crescimento superficial*.

Em princípio é possível imaginar maneiras de contornar os problemas causados pelo crescimento superficial. Um caminho seria tentar desenvolver algoritmos para a fatoração estocástica com uma taxa de convergência superlinear. Uma outra alternativa seria usar heurísticas para diminuir a complexidade computacional de cada iteração da aproximação. Por exemplo, existem na literatura algoritmos de clusterização desenvolvidos especialmente para problemas em espaços de alta dimensionalidade cujo custo computacional cresce com $|S|^2$ [100, 115]. Embora esta questão mereça uma investigação mais aprofundada, acredito ser muito difícil encontrar um algoritmo com uma baixa complexidade computacional que não resulte em uma degeneração da qualidade da fatoração. Do ponto de vista do custo computacional, portanto, a fatoração estocástica *não* parece ser uma alternativa viável para se solucionar um processo de Markov.

[8] Além disso, pode não fazer muito sentido realizar a clusterização em um espaço de alta dimensionalidade [23, 60].

[9] Para uma discussão semelhante, embora em um contexto bastante diferente, sugiro a leitura do interessantíssimo ensaio de Stephen Jay Gould [56] intitulado "Tamanho e Forma."

3.4 Redução de um Processo de Decisão de Markov

Se a fatoração estocástica não constitui uma estratégia muito atraente para solucionar um processo de Markov, a idéia de incorporá-la à solução de um MDP parece pouco promissora. Surpreendentemente, no entanto, existem situações em que isso pode significar de fato uma redução de custo computacional. Em contraste com o problema de se calcular a função de valor de um processo de Markov, cuja solução pode ser encontrada dentro de limites de computação bem conhecidos, a complexidade computacional dos algoritmos para solucionar um MDP ainda é uma questão em aberto [92]. Sabe-se que a busca por uma política ótima de um MDP pode ser formulada como um problema de programação linear, e portanto pode ser teoricamente efetuada em tempo polinomial [128, 22]. No entanto, o grau desse polinômio ainda não é conhecido, e sabe-se que ele é grande o suficiente para tornar os algoritmos de programação linear muito ineficientes na prática [92]. Da mesma forma, os métodos de programação dinâmica desenvolvidos especificamente para resolver um MDP requerem em alguns casos um número de iterações proibitivamente grande para convergir. Littman *et al.* [92] mostram um exemplo de MDP em que o número de iterações executadas pelo algoritmo de iteração de valor para encontrar a política ótima π^* cresce mais rápido do que $(1-\gamma)^{-1}$. Um exemplo parecido é apresentado por Melekopoglou e Condon [97] para uma variedade específica do algoritmo de iteração de política. Os autores mostram uma família de MDPs em que esse algoritmo avalia $2^{|S|/2-2}$ políticas de decisão antes de chegar em π^*. Como discutido na Seção 2.2.2, Mansour e Singh [93] apresentaram uma cota superior de $O(2^{|S|}/|S|)$ para o número de passos executados pelo algoritmo de iteração de política convencional.

Os exemplos acima deixam claro que o número de iterações executadas pelos algoritmos de programação dinâmica pode inviabilizar a sua aplicação na prática. O processo de avaliação de uma política é $O(|S|^3)$ e a sua melhoria pode ser efetuada em $O(|S|^2|A|)$ operações aritméticas. Como discutido no Capítulo 2, os detalhes de como esses dois passos são implementados define um espectro de algoritmos que tem a iteração de política em um extremo e a iteração de valor no outro. À medida que se move do primeiro em direção ao segundo o custo por iteração cai, mas o seu número tende a aumentar. Se a multiplicação do número de iterações pelo seu custo for suficientemente grande, o

uso de $m << |S|$ arquétipos no lugar dos estados originais do problema pode gerar uma economia de tempo de computação que justifique o custo extra para se determinar esses arquétipos. Obviamente, isso só é possível se existir um conjunto de arquétipos a partir dos quais seja possível recuperar um processo de Markov $\mathbf{M}^\pi$ derivado de *qualquer* política π definida no MDP. Assim, ao invés de realizar uma fatoração estocástica para cada processo $\mathbf{M}^\pi$ que surgir na iteração de política generalizada, pode-se fatorar estocasticamente o próprio MDP.

Sejam $\mathbf{P}^a$ e $\mathbf{r}^a$, com $a \in A$, as matrizes de transições e os vetores-recompensa de um MDP. Então, concatenando esses elementos como feito em (3.44) pode-se falar em processos de Markov $\mathbf{M}^a$ associados com cada ação do MDP em questão. O interessante é que a partir desses processos é possível obter qualquer processo de Markov $\mathbf{M}^\pi$ induzido por uma política π definida no MDP. Em particular, a i-ésima linha de $\mathbf{M}^\pi$ corresponde à i-ésima linha do processo $\mathbf{M}^{\pi(s_i)}$, ou seja,

$$\mathbf{m}_i^\pi = \mathbf{m}_i^{\pi(s_i)}, \text{ com } \pi(s_i) \in A,$$

onde $\pi(s_i)$ é a ação selecionada por π no estado s_i. Suponha agora que haja uma matriz $\mathbf{W}$ e matrizes $\mathbf{D}^a$ tais que $\mathbf{M}^a = \mathbf{D}^a\mathbf{W}$ para todo a. Não é difícil perceber que o processo de Markov $\mathbf{M}^\pi$ associado com uma política π poderia ser representado a partir de $\mathbf{W}$. Se a matriz $\mathbf{W}$ for "semi-estocástica" e todas as matrizes $\mathbf{D}^a$ forem estocásticas, a Proposição 3.1 é diretamente aplicável a $\mathbf{M}^\pi$.

Resta saber como calcular $\mathbf{W}$ e $\mathbf{D}^a$. Para tal, defina a matriz $\mathbf{M}$ de dimensões $|S||A| \times |S| + 1$ "empilhando" todas as matrizes $\mathbf{M}^a$ (veja (3.44)):

$$\mathbf{M} = \begin{bmatrix} r_1^1 & p_{11}^1 & p_{12}^1 & \cdots & p_{1|S|}^1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ r_{|S|}^1 & p_{|S|1}^1 & p_{|S|2}^1 & \cdots & p_{|S||S|}^1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ r_1^{|A|} & p_{11}^{|A|} & p_{12}^{|A|} & \cdots & p_{1|S|}^{|A|} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ r_{|S|}^{|A|} & p_{|S|1}^{|A|} & p_{|S|2}^{|A|} & \cdots & p_{|S||S|}^{|A|} \end{bmatrix}. \tag{3.45}$$

Então, as matrizes $\mathbf{W}$ e $\mathbf{D}^a$ podem ser obtidas a partir de qualquer fatoração estocástica

$\mathbf{M} = \mathbf{DW}$ em que $m \leq |S|$. Em particular, a i-ésima linha de $\mathbf{D}^a$ é dada por

$$\mathbf{d}_i^a = \mathbf{d}_{(j-1)|A|+i}, \tag{3.46}$$

onde j é a ordem em que $\mathbf{M}^a$ foi empilhado em $\mathbf{M}$, ou seja, j é a linha que o vetor $\mathbf{m}_1^a$ ocupa na matriz $\mathbf{M}$.

Uma vez de posse de $\mathbf{W}$ e $\mathbf{D}^a$, pode-se usá-las no cálculo da função de valor de cada política que eventualmente surja no processo de iteração de política generalizada. Dada uma política π qualquer, $\mathbf{D}^\pi$ é definida de forma análoga à $\mathbf{M}^\pi$, simplesmente escolhendo como a i-ésima linha dessa matriz a linha correspondente de $\mathbf{D}^{\pi(s_i)}$. A matriz $\bar{\mathbf{P}}^\pi$ pode então ser obtida a partir da multiplicação $\mathbf{KD}^\pi$, onde $\mathbf{K}$ é a matriz $\mathbf{W}$ sem a primeira coluna. Chamando esta primeira coluna de $\bar{\mathbf{r}}$, pode-se definir um processo de Markov com m estados, exatamente como na Seção 3.2. Aplicando a Proposição 3.1 é possível obter a função de valor do processo original simplesmente fazendo $\mathbf{v}^\pi = \mathbf{D}^\pi \bar{\mathbf{v}}^\pi$, onde $\bar{\mathbf{v}}^\pi$ é a função do processo descrito por $\bar{\mathbf{P}}^\pi$ e $\bar{\mathbf{r}}$. A partir de $\mathbf{v}^\pi$ é possível derivar uma nova política, cuja função de valor pode ser calculada usando-se os mesmos m arquétipos, e assim por diante. O Algoritmo 3.2 traz uma descrição resumida desse processo. Por motivos óbvios irei me referir a esse algoritmo como *iteração de política baseada na fatoração estocástica* (PISF[†]).

É interessante notar que a função $\bar{\mathbf{v}}^\pi$ não precisa ser calculada de maneira exata no Algoritmo 3.2. A princípio, a avaliação de política poderia ser realizada de maneira aproximada, como acontece no Algoritmo 2.2. No entanto, é justamente no cômputo indireto de $\mathbf{v}^\pi$ através da função $\bar{\mathbf{v}}^\pi$ que o algoritmo PISF obtém uma grande economia computacional, com uma redução do custo de $O(|S|^3)$ para $O(m^3)$. Como em princípio quanto mais preciso o cálculo de $\mathbf{v}^\pi$ mais efetiva é a melhoria de política subseqüente, o custo extra de se calcular $\bar{\mathbf{v}}^\pi$ de maneira exata tende a ser compensado pela redução do número total de iterações.

[†] *Policy iteration with stochastic factorization*

Algoritmo 3.2 Iteração de política baseada na fatoração estocástica (PISF)

Requer $\mathbf{P}^a \in \mathbb{R}^{|S|\times|S|}$, $\mathbf{r}^a \in \mathbb{R}^{|S|}$ para cada $a \in A$, $\gamma \in [0,1)$, $m \in [1,|S|]$, $\varepsilon \in \mathbb{R}_*^+$

Retorna $\pi \approx \pi^*$

Obtenha $\mathbf{M}$ "empilhando" as matrizes $\mathbf{M}^a = [\mathbf{r}^a, \mathbf{P}^a]$ (*cf.* (3.45))

Encontre uma fatoração estocástica do tipo $\mathbf{M} \approx \mathbf{DW}$ com m arquétipos

Defina $\bar{\mathbf{r}}$ como a primeira coluna de $\mathbf{W}$ e $\mathbf{K}$ como as demais

Obtenha as matrizes $\mathbf{D}^a$, para todo $a \in A$, usando (3.46)

$\pi \leftarrow$ vetor aleatório em $A^{|S|}$

repita

$\pi' \leftarrow \pi$

para $i \leftarrow 1,2,...,|S|$ **faça** $\mathbf{d}_i^\pi \leftarrow \mathbf{d}_i^{\pi(s_i)}$

$\bar{\mathbf{P}}^\pi \leftarrow \mathbf{K}\mathbf{D}^\pi$

$\bar{\mathbf{v}}^\pi \leftarrow (\mathbf{I} - \gamma\bar{\mathbf{P}}^\pi)^{-1}\bar{\mathbf{r}}$

$\mathbf{v}^\pi \leftarrow \mathbf{D}^\pi\bar{\mathbf{v}}^\pi$

para $i \leftarrow 1,2,...,|S|$ **faça** $\pi(s_i) \leftarrow \text{des}\left[\text{argmax}_a\left(r_i^a + \gamma\sum_j p_{ij}^a v_j^\pi\right)\right]$

até $\| \pi - \pi' \| < \varepsilon$

3.4.1 Análise teórica

No algoritmo PISF exige-se apenas que $\mathbf{M} \approx \mathbf{DW}$. No entanto, caso uma fatoração estocástica exata de $\mathbf{M}$ seja encontrada, *tem-se a garantia de convergência para a política ótima* π^*. Isso é uma característica muito desejável do Algoritmo 3.2 que segue diretamente da Proposição 3.1. Uma questão pertinente nesse caso é se o algoritmo PISF é bem-comportado de uma maneira geral, ou seja, se a política π retornada por ele aproxima-se de π^* à medida que $\| \mathbf{M} - \mathbf{DW} \| \to 0$. Para começar a responder a esta pergunta, apresento uma proposição de Bertsekas e Tsitsiklis [22] que fornece uma cota superior para a diferença entre $\mathbf{v}^*$ e a função $\mathbf{v}^\pi$ retornada por um algoritmo de iteração de política aproximado genérico:

Proposição 3.4. *Seja* $\pi_1, \pi_2, ..., \pi_n$ *a seqüência de políticas geradas por um algoritmo de iteração de política aproximado e suponha que*

$$\| \tilde{\mathbf{v}}^{\pi_i} - \mathbf{v}^{\pi_i} \|_\infty \leq \epsilon_1 \text{ para todo } i = 1,2,...,n \tag{3.47}$$

$$\| \tilde{\Upsilon}\tilde{\mathbf{v}}^{\pi_i} - \Upsilon\tilde{\mathbf{v}}^{\pi_i} \|_\infty \leq \epsilon_2 \text{ para todo } i = 1,2,...,n.$$

Então

$$\limsup_{n\to\infty} \| \tilde{\mathbf{v}}^{\pi_n} - \mathbf{v}^* \|_\infty \leq \frac{\epsilon_2 + 2\gamma\epsilon_1}{(1-\gamma)^2}. \tag{3.48}$$

Demonstração. Bertsekas e Tsitsiklis [22], Proposição 6.2, p. 276. □

Note que a proposição acima cobre o caso em que a melhoria de política não é feita de maneira exata, ou seja, em que utiliza-se um operador aproximado $\tilde{\Upsilon}$ no lugar de Υ (veja equação 2.15). Ao contrário da iteração de política original, a versão aproximada desse algoritmo não tem garantia de convergência para uma política específica. Lembrando que $\mathbf{v}^*$ representa as recompensas acumuladas ao se seguir a política π^*, a Proposição 3.4 anuncia que um algoritmo de iteração de política aproximado eventualmente produzirá políticas π_i cujos desempenhos estão a no máximo uma constante do melhor desempenho possível no MDP. O que determina a magnitude dessa constante são as variáveis ϵ_1 e ϵ_2, que indicam a precisão com que são executadas as fases de avaliação e melhoria de política. Resta saber como o algoritmo PISF se enquadra neste contexto. Essa questão é esclarecida pelo corolário abaixo:

Corolário 3.1. *Suponha que $\pi_1, \pi_2, ..., \pi_n$ seja a seqüência de políticas geradas pelo algoritmo de iteração de política baseada na fatoração estocástica. Então*

$$\limsup_{n\to\infty} \| \tilde{\mathbf{v}}^{\pi_n} - \mathbf{v}^* \|_\infty \leq \frac{2\gamma\upsilon}{(1-\gamma)^3}, \tag{3.49}$$

onde

$$\upsilon = \max_a \| \mathbf{r}^a - \mathbf{D}^a\bar{\mathbf{r}} \|_\infty + \frac{\gamma}{1-\gamma} \max_a \| \mathbf{P}^a - \mathbf{D}^a\mathbf{K} \|_\infty \max_a \| \mathbf{r}^a \|_\infty.$$

Demonstração. O resultado acima é uma composição das Proposições 3.3 e 3.4. Como no algoritmo PISF a melhoria de política é feita de maneira exata, tem-se $\epsilon_2 = 0$ em (3.48). Além disso, para qualquer política π sabe-se que

$$\| \mathbf{r}^\pi - \mathbf{D}^\pi\bar{\mathbf{r}} \|_\infty \leq \max_a \| \mathbf{r}^a - \mathbf{D}^a\bar{\mathbf{r}} \|_\infty$$

$$\| \mathbf{P}^\pi - \mathbf{D}^\pi\mathbf{K} \|_\infty \leq \max_a \| \mathbf{P}^a - \mathbf{D}^a\mathbf{K} \|_\infty$$

$$\| \mathbf{r}^\pi \|_\infty \leq \max_a \| \mathbf{r}^a \|_\infty.$$

Fazendo

$$\epsilon_1 = \frac{1}{1-\gamma}\left(\max_a \| \mathbf{r}^a - \mathbf{D}^a\bar{\mathbf{r}} \|_\infty + \frac{\gamma}{1-\gamma} \max_a \| \mathbf{P}^a - \mathbf{D}^a\mathbf{K} \|_\infty \max_a \| \mathbf{r}^a \|_\infty\right) \tag{3.50}$$

sabe-se a partir de (3.31) que (3.47) é verdadeira. Substituindo (3.50) em (3.48) tem-se (3.49). □

Considerando os experimentos da Seção 3.3.2, a relevância prática da cota (3.49) pode ser questionada. Mesmo assim, a *existência* de uma cota superior para a diferença $\| \tilde{\mathbf{v}} - \mathbf{v}^* \|_\infty$ é de uma importância fundamental. Basicamente, ela significa que o desempenho da política gerada pelo algoritmo PISF depende unicamente da qualidade da fatoração do MDP. Isso é uma garantia de que um erro na aproximação de $\mathbf{M}$ não resultará em uma divergência da seqüência $\{\mathbf{v}^{\pi_1}, \mathbf{v}^{\pi_2}, ...\}$ em relação ao vetor $\mathbf{v}^*$. Em outras palavras, a magnitude do erro de aproximação de $\mathbf{M}$ define uma região em torno de $\mathbf{v}^*$ na qual a seqüência acima eventualmente estará confinada.

Esse tipo de garantia ilustra uma vantagem em se aproximar o MDP ao invés da função de valor (veja discussão na Seção 2.4). Quando a função V^π é aproximada diretamente, é possível que um erro na estimativa dessa função seja reincorporado ao processo de aprendizagem, criando um ciclo de realimentação que acabe por ocasionar a divergência da seqüência $\{\mathbf{v}^{\pi_1}, \mathbf{v}^{\pi_2}, ...\}$. Mesmo nos casos em que essa seqüência converge garantidamente, é muito difícil estabelecer qualquer garantia em relação à região de convergência, porque em geral não se conhece de antemão uma cota superior para o erro de aproximação de $\mathbf{v}^\pi$.

Com o algoritmo PISF é possível obter uma política π tão próxima de π^* quanto se queira, bastando para isso melhorar a fatoração aproximada $\mathbf{DW} \approx \mathbf{M}$. Sabe-se que existe uma tolerância ϵ tal que se $\| \tilde{\mathbf{v}}^\pi - \mathbf{v}^\pi \|_\infty < \epsilon$ a política gerada com base em $\tilde{\mathbf{v}}^\pi$ é a mesma que seria gerada a partir de $\mathbf{v}^\pi$(veja por exemplo a Proposição 6.1 de Bertsekas e Tsitsiklis [22]). Portanto, se o valor da variável ϵ_1 em (3.50) for suficientemente pequeno, o algoritmo PISF converge para π^* mesmo com $\mathbf{DW} \neq \mathbf{M}$. Obviamente, o valor de ϵ_1 que garante um desempenho ótimo em geral não é conhecido. Na próxima seção avalio empiricamente o efeito de um erro de aproximação arbitrário sobre a qualidade das políticas geradas pelo algoritmo PISF.

3.4.2 Análise empírica

Na Seção 3.3.2 eu mostrei como as características da matriz de transições $\mathbf{P}^\pi$ e do vetor de recompensas $\mathbf{r}^\pi$ influenciam a fatoração estocástica do processo de Markov $\mathbf{M}^\pi$,

e como esta última tem um impacto sobre o cálculo da função de valor aproximada $\tilde{\mathbf{v}}^{\pi}$. Nesta seção dou um passo adiante, e discuto as conseqüências de se usar aproximações da função de valor no processo de iteração de política generalizada. Em particular, estou interessado em avaliar o impacto de se adotar $\tilde{\mathbf{v}}^{\pi_i}$ no lugar de $\mathbf{v}^{\pi_i}$ sobre a qualidade da política de decisão π_n encontrada ao final do processo. Para tal, é conveniente definir uma medida formal da qualidade de uma política π. Uma escolha óbvia nesse caso é a seguinte:

$$\chi_m(\pi) = \frac{1}{|S|}\sum_{i=1}^{|S|}\delta(\pi(s_i), \pi^*(s_i)), \text{ onde } \delta = \begin{cases} 0, \text{ se } \pi(s_i) = \pi^*(s_i) \\ 1, \text{ caso contrário.} \end{cases}$$

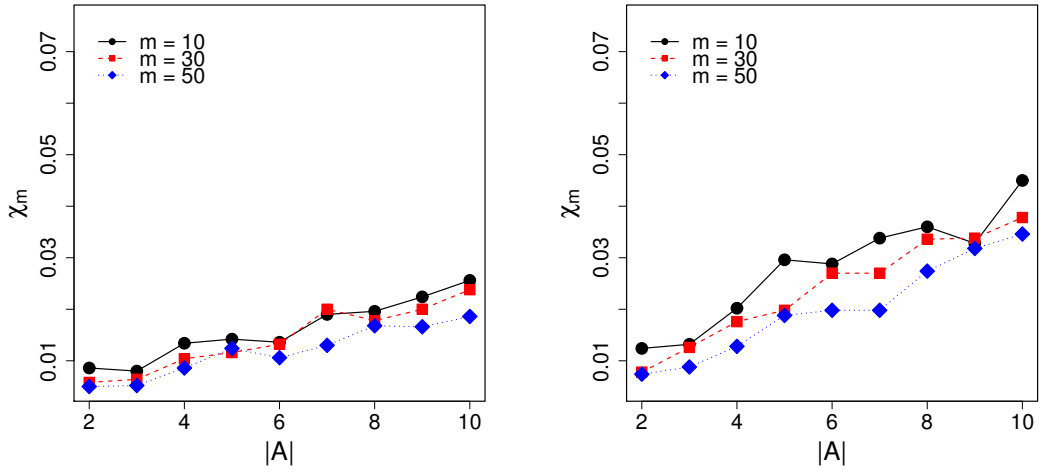
Vários fatores influenciam a dificuldade de se fatorar estocasticamente um MDP, como o número m de arquétipos usados na fatoração, o fator de desconto γ e as características das matrizes de transições $\mathbf{P}^a$ e dos vetores-recompensa $\mathbf{r}^a$. Como a busca por uma política ótima de um MDP pode ser vista como a solução de uma sucessão de processos de Markov, a expectativa é que esses parâmetros tenham sobre o primeiro o mesmo impacto que têm sobre os últimos, já estudados na Seção 3.3.2. Porém, quando se lida com um processo de decisão de Markov existe uma dimensão extra no problema: a cardinalidade $|A|$ do conjunto de ações possíveis (para facilitar, considerarei que as mesmas ações estão disponíveis em todos os estados, como no Capítulo 2). A expectativa é que o valor da função χ_m aumente quando $|A| \to \infty$. Para enxergar isso, basta notar que a chance de acertar ao acaso a ação ótima no estado s_i é $1/|A|$.

Nos experimentos desta seção foram gerados MDPs com diferentes valores para o parâmetro $|A|$. Para tal, adotei a mesma estratégia da Seção 3.3.2 para obter $|A|$ matrizes $\mathbf{P}^a$ e vetores $\mathbf{r}^a$ a cada MDP gerado. Note que o parâmetro ϑ tem um efeito um pouco diferente na construção de um MDP: como cada matriz $\mathbf{P}^a$ é gerada de maneira independente, é muito pouco provável que as suas linhas estejam na vizinhanças dos *mesmos* ϑ vértices de $\Delta^{|S|-1}$. Isso dificulta consideravelmente o problema em relação à aproximação de um processo de Markov isolado. A Figura 3.11 mostra o desempenho do algoritmo PISF combinado com o k-means na aproximação de um conjunto de MDPs. Note que o efeito de $|A|$, γ e m sobre χ_m corresponde às expectativas. Em particular, χ_m parece ser uma função linear de $|A|$ cuja inclinação é determinada (ou pelo menos influenciada) pelo

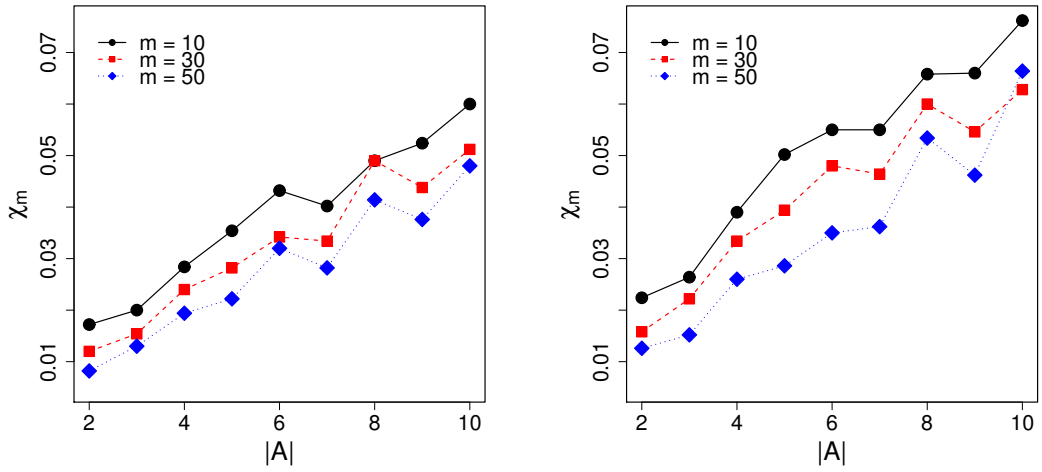
fator de desconto γ: quanto maior esse parâmetro, maior o coeficiente angular da reta. Ao contrário dos casos estudados anteriormente, aqui faz sentido comentar a respeito da magnitude da função de custo. Observe que no pior caso, que corresponde a $m = 10$, $\gamma = 0.9$ e $|A| = 10$ (Figura 3.11d), o número médio de estados em que uma ação sub-ótima é escolhida gira em torno de 7.5. O desvio-padrão associado com esse resultado é de aproximadamente 3.28, indicando uma variação de pouco mais de 3 estados para mais ou para menos. Considerando todas as configurações de parâmetros testadas nos experimentos, o pior resultado encontrado nas 50 execuções do algoritmo PISF foi $\chi_m = 0.19$.

Uma questão pertinente que pode ser levantada se refere à validade da função de custo χ_m como critério para a avaliação das políticas geradas pelo algoritmo PISF. Isso porque os "erros" cometidos por uma política π podem ter pesos diferentes dependendo do estado s_i onde eles ocorreram: como um exemplo extremo, imagine o caso em que o MDP seja um problema de menor caminho e π selecione a ação ótima em todos os estados, com exceção justamente daqueles que levam para o estado terminal. Nesse caso π teria uma avaliação idêntica à de uma política que cometesse o mesmo número de erros em regiões menos importantes do espaço de estados S. Uma medida de custo mais informativa seria a diferença média no total de recompensas acumuladas por π e π^*, ou seja, $\xi_m(\mathbf{v}^*, \mathbf{v}^\pi)$. Como nos MDPs gerados para os experimentos desta seção as recompensas são mais ou menos uniformes, a função $\xi_m(\mathbf{v}^*, \mathbf{v}^\pi)$ apresenta um comportamento parecido com o de $\chi_m(\pi^*, \pi)$, e portanto faria pouco sentido discuti-la em detalhes. Apenas para se ter uma idéia, nos experimentos mostrados na Figura 3.11 $\xi_m(\mathbf{v}^*, \mathbf{v}^\pi) < 10^{-5}$ para $\gamma \leq 0.7$. Para o caso em que $m = 10$, $\gamma = 0.9$ e $|A| = 10$ essa diferença é de aproximadamente 2×10^{-4}, e o pior resultado encontrado nas 50 execuções foi $\xi_m(\mathbf{v}^*, \mathbf{v}^\pi) \approx 2.56 \times 10^{-3}$.

A avaliação da qualidade dos resultados obtidos pelo algoritmo PISF depende do fenômeno modelado pelo MDP. Se o processo de decisão em questão fosse o jogo de golfe, como no exemplo do começo deste capítulo, as políticas encontradas pela iteração de política baseada na fatoração estocástica provavelmente seriam competitivas. Por outro lado, em outros cenários o desempenho dessas políticas poderia não ser satisfatório. Quando se fala em um sistema controlador de vôo ou em um robô-cirurgião, por exemplo, é pouco provável que uma taxa de erro de $7,5\%$ seja aceitável. É bom lembrar que a proposta de redução de um MDP apresentada aqui se destina aos casos em que o custo computacional



(a) $\gamma = 0.3$ (b) $\gamma = 0.5$



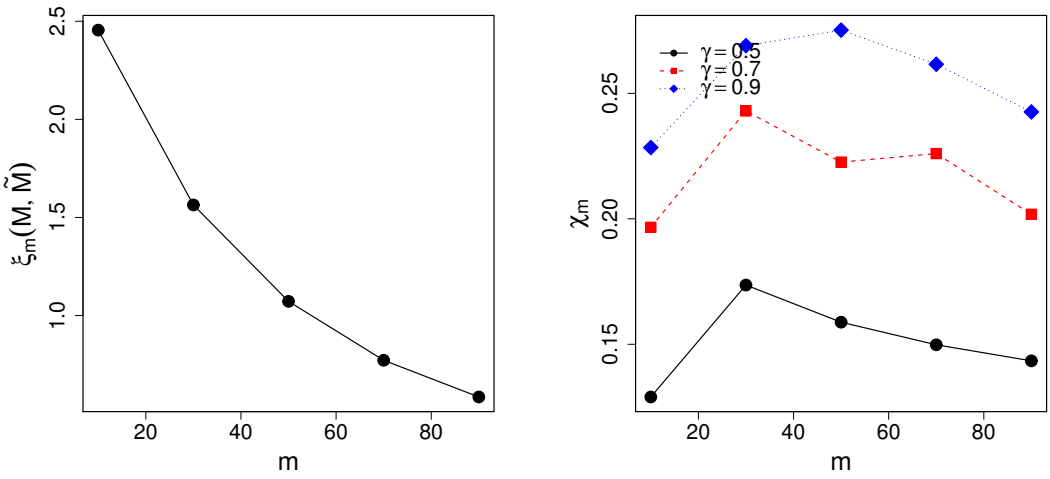
(c) $\gamma = 0.7$ (d) $\gamma = 0.9$

Figura 3.11: Diferença entre a política ótima π^* e a política encontrada pelo algoritmo PISF (*cf.* Algoritmo 3.2). A fatoração estocástica dos processos de Markov foi feita pelo algoritmo k-means. As matrizes de transições $\mathbf{P}^a$ foram geradas com $\vartheta = 50$, $\sigma = 5$ e $\eta = 0$ e os vetores-recompensa $\mathbf{r}^a$ foram obtidos como nos experimentos anteriores. Os valores se referem a uma média de 50 execuções do algoritmo PISF.

de se resolver o problema diretamente é proibitivo, ou seja, em que o desempenho ótimo não é alcançável com os recursos computacionais disponíveis. Sendo assim, a comparação com uma política ótima talvez não seja adequada; melhor seria contrastar o desempenho do algoritmo PISF com o de outras técnicas de programação dinâmica aproximada. Isso será feito no Capítulo 4.

O leitor atento terá notado uma diferença neste último experimento em relação aos anteriores. Trata-se do desvio-padrão usado na geração das matrizes de transições, que vinha sendo fixado em $\sigma = 1$ e aqui assumiu o valor de $\sigma = 5$. A razão para isso é que um estranho fenômeno ocorre com os resultados do algoritmo PISF quando $\sigma = 1$. Ao contrário do esperado, nesse caso a função de custo χ_m não decresce monotonicamente com o aumento de m. Como mostra a Figura 3.12b, o valor dessa função aumenta quando m muda de 10 para 30, e no caso em que $\gamma = 0.9$ essa tendência continua até $m = 50$. Em todos os casos o mínimo é atingido com $m = 10$. Comportamentos parecidos podem ser observados para todos os valores de $|A| \in \{2, ..., 9\}$. O que torna esse fenômeno particularmente intrigante é o fato de a função $\xi_m(\mathbf{M}, \mathbf{DW})$ se comportar de acordo com a expectativa, como mostra a Figura 3.12a. Isso significa que um erro de aproximação de $\mathbf{M}$ tende a ser menos danoso quando $m \approx 10$. Essa tendência foi observada de forma consistente em vários experimentos realizados com $\sigma = 1$. Acredito que a investigação das causas desse fenômeno possa revelar novos aspectos da fatoração estocástica como estratégia de redução de um MDP. Intuitivamente me parece que a explicação está relacionada com a natureza da aproximação realizada pelo algoritmo k-means, mas esse assunto precisa ser analisado com mais cuidado. Vale lembrar que quando $\sigma \geq 5$ tanto a função ξ_m quanto χ_m se comportam exatamente como esperado.

Antes de encerrar, não posso deixar de fazer um comentário a respeito dos experimentos desta seção. Como discutido, a iteração de política baseada na fatoração estocástica significa um ganho computacional em relação à versão original do algoritmo apenas quando o número de iterações executadas por este último é muito grande. Esse não é o caso dos MDPs gerados aqui. Em geral a política ótima desses MDPs foram encontradas em um pequeno número de iterações, nunca mais do que 20. Isso significa que nos exemplos mostrados o uso da fatoração estocástica não resultou em nenhuma economia de tempo de processamento. Ironicamente, como observam Mansour e Singh [93], é difí-



(a) $\xi_m(\mathbf{M}, \tilde{\mathbf{M}}^\pi)$ (b) $\chi_m(\pi^*, \pi)$

Figura 3.12: Ilustração do comportamento do algoritmo PISF combinado com o algoritmo k-means quando $\sigma = 1$. Com exceção do desvio-padrão σ, os MDPs foram gerados exatamente como descrito na Figura 3.11, com $|A| = 10$. Os valores se referem a uma média de 50 execuções do algoritmo PISF.

cil na prática construir MDPs em que a iteração de política leve mais do que $|S|$ iterações para encontrar π^*. Isso levanta a pergunta se a questão do custo computacional ocasionado pela explosão do número de iterações não é mais teórica do que prática. Se de fato existirem MDPs de interesse cuja solução requeira um número grande de iterações, então é necessário criar métodos para identificá-los. Caso contrário o uso do Algoritmo 3.2 ficaria condicionado ao resultado de um processo de tentativa-e-erro.

3.5 Resumo

Neste capítulo eu apresentei uma estratégia para a redução de um processo de decisão de Markov baseada na intuição de que as transições dos estados originais do problema podem ser redirecionadas para estados fictícios, chamados de arquétipos, que representam bem a dinâmica do MDP. Como discutido, esse redirecionamento de transições pode ser formalizado como a fatoração de uma matriz de transições $\mathbf{P}^\pi$ em duas matrizes estocásticas $\mathbf{D}$ e $\mathbf{K}$. O apelo da fatoração estocástica é que a multiplicação $\mathbf{KD}$ dá origem

a uma matriz de transições válida, e portanto a um processo de Markov potencialmente muito menor do que o original. Dependendo do vetor-recompensa desse processo reduzido, é possível recuperar a partir da sua função de valor a função do processo original, o que pode significar uma imensa economia de tempo de processamento. Como mostrado nas análises deste capítulo, a redução de um MDP baseada na fatoração estocástica é um processo bem fundamentado teoricamente, tanto na sua versão exata quanto na sua versão aproximada. Isso significa que ao se fatorar estocasticamente um MDP tem-se a garantia de que a função de valor encontrada para o problema reduzido estará na vizinhança da função ótima do problema original. O raio dessa vizinhança é determinado unicamente pela magnitude do erro de fatoração.

As observações acima deixam claro que pelo menos em princípio a fatoração estocástica seria uma alternativa viável para reduzir as dimensões de um MDP—ou seja, para diminuir o tempo de processamento necessário para encontrar uma política razoável para o problema. O único obstáculo para a sua aplicação na prática é que o processo de fatoração estocástica de uma matriz tem ele mesmo um alto custo computacional, tão alto que muitas vezes a economia de computação posterior não se justifica. No caso de um processo de Markov isolado, a fatoração estocástica dificilmente pode ser considerada como uma alternativa viável para o cálculo da função de valor. Já no caso de MDPs, existem situações em que ela pode significar de fato uma redução do custo computacional. Isso ocorre quando o número de iterações executadas pelos algoritmos de programação dinâmica é muito grande. Embora seja possível encontrar na literatura exemplos de MDPs artificiais cuja solução requer um número exponencial de iterações por parte de alguns algoritmos, a importância desse fenômeno na prática ainda precisa ser verificada.

Existem duas possibilidades de extensão do assunto desenvolvido neste capítulo. Uma delas é tentar identificar quais as características de um MDP que acarretam uma explosão do número de iterações necessárias para resolvê-lo. Se esta empreitada for bem sucedida, tem-se não somente como avaliar a freqüência com que esse fenômeno ocorre na prática, como também uma forma de identificar MDPs suscetíveis à fatoração estocástica. Uma outra alternativa, mais promissora na minha opinião, seria tentar acelerar os algoritmos usados para fatorar estocasticamente um processo de Markov. Como discutido, a causa principal do alto custo computacional desses algoritmos é o que chamei de maldição do

crescimento superficial. Seria possível "quebrar" essa maldição? Esse é o assunto do próximo capítulo.

Capítulo 4

Fatoração Estocástica no Espaço de Estados

"You can never solve a problem on the level on which it was created."

Albert Einstein

O filósofo Platão apresenta no livro VII de sua obra *República* uma parábola que ficou conhecida como "a alegoria da caverna." Imagine um grupo de homens presos em uma caverna desde o seu nascimento, de tal maneira que eles mal possam mover os seus pescoços para olhar ao redor. Esses homens estão de costas para uma fresta por onde atravessa um feixe de luz. Tudo o que vêem, portanto, são as sombras daquilo que se passa lá fora projetadas sobre a parede à sua frente. Em particular, esse homens jamais contemplaram a si mesmos ou a outro ser humano. Para eles, o bruxuleio fantasmagórico das sombras são a única realidade que existe.

Platão usou a parábola acima como uma metáfora para o conhecimento humano: de acordo com ele, o mundo sensorial seria a caverna de todos os homens, e a essência da realidade seria descoberta através da luz da dialética, apenas. Assim como um mesmo objeto pode gerar sombras das mais variadas formas e tamanhos, a variabilidade das coisas seria o efeito de uma percepção distorcida de um mundo ideal. Existiria na programação dinâmica o equivalente a esse espaço platônico?

A programação dinâmica trata da resolução de processos de decisão de Markov, mas a solução de um problema de tomada de decisão seqüencial inicia-se em uma etapa anterior, que é a formulação do próprio MDP. Embora isso não seja evidente a princípio, um processo de decisão de Markov traz embutida uma série de suposições a respeito do fenômeno que pretende modelar. Uma delas tem um caráter intrinsecamente subjetivo: o esquema de recompensas que descreve as expectativas do projetista em relação ao problema a ser resolvido. Em um nível mais elementar, o MDP carrega em si uma categorização do fenômeno em questão. Isso porque a determinação das probabilidades de transições entre estados depende inexoravelmente do *conceito* do que seria um estado do problema.

A definição da informação que irá compor cada estado do modelo é um passo fundamental na formalização de um problema de tomada de decisão. É a partir da caracterização de um estado que as matrizes de transições de um MDP são determinadas, e são estas últimas que definem a "estrutura" do sistema de recompensas que reflete os objetivos do problema. Esta afirmação me parece bem óbvia. O que talvez não seja tão evidente é que do ponto de vista da programação dinâmica a determinação das transições e das recompensas é o *único* aspecto relevante na formalização de um problema de tomada de decisão seqüencial. Isso significa que toda a informação relevante a respeito de um determinado problema fica retida na descrição do MDP. Assim, dois conceitos de estado diferentes que originem o mesmo processo de decisão de Markov são, sob a perspectiva da aprendizagem, totalmente equivalentes. Seriam, portanto, versões distorcidas de um modelo ideal—as "sombras" de um MDP.

Em geral, após a definição de um MDP a informação que lhe deu origem pode ser simplesmente descartada. No entanto, em alguns casos pode ser vantajoso do ponto de vista computacional manipular diretamente uma descrição dos estados do sistema. Isso porque a dimensão do espaço de estados não está sujeita à "maldição do crescimento superficial" discutida no final do capítulo anterior. Essa pode, portanto, ser uma alternativa para contornar a explosão do custo computacional que ocorre quando se trabalha diretamente com probabilidades de transições e recompensas. Neste capítulo discuto em detalhes como isso é possível.

Inicio o capítulo descrevendo o processo de modelagem envolvido na programação dinâmica. Em particular, descrevo como um problema de tomada de decisão seqüencial pode ser convertido em um MDP. Um passo fundamental desse processo é a identificação e formalização do espaço de estados S. Como será discutido, se S for munido com uma métrica que atenda a certas condições, é possível manipular o MDP a partir desse espaço—o que do ponto de vista prático se traduz em uma grande economia de tempo de processamento. A partir desta constatação, apresento na Seção 4.2 uma formulação do algoritmo PISF em que a fatoração estocástica é realizada no espaço de estados. Como será visto, essa versão do algoritmo não está sujeita ao crescimento superficial. Seguindo a mesma linha de raciocínio, na Seção 4.3 discuto uma técnica de fatoração estocástica que pode ser aplicada a partir de um conjunto de amostras de transições. Isso dá origem a um poderoso algoritmo, que eu chamo de *fatoração estocástica baseada em kernels*. Assim como no capítulo anterior, as discussões deste capítulo são embasadas por experimentos computacionais simples. Para que se tenha uma idéia melhor da efetividade das abordagens propostas, na Seção 4.4 discuto a sua aplicação a um problema mais desafiador. Nessa seção os algoritmos apresentados são comparados com outras técnicas de aprendizagem por reforço em um problema em que o objetivo é manter equilibrado um ou dois bastões. A Seção 4.5 traz uma breve revisão e algumas conclusões a respeito do assunto tratado neste capítulo.

4.1 A Modelagem na Programação Dinâmica

No contexto da programação dinâmica, *modelagem* se refere à formulação de um MDP que descreva o fenômeno de interesse. Esse processo tem início com a definição da informação que irá compor um estado do modelo. A partir da formulação dos estados do sistema é possível determinar as probabilidades de transições entre eles. Essa etapa do processo de modelagem é, pelo menos a princípio, totalmente objetiva: as matrizes de transições seguem automaticamente da dinâmica do problema. O último passo na formulação de um MDP é justamente aquele que incorpora a maior carga de subjetividade. É a partir da definição dos vetores-recompensa do MDP que o projetista traduz os seus objetivos em relação ao problema.

4.1.1 O conceito de estado

A definição da informação que irá compor um estado do MDP reflete o grau de abstração com que se quer (ou se pode) lidar com um problema. Embora em geral essa questão não seja discutida na literatura sobre programação dinâmica e aprendizagem por reforço, ela constitui um passo fundamental no processo de solução de um problema de tomada de decisão seqüencial. Na programação dinâmica, uma certa quantidade de informação constitui um estado válido se ela permite a determinação das probabilidades de transição para outros estados, bem como das recompensas associadas a essas transições. Menos informação do que isso acarreta a existência de estados ocultos (ou seja, o processo de decisão deixa de ser markoviano). Mais informação pode tornar o MDP determinístico.

Talvez esta discussão fique mais clara com um exemplo ilustrativo. Para tal, convido o leitor de volta ao campo de golfe do início do capítulo anterior. Como se sabe, o objetivo do golfe é tentar levar a bolinha ao buraco no menor número de tacadas possível. Imagine um cenário em que a única decisão relevante a cada jogada seja a escolha do taco a ser usado: dados a posição da bolinha e o taco escolhido, suponha que seja possível determinar com bastante precisão a probabilidade de a bolinha terminar em cada setor do campo. A dificuldade do jogo nesse caso decorre da escolha entre uma tacada mais conservadora e uma mais arriscada, que no entanto deixe a bolinha mais próxima do buraco. Considere que o objetivo do jogo seja simplesmente minimizar o número total de tacadas, independentemente do adversário. No entanto, para tornar o desafio mais interessante, suponha que o jogador receba um bônus caso o número de tacadas executadas esteja abaixo do esperado—ou seja, abaixo do *par* do buraco.

Existem várias definições possíveis do que seria cada estado do modelo nesse caso, e cada uma delas leva a uma formulação diferente do problema. Se um jogador sabe a sua posição no campo, o número de tacadas executadas até chegar ali e o *par* do buraco atual, então tem-se um processo de decisão de Markov, em que a escolha de uma ação em um determinado estado resulta em uma distribuição de probabilidades sobre recompensas e estados futuros. Se, por outro lado, o jogador não souber o *par* do buraco, então o problema não é mais markoviano: a mesma tacada na mesma posição do campo pode levar a probabilidades diferentes de se conseguir o bônus, dependendo da informação

oculta no estado. Essa falta de informação claramente prejudica a decisão do jogador sobre quando arriscar ou não. Finalmente, se além do *par* o jogador tem consciência de todas as variáveis que influenciam uma tacada—como condições do vento, do campo, *etc.*—então o processo se torna um MDP determinístico.

Seria possível, em qualquer situação, criar um modelo determinístico com a quantidade certa de informação? Isso é uma questão que ocupa filósofos e pensadores desde os tempos de Laplace, que imaginou a existência de um intelecto (mais tarde identificado com um "demônio") capaz de prever o futuro com perfeição [84]. Felizmente, os requisitos formais impostos pela programação dinâmica não dependem de uma solução para essa questão filosófica. Os processos de decisão de Markov são uma versão amena do paradigma de Laplace: enquanto neste último o conhecimento de um estado revela o próximo estado do sistema com exatidão, um MDP requer apenas uma distribuição de probabilidades sobre possíveis estados futuros.

Diante desta discussão, surge a pergunta se existiriam regras gerais para orientar o processo de definição de um estado na solução de um problema de tomada de decisão seqüencial. Uma sistematização do processo de formulação de um estado é desejável, mas muito difícil, uma vez que essa questão depende em grande medida do domínio da aplicação. O que se pode colocar como regra geral é que a informação que irá constituir cada estado do modelo deve ser definida de forma a minimizar a incerteza associada com transições e recompensas, ou seja, tal que o MDP seja tão "laplaciano" quanto possível. A razão para isso é simples: estratégias de decisão baseadas em modelos mais precisos do mundo tendem a apresentar um desempenho melhor na tarefa real.

4.1.2 Um exemplo de derivação de um MDP

Até este ponto discuti a questão da modelagem na programação dinâmica de uma maneira bastante genérica. Para que a discussão não se perca em abstrações e exemplos anedóticos, nesta seção eu mostro como o processo de modelagem ocorre concretamente. Para tal apresento um problema que será usado neste capítulo para ilustrar diversos conceitos. Imagine um carro em um vale, como mostra a Figura 4.1. O objetivo do problema

é dirigir o carro para fora do vale. No entanto, o motor do veículo não é forte o suficiente para levá-lo diretamente até o topo da montanha à sua frente, e a única maneira de conseguir chegar ao objetivo é usar a outra encosta do vale para ganhar alguma velocidade. Trata-se portanto de um problema em que é necessário primeiramente afastar-se do objetivo para só então ser possível alcançá-lo. Isso cria dificuldades para algumas técnicas de controle convencionais, que muitas vezes precisam da ajuda explícita de um ser humano para resolver o problema [161].

A cada instante de tempo t existem três ações a_t possíveis no problema do carro preso no vale: acelerar à frente, dar marcha-à-ré ou permanecer em ponto-morto. Essas ações correspondem aos valores 1, -1 e 0 na equação que descreve a variação da velocidade $\dot{x}$ do automóvel:

$$\dot{x}_{t+1} = \max(\min(\dot{x}_t + 0.001a_t - 0.0025\cos(3x_t), 0.07), -0.07), \tag{4.1}$$

onde x_t é a posição do carro no instante t, calculada como $x_t = x_{t-1} + \dot{x}_{t-1}$. Note que $-0.07 \leq \dot{x} \leq 0.07$. Para tornar o desafio mais interessante, a cada aplicação de (4.1) foi acrescentado à variável a_t um ruído advindo de uma distribuição normal com média zero e desvio-padrão σ. Essa estratégia permite o controle do "nível de estocasticidade" do problema: em particular, quando $\sigma = 0$ o problema torna-se determinístico, como em sua versão original. Seguindo Sutton e Barto [161], a posição do carro ficou restrita ao intervalo $[-1.2, 0.5]$. A tarefa foi considerada um sucesso quando $x \geq 0.5$; todas as vezes em que $x < -1.2$, o veículo foi reposicionado em $x = -1.2$ com velocidade zero.

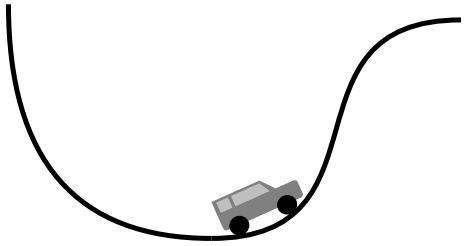


Figura 4.1: O problema do carro preso no vale.

Como discutido, o primeiro passo para aplicar a programação dinâmica a um problema de tomada de decisão seqüencial é definir a informação que irá constituir um estado do modelo. No exemplo acima a abordagem mais direta é considerar que um estado é composto pela posição e pela velocidade do carro. Isso dá origem a um espaço de estados

contínuo bidimensional. Note que essa é uma situação específica, em que se sabe exatamente quais são as variáveis envolvidas nas equações de movimento. Isso permite a formulação do que pode ser considerado um estado "ideal." Por um lado, se uma das variáveis x ou $\dot{x}$ for descartada, o modelo deixa de ser markoviano. Por outro, o acréscimo de uma nova variável ao modelo não traria nenhum benefício para a derivação de uma política de decisão. É importante notar que nem sempre a modelagem é feita em circunstâncias tão favoráveis assim. Em situações reais, é comum que não se tenha acesso às equações de movimento. Além disso, mesmo se esse for o caso, é possível que as variáveis em si não estejam acessíveis. Um exemplo dessa situação é dada por Gordon [55], que apresenta uma versão do problema do carro preso no vale em que cada estado s_i é uma imagem mostrando duas posições sucessivas do veículo.

Uma vez definidos os estados do MDP, o próximo passo é determinar as probabilidades de transições entre eles. Como nesse caso o espaço de estados original é contínuo, a determinação das matrizes de transições requer uma discretização. Um espaço de estados contínuo pode ser discretizado de várias maneiras [33]. Sem perda de generalidade, suponha que os intervalos de x e $\dot{x}$ tenham sido divididos em ι subintervalos com o mesmo comprimento e que cada partição da malha resultante represente um estado s_i. Isso dá origem a um espaço de estados S finito com cardinalidade $|S| = \iota^2$. Teoricamente, a definição de S mais a dinâmica do problema determinariam as probabilidades de transição de maneira inequívoca. Quando se conhece as equações de movimento do sistema, em alguns casos é possível de fato definir analiticamente as matrizes $\mathbf{P}^a$. Uma outra maneira de determinar as matrizes de transições que não depende de um modelo analítico do problema é usar amostras de transições para estimar as probabilidades reais, como discutido na Seção 2.3. Nesse caso, as aproximações tornam-se arbitrariamente precisas à medida que o número de amostras tende ao infinito [111].

O último passo no processo de modelagem é definir as recompensas descrevendo os objetivos do problema. Quando a interação do agente com o ambiente pode ser naturalmente dividida em episódios, como é o caso do carro preso no vale, pode-se optar por usar recompensas descontadas ou não. Normalmente, no primeiro caso o agente recebe uma recompensa quando atinge um estado terminal, e no segundo uma recompensa é entregue em todos os passos da interação. Esta última estratégia tende a facilitar a definição de

técnicas de exploração, porque o agente coleta informação a respeito do ambiente a cada execução de uma ação. Por outro lado, um sistema de recompensas não descontado origina um MDP mais suscetível a instabilidades [55, 22], ao passo que um fator de desconto $\gamma < 1$ garante a existência e unicidade de uma solução para o problema, como discutido no capítulo anterior. Como aqui a estabilidade dos MDPs originados da fatoração estocástica é fundamental, todos os problemas deste capítulo foram formulados como problemas descontados em que a recompensa final é $r = \pm 1$. É importante notar que quando se modela um problema como um MDP descontado o fator de desconto γ deve estar de acordo com o número de passos necessários para resolvê-lo. Caso contrário, corre-se o risco de descaracterizar completamente o problema.

A Figura 4.2 mostra a função de custo do problema do carro preso no vale modelado como descrito nesta seção.

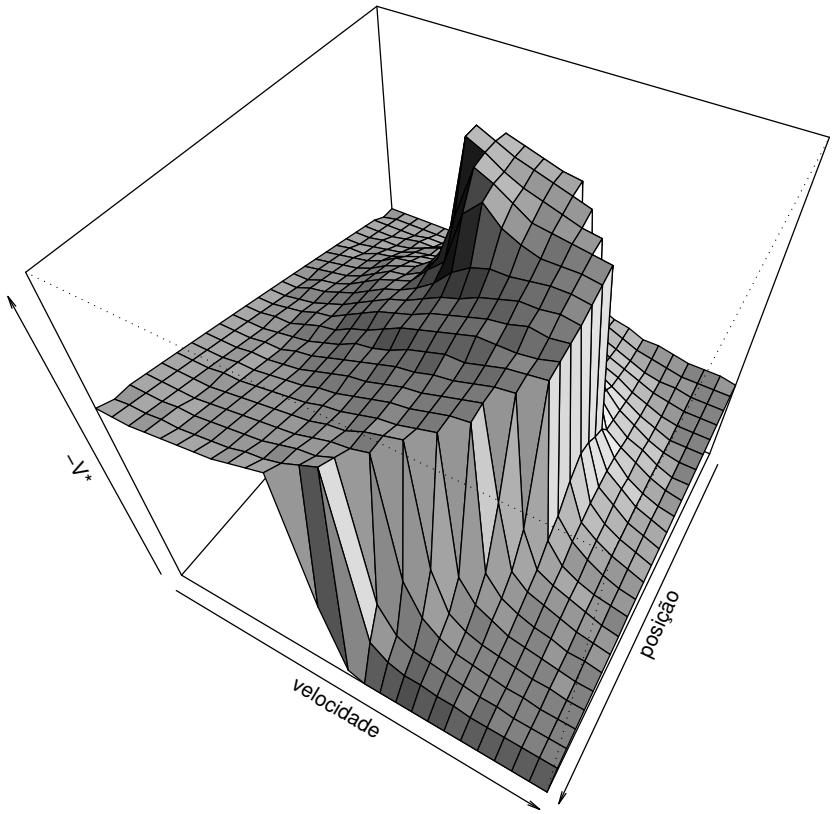


Figura 4.2: Função de custo ($-V^*$) do problema do carro preso no vale para $\gamma = 0.995$ e $\sigma = 0$. Calculada a partir de uma discretização usando $\iota = 250$ intervalos.

4.2 Fatoração Estocástica com Atribuição Proporcional

Encerrei o Capítulo 3 afirmando que o maior obstáculo para a aplicação da fatoração estocástica na redução de um processo de decisão de Markov é o seu custo computacional. Nesta seção eu apresento uma solução para esse problema para o caso em que S é um espaço métrico finito cuja topologia atende a algumas propriedades. Nesse contexto, pode-se realizar a fatoração em S, que não está sujeito ao crescimento superficial.

4.2.1 A relação topológica entre o espaço de estados e o espaço markoviano

A derivação de um processo de decisão de Markov finito pode ser vista como um mapeamento entre dois conjuntos:

$$\begin{aligned} \varphi^a : \quad & S \mapsto M^{|S|} \\ & \varphi^a(s_i) = \mathbf{m}_i^a \end{aligned} \qquad \text{para todo } a \in A, \tag{4.2}$$

onde $\mathbf{m}_i^a$ é a i-ésima linha da matriz $\mathbf{M}^a$ que descreve o processo de Markov associado com a ação a (veja (3.44)). Como discutido no Capítulo 3, os vetores-linha $\mathbf{m}_i^a$ pertencem ao simplex $\Delta^{|S|-1}$ acrescido de uma dimensão livre. Chamarei esse conjunto de *espaço markoviano* e denota-lo-ei por $M^{|S|}$.

Uma vez definidas as matrizes $\mathbf{P}^a$ e os vetores $\mathbf{r}^a$, a estrutura original de S torna-se irrelevante: o que determina a política de decisão final são os processos de Markov $\mathbf{M}^a$ definidos em (4.2). Note que é possível estabelecer uma correspondência entre duas formulações distintas de um mesmo problema: no exemplo do carro preso no vale, pode-se criar uma discretização dos intervalos de x e $\dot{x}$ que origine as mesmas matrizes de transições do MDP derivado das figuras de Gordon [55]. É nesse sentido que o espaço markoviano $M^{|S|}$ pode ser comparado com o espaço ideal de Platão.

Em geral, toda a informação relevante para a derivação de uma política de decisão está contida na descrição do MDP. Portanto, na solução de um problema de tomada de decisão seqüencial os estados $s_i \in S$ servem apenas para definir o mapeamento (4.2), ou seja, apenas como um mecanismo de identificação—por isso não seria absurdo comparar S a

um espaço sensorial. No entanto, em algumas situações pode ser vantajoso do ponto de vista computacional trabalhar no espaço S ao invés de manipular o MDP. Isso ocorre por exemplo quando a dimensão $\dim(S)$ desse espaço é muito menor do que a sua cardinalidade $|S|$. Para que seja possível trabalhar no espaço S diretamente, porém, é necessário estabelecer uma correspondência entre a sua topologia e a do espaço $M^{|S|}$ que contém as linhas das matrizes $\mathbf{M}^a$.

Sabe-se que $M^{|S|} \subset \mathbb{R}^{|S|+1}$. Como discutido, a fatoração estocástica em $M^{|S|}$ depende da definição de uma métrica que caracterize a qualidade de uma aproximação. Por exemplo, quando se usa a norma de Frobenius como função de custo, como em (3.34), assume-se implicitamente que a noção de distância adotada seja aquela induzida pela norma euclidiana. Formalmente, uma *distância* ou *métrica* é uma função real $\rho_M(\mathbf{m}_i^a,\mathbf{m}_j^a)$ definida para todo $\mathbf{m}_i^a, \mathbf{m}_j^a \in M^{|S|}$ tal que [80]:

i. $\rho_M(\mathbf{m}_i^a, \mathbf{m}_j^a) \geq 0$ (não-negatividade),

ii. $\rho_M(\mathbf{m}_i^a, \mathbf{m}_j^a) = 0 \iff \mathbf{m}_i^a = \mathbf{m}_j^a$ (identidade dos elementos indistinguíveis),

iii. $\rho_M(\mathbf{m}_i^a, \mathbf{m}_j^a) = \rho_M(\mathbf{m}_j^a, \mathbf{m}_i^a)$ (simetria),

iv. $\rho_M(\mathbf{m}_i^a, \mathbf{m}_j^a) \leq \rho_M(\mathbf{m}_i^a, \mathbf{m}_k^a) + \rho_M(\mathbf{m}_k^a, \mathbf{m}_j^a)$ (desigualdade triangular).

Note que o conjunto $M^{|S|}$ mais a distância ρ_M constituem um espaço métrico, o que justifica a nomenclatura adotada para $M^{|S|}$. Observe também que a propriedade de simetria não é essencial aqui. Quando a função a ser minimizada na fatoração estocástica é a divergência de Kullback-Leibler [81], por exemplo, essa propriedade não se cumpre.

Considere que S também seja um espaço métrico com distância ρ_S. A forma exata da função ρ_S não é importante, contanto que ela atenda as propriedades listadas acima. Se for possível estabelecer uma correspondência entre as distâncias definidas nos espaços $M^{|S|}$ e S, em alguns casos torna-se indiferente trabalhar no primeiro ou no segundo. Suponha por exemplo que o objetivo seja fatorar estocasticamente um processo de Markov $\mathbf{M}^\pi$ induzido por uma política π (*cf.* (3.44)). Pode-se a partir de (4.2) definir a função $\varphi^\pi : \varphi^\pi(s_i) = \varphi^{\pi(s_i)}(s_i)$, que fornece os vetores $\mathbf{m}_i^\pi$ associados com cada estado $s_i \in S$.

Suponha que a função φ^π preserve a distância definida no espaço de estados, ou seja:

$$\rho_S(s_i, s_j) = \rho_M(\varphi^\pi(s_i), \varphi^\pi(s_j)) = \rho_M(\mathbf{m}_i^\pi, \mathbf{m}_j^\pi) \text{ para todos } s_i, s_j \in S. \tag{4.3}$$

Nesse caso, qualquer operação que dependa exclusivamente da distância entre os elementos de um conjunto pode ser efetuada tanto no espaço S quando no espaço $M^{|S|}$. O resultado de uma operação desse tipo no espaço S seria um conjunto de elementos que se relacionam através do mapeamento φ^π com a solução que seria encontrada em $M^{|S|}$.

Para tornar esta discussão mais concreta, imagine que a fatoração estocástica de $\mathbf{M}^\pi$ esteja sendo realizada pelo algoritmo *k-medoids* [72], que é a variante do k-means que restringe os centros dos agrupamentos aos pontos dados (ou seja, nesse caso cada linha da matriz $\mathbf{D}$ teria apenas um elemento não-nulo e as linhas de $\mathbf{W}$ seriam um subconjunto das linhas de $\mathbf{M}^\pi$). No algoritmo k-medoids a clusterização é baseada em uma matriz de distâncias calculada no início do processo. Não é difícil perceber que, caso a relação (4.3) se cumpra, o resultado de uma clusterização em S será idêntico ao agrupamento realizado diretamente no espaço $M^{|S|}$—ou seja, os estados s_i selecionados como centros dos *clusters* correspondem aos vetores $\mathbf{m}_i^\pi$ que seriam selecionados na clusterização de $\mathbf{M}^\pi$. No entanto, dependendo da dimensão de S, executar o agrupamento nesse espaço pode significar uma grande economia de tempo de computação. Tome como exemplo o problema do carro preso no vale. Quando se trabalha diretamente no espaço $M^{|S|}$, o cálculo da matriz de distâncias usada pelo algoritmo k-medoids envolve um número de operações na ordem de ι^6, onde ι é o número de intervalos usado para discretizar cada dimensão do problema. No entanto, como o cálculo da distância ρ_S independe da cardinalidade de S, a determinação da matriz de distâncias nesse espaço seria $O(\iota^4)$ apenas.

A relação (4.3), embora útil, é muito difícil de garantir, principalmente quando se considera que no contexto da programação dinâmica ela teria que ser atendida para todas as políticas de decisão que surgissem no processo de iteração de política generalizada.[1] Uma condição mais fraca é exigir que as distâncias ρ_S e ρ_M atendam à seguinte relação,

[1] A relação (4.3) é automaticamente atendida para toda política π se

$$\rho_S(s_i, s_j) = \rho_M(\mathbf{m}_i^{a_k}, \mathbf{m}_j^{a_l}) \text{ para todos } s_i, s_j \in S \text{ e todos } a_k, a_l \in A.$$

dada uma política π:

$$\rho_S(s_i, s_j) \leq \rho_S(s_i, s_k) \Longrightarrow \rho_M(\mathbf{m}_i^\pi, \mathbf{m}_j^\pi) \leq \rho_M(\mathbf{m}_i^\pi, \mathbf{m}_k^\pi) \text{ para todos } s_i, s_j, s_k \in S. \tag{4.4}$$

Quando a condição acima é atendida, o mapeamento φ^π preserva as *distâncias relativas* entre os elementos de S. No caso da fatoração estocástica, ao se substituir (4.3) por (4.4) perde-se a garantia de que os arquétipos selecionados em S serão os mesmos que seriam selecionados em $M^{|S|}$. No entanto, dependendo da estratégia adotada para realizar a *atribuição* de estados a arquétipos, o resultado dessa operação será o mesmo em ambos os espaços. Por exemplo, se cada estado s_i for associado ao arquétipo q_j mais próximo, como ocorre nos algoritmos k-means e k-medoids, a matriz $\mathbf{D}$ determinada a partir dos elementos $s_i \in S$ será idêntica à que seria calculada com base nos vetores $\mathbf{m}_i^\pi \in M^{|S|}$ (considerando, é claro, o mesmo conjunto de arquétipos). Mesmo no caso em que $\mathbf{D}$ contém mais de uma entrada não-nula por linha, é fácil perceber que a ordem de magnitude dos elementos d_{ij} em cada linha $\mathbf{d}_i$ seria preservada ao se deslocar o processo de atribuição de $M^{|S|}$ para S.

As expressões (4.3) e (4.4) são dois exemplos de correspondência topológica entre os espaços $M^{|S|}$ e S. Elas ilustram bem uma tendência geral: quanto mais restritivas as condições impostas sobre o mapeamento φ^π, mais forte a relação entre as topologias de $M^{|S|}$ e S, e portanto maiores as garantias em relação a uma fatoração realizada no espaço de estados. Assim sendo, é aconselhável durante o processo de modelagem levar em consideração o efeito que as variáveis selecionadas para descrever os estados s_i terão sobre as funções φ^a.

Inconsistências topológicas

Em geral, é desejável definir o espaço S de forma a existir uma correspondência topológica entre esse espaço e o espaço markoviano derivado através de (4.2). No entanto, mesmo quando isso é possível, podem ocorrer inconsistências pontuais entre as topologias de S e $M^{|S|}$. Em outras palavras, mesmo no caso em que condições como as mostradas nas expressões (4.3) e (4.4) são atendidas na maioria das vezes, pode acontecer de elas serem violadas esporadicamente. Observe por exemplo a função de valor do problema

do carro preso no vale, mostrada na Figura 4.2. A partir da figura e da descrição da tarefa, é correto dizer que a condição (4.4) é normalmente atendida nessa formulação do problema. No entanto, a descontinuidade da função V^*—mostrada como uma "escarpa" na Figura 4.2—indica claramente que essa condição é violada em alguns casos. Como mostra a Figura 4.3, pode-se identificar na função de valor uma "fronteira" separando os estados a partir dos quais é possível aproximar-se do objetivo final do problema daqueles em que isso é impossível—ou seja, daqueles em que a velocidade do carro é insuficiente para levá-lo diretamente ao topo da montanha. Em um certo sentido essa fronteira representa uma ruptura da topologia de $M^{|S|}$ em relação à topologia de S: é razoável considerar que estados s_i e s_j que sejam vizinhos neste último e que estejam em lados opostos da fronteira não sejam vizinhos em $M^{|S|}$, dada a diferença entre os vetores $\mathbf{m}_i^\pi$ e $\mathbf{m}_j^\pi$ correspondentes.

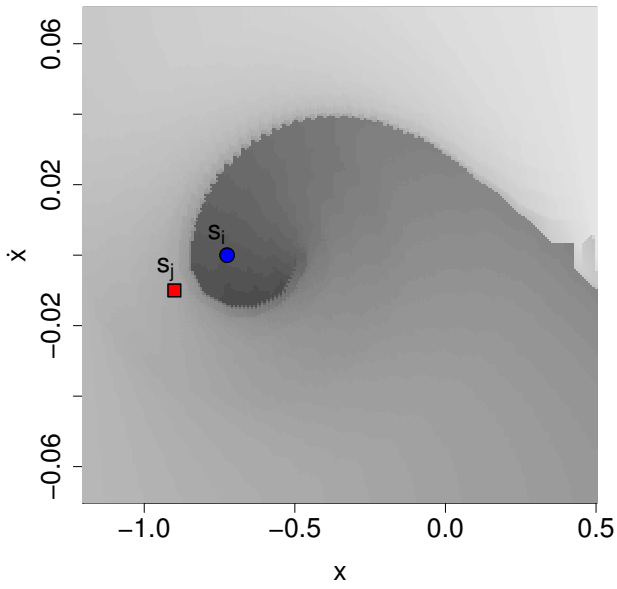


Figura 4.3: Função de valor do problema do carro preso no vale representada como um mapa em escalas de cinza. Quanto mais clara uma região, maior o valor de V^*. Valores calculados com $\gamma = 0.995$, $\sigma = 0$ e $\iota = 250$. O quadrado e o círculo representam dois estados s_i e s_j que, embora vizinhos em S, são muito diferentes em $M^{|S|}$.

Supor que a relação (4.4) se cumpre quando esse não é o caso pode ter conseqüências desastrosas. Imagine por exemplo que o estado s_i mostrado como um círculo na Figura 4.3 tenha sido selecionado como um dos arquétipos da fatoração estocástica. Nesse

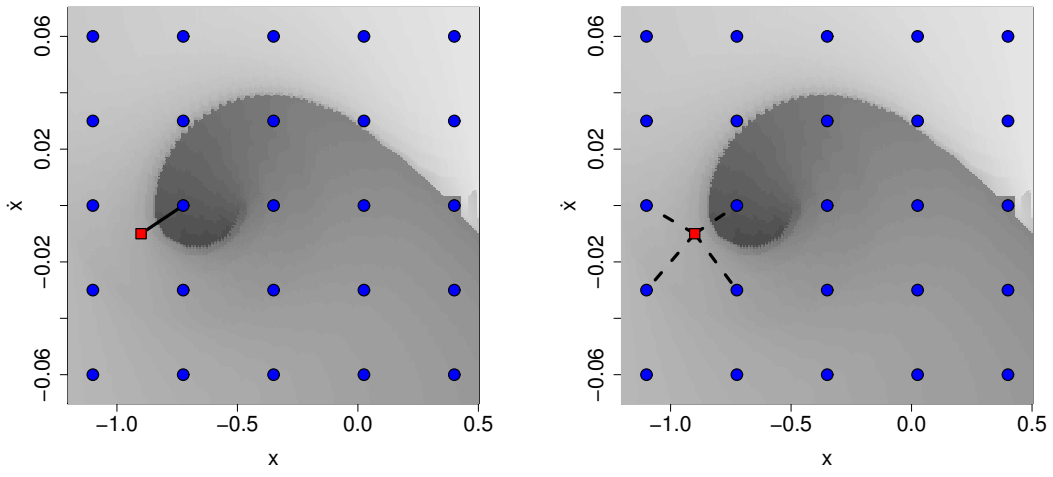
caso, é possível que o estado s_j, representado na figura por um quadrado, seja atribuído a esse arquétipo. Isso significa que na fatoração estocástica de $\mathbf{M}^{\pi^*}$ o vetor-linha $\mathbf{m}_j^{\pi^*}$ seria representado pelo vetor $\mathbf{m}_i^{\pi^*}$. Ou seja, todas as transições de s_j seriam redirecionadas para s_i, que tem características completamente diferentes: enquanto a partir de s_j é possível atingir o topo da montanha apenas acelerando o carro, no estado s_i é necessário dar marcha-à-ré para ganhar velocidade na encosta oposta do vale. Não é prudente criar qualquer expectativa em relação a uma política de decisão derivada de um modelo em que esses estados são indistinguíveis.

4.2.2 Atribuição proporcional de estados a arquétipos

A existência de uma correspondência topológica entre $M^{|S|}$ e S permite, pelo menos em princípio, que a fatoração estocástica seja transposta do primeiro espaço para o segundo, o que pode resultar em uma grande economia de tempo de processamento. Na prática, porém, é muito difícil garantir que condições como (4.3) e (4.4) sejam atendidas, e mesmo no caso em que isso é possível violações esporádicas podem ocorrer. Uma possibilidade para lidar com essa situação é tentar durante o processo de modelagem definir o espaço S de maneira a induzir, ainda que artificialmente, uma correspondência entre as topologias do espaço de estados e do espaço markoviano. Uma outra idéia é desenvolver uma estratégia robusta para realizar a fatoração estocástica em S, que funcione mesmo no caso em que a correspondência topológica entre S e $M^{|S|}$ seja apenas parcial.

Uma maneira de contornar eventuais inconsistências topológicas é adotar uma *atribuição proporcional* de estados a arquétipos. Esse tipo de atribuição implica em uma matriz de desvio $\mathbf{D}$ com várias entradas não-nulas por linha. A magnitude de cada elemento d_{ij} deve ser inversamente proporcional à distância $\rho_S(s_i, s_j)$, que em geral é uma boa estimativa da dissimilaridade entre $\mathbf{m}_i^a$ e $\mathbf{m}_j^a$, com $a \in A$. Quando se interpreta os elementos de $\mathbf{D}$ como probabilidades de realocação das transições originais, a atribuição proporcional corresponde a desviar as transições para vários arquétipos ao invés de concentrá-las em apenas um. A expectativa é que o efeito de eventuais inconsistências entre as topologias de S e $M^{|S|}$ sejam compensados pelos casos em que as condições impostas sobre φ^a se cumprem normalmente.

A Figura 4.4 ilustra essa situação. Na Figura 4.4a o estado em destaque é atribuído a um único arquétipo, como ocorre por exemplo na versão original do algoritmo k-medoids. Nesse contexto cada arquétipo pode ser identificado com um conjunto convencional ou *crisp*. Como nesse caso o elemento $\mathbf{m}_i^{\pi}$ fica representado por apenas um estado arquetípico, uma atribuição equivocada como aquela mostrada na figura pode ter efeitos catastróficos, pelas razões discutidas acima. Por outro lado, se a associação entre estados e arquétipos for feita de maneira proporcional, respeitando o grau de similaridade entre eles, é razoável esperar que os efeitos indesejáveis de uma atribuição errônea sejam amenizados pelas associações corretas. Essa situação é mostrada na Figura 4.4b. Note como nessa figura o estado em destaque é associado a quatro arquétipos diferentes, como ocorre na teoria dos conjuntos *fuzzy* [191]. Dos quatro arquétipos, apenas um encontra-se do lado oposto da fronteira, enquanto os demais apresentam um padrão de transição bem próximo daquele do estado em questão. Isso implica em uma representação sensivelmente superior àquela mostrada na Figura 4.4a.



(a) Atribuição simples ou *crisp*: cada estado s_i é associado a um único arquétipo

(b) Atribuição proporcional ou *fuzzy*: o estado s_i está associado com vários arquétipos através de diferentes graus de pertinência

Figura 4.4: Duas maneiras possíveis de se fazer a associação entre estados e arquétipos. Os círculos representam os arquétipos selecionados por um algoritmo qualquer e o quadrado o estado cuja atribuição está sendo analisada. A linha contínua representa uma atribuição simples; as linhas pontilhadas são atribuições proporcionais.

Uma analogia que ajuda a entender a diferença entre uma matriz de desvio *crisp* e

uma *fuzzy* é a aprendizagem baseada em instâncias, cujo representante mais conhecido é o algoritmo dos k *vizinhos mais próximos* (k-NN[†]) [102, 60]. Como se sabe, o algoritmo k-NN pode ser usado para aproximar uma função $f : \mathbb{R}^n \mapsto \mathbb{R}^m$. Nesse caso, o valor de f em um ponto x_i é a soma ponderada do valor dessa função nos k vizinhos mais próximos de x_i, selecionados de um conjunto de pontos previamente armazenado pelo algoritmo. Em geral, os pesos usados na aproximação são inversamente proporcionais à distância entre os pontos. No caso da fatoração estocástica, a função f a ser aproximada seria o mapeamento φ^π, e as instâncias armazenadas seriam os arquétipos. Uma matriz $\mathbf{D}$ com apenas um elemento não-nulo por linha corresponderia ao caso em que $k = 1$, ou seja, em que apenas o vizinho mais próximo é usado na aproximação. Essa configuração representa a situação em que o aproximador tem o maior número de graus de liberdade, o que em geral aumenta a variância da função de aproximação [60]. À medida que $k \to m$, a aproximação realizada pelo k-NN se torna mais suave, e portanto mais robusta a variações da função φ^π. O parâmetro k pode ser visto como uma maneira de lidar com o famoso dilema entre o *bias* e a variância de uma aproximação [61, 60], e o seu valor ótimo depende da função a ser aproximada. Pode-se esperar, portanto, que o mesmo fenômeno ocorra na determinação dos elementos da matriz $\mathbf{D}$. Voltarei a este assunto à frente.

Note que embora a técnica de atribuição proporcional seja mais robusta do que a atribuição simples, ela ainda pressupõe uma correspondência entre as topologias de S e $M^{|S|}$. Estabelecer condições para que a aplicação dessa estratégia seja bem-sucedida não é trivial, e além disso esse tipo de tecnicalidade é raramente verificável na prática. Freqüentemente, porém, é possível estabelecer a partir da descrição do problema se as noções de similaridade em S e $M^{|S|}$ estão relacionadas. Basta verificar se em geral quanto menor a distância $\rho_S(s_i, s_j)$ menor a distância correspondente $\rho_M(\mathbf{m}_i^a, \mathbf{m}_j^a)$, para qualquer ação $a \in A$. Ou seja, basta checar se a proximidade entre dois estados implica em probabilidades de transições e recompensas semelhantes. Se esse for o caso, espera-se que o efeito de eventuais violações dessa tendência seja amenizado à medida que se aumenta o número de arquétipos usados na fatoração estocástica. O número m de arquétipos necessários para uma boa fatoração em S depende do domínio da aplicação. O ideal é que a densidade da fatoração estocástica seja tal que todo estado s_i seja representado majoritariamente por

[†] *k-nearest neighbors*

arquétipos com padrões de transições similares ao seu (aqui, pode-se recorrer mais uma vez à analogia com o algoritmo k-NN para se ter uma ilustração desta afirmação).

Kernels

Uma questão pertinente se refere à implementação da atribuição proporcional. Como discutido, uma matriz $\mathbf{D}$ válida deve respeitar os seguintes critérios:

1. Como se trata de uma matriz estocástica, os elementos em cada uma de suas linhas devem ser não-negativos e ter soma 1, ou seja,

$$\begin{aligned} &d_{ij} \geq 0 && \text{para } i = 1, 2, ..., |S| \text{ e } j = 1, 2, ..., m \\ &\textstyle\sum_{j=1}^{m} d_{ij} = 1 && \text{para } i = 1, 2, ..., |S|. \end{aligned} \tag{4.5}$$

2. O elemento d_{ij} representa a pertinência do estado s_i ao arquétipo q_j, e portanto a sua magnitude deve ser inversamente proporcional à distância $\rho_S(s_i, q_j)$. Ou, de maneira mais precisa:

$$\rho_S(s_i, q_j) < \rho_S(s_i, q_k) \Longrightarrow d_{ij} \geq d_{ik}. \tag{4.6}$$

Uma das maneiras de definir uma matriz de desvio que atenda aos requisitos acima é usar os coeficientes da regressão local de Nadaraya-Watson [60]. Nesse caso, os elementos da matriz $\mathbf{D}$ seriam dados por:

$$d_{ij} = \kappa(s_i, q_j) = \frac{\phi\left(\frac{\rho_S(s_i, qj)}{\tau}\right)}{\sum_{k=1}^{m} \phi\left(\frac{\rho_S(s_i, q_k)}{\tau}\right)}. \tag{4.7}$$

A função de pertinência κ é normalmente chamada de *kernel* [111, 60]. Note que por definição a pertinência de um estado s_i a dois arquétipos eqüidistantes é sempre a mesma, embora dois estados que estejam a igual distância de um arquétipo q_j possam ter diferentes graus de pertinência a esse arquétipo. A função ϕ é um mapeamento $\mathbb{R}^+ \mapsto \mathbb{R}^+$ limitado superiormente e que atende à seguinte condição:

$$x_1 < x_2 \Longrightarrow \phi(x_1) \geq \phi(x_2), \text{ com } x_1, x_2 \in \mathbb{R}^+. \tag{4.8}$$

Como ϕ define a "forma" do *kernel* κ, irei me referir a essa função como *função-núcleo*. A partir da definição da função-núcleo fica claro que os elementos d_{ij} determinados pela

expressão (4.7) atendem às condições (4.5) e (4.6) listadas no início desta seção. O parâmetro τ em (4.7) controla a velocidade de decaimento de $\phi(x)$ em função de x: quanto maior esse parâmetro, mais lento é o decréscimo do valor da função. Eis alguns exemplos de funções candidatas ao papel de função-núcleo:

Função uniforme

$$\phi(x) = \begin{cases} \frac{1}{2}, \text{ se } x \leq 1 \\ 0, \text{ caso contrário} \end{cases}$$

Função linear

$$\phi(x) = \max(1 - x, 0)$$

Função gaussiana

$$\phi(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right)$$

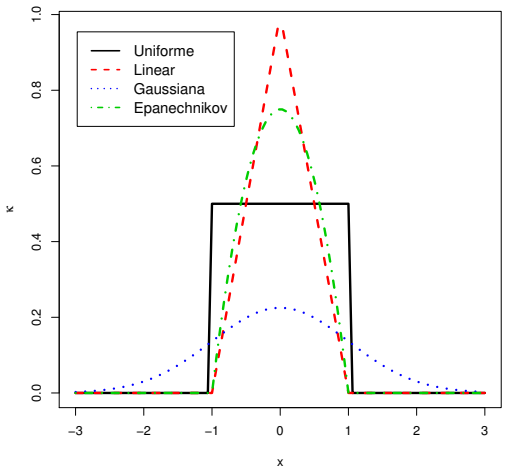
Função de Epanechnikov

$$\phi(x) = \max\left(\frac{3}{4}\left[1 - x^2\right], 0\right).$$

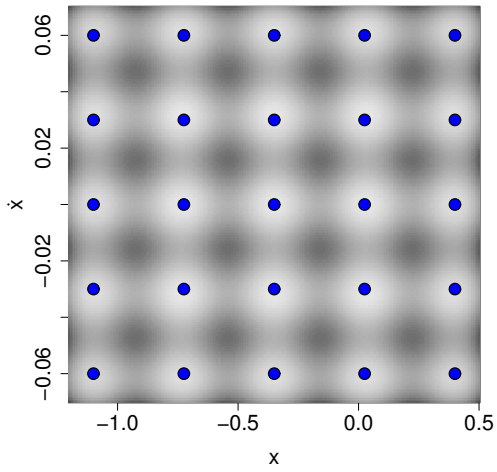
É interessante observar que quando o argumento da função ϕ é uma distância calculada em S, como em (4.7), ela pode ser interpretada como uma *função de base radial* (RBF[†]) definida nesse espaço [125]. Sob esta perspectiva, os arquétipos q_j seriam os *centros* das RBFs e o parâmetro τ corresponderia à sua *largura*. A Figura 4.5a mostra as RBFs correspondentes às funções acima para o caso hipotético em que $S = \mathbb{R}$. Vale ressaltar que qualquer função-núcleo ϕ que atenda à condição (4.8) e seja constante em apenas um intervalo fechado de $\mathbb{R}$ origina naturalmente uma função de base radial local não-negativa em S. A Figura 4.5b ilustra o uso da função gaussiana como função de pertinência dos arquétipos mostrados na Figura 4.4. Observe como os centros das funções radiais coincidem com os arquétipos mostrados na figura anterior.

É importante ressaltar que a expressão (4.7) não é de forma alguma a única maneira de se implementar a atribuição proporcional de estados a arquétipos. Qualquer estratégia que resulte em elementos d_{ij} que atendam às restrições (4.5) e (4.6) é a princípio uma

[†] *Radial basis function*



(a) Exemplos de RBFs em $\mathbb{R}$. Todas as funções têm centro na origem e largura $\tau = 1$. A distância adotada é aquela derivada da norma euclidiana.



(b) Uso da função gaussiana como função de pertinência dos arquétipos mostrados na Figura 4.4. A fim de compensar a diferença de escala das variáveis foi adotada uma norma ponderada como métrica.

Figura 4.5: O uso de funções de base radiais para implementar a atribuição proporcional.

alternativa válida. Por exemplo, uma opção seria tomar emprestado de Bezdek a sua estratégia de atribuição do algoritmo *fuzzy* k-means [24],

$$d_{ij} = \frac{1}{\sum_{k=1}^{m} \left(\frac{\rho_S(s_i, s_j)}{\rho_S(s_i, s_k)} \right)^{\frac{2}{\tau - 1}}}, \tag{4.9}$$

que também resultaria em uma matriz $\mathbf{D}$ válida. Na expressão acima o parâmetro τ tem uma interpretação análoga à que tem no caso das funções de base radiais.

Um ponto fundamental que precisa ser enfatizado é que, tanto em (4.7) quanto em (4.9), *o único requerimento para a aplicação do kernel* $\kappa : S \times S \mapsto \mathbb{R}$ *é que o espaço* S *seja um espaço métrico*. Isso garante a generalidade da abordagem proposta aqui. Especificamente, isso significa que a fatoração estocástica pode a princípio ser realizada em qualquer conjunto de elementos S em que seja possível definir uma noção de dissimilaridade ρ_S. Esta definição é genérica o suficiente para incluir espaços de estados pouco convencionais, como por exemplo espaços heterogêneos em que algumas das variáveis pertencem a um alfabeto finito. A possibilidade de se trabalhar em espaços estruturados genéricos é freqüentemente citada como um dos atrativos da *aprendizagem com kernels*,

uma das áreas de pesquisa em aprendizagem de máquina mais ativas na atualidade [144]. Não custa lembrar que neste contexto só faz sentido aplicar a fatoração estocástica em S se a métrica deste espaço estiver relacionada com a noção de distância adotada em $M^{|S|}$.

4.2.3 Algoritmo

Uma vez definidos os arquétipos da fatoração estocástica, pode-se usar uma das estratégias descritas na última seção para determinar a matriz $\mathbf{D}$. Falta ainda descrever como a matriz $\mathbf{W}$ seria construída nesse caso. Os vetores-linha de $\mathbf{W}$ podem ser facilmente obtidos a partir dos arquétipos selecionados em S e do mapeamento (4.2). Note portanto que a técnica escolhida para a fatoração estocástica deve necessariamente restringir a seleção dos arquétipos ao conjunto S, de maneira que seja possível estabelecer uma correspondência posterior com o espaço $M^{|S|}$. Isso descarta por exemplo o uso da versão original do algoritmo *fuzzy* k-means [44, 24], que de outra forma seria perfeito para esse propósito.

Adicionalmente, é preciso contornar uma disparidade estrutural entre os espaços S e $M^{|S|}$. Como cada elemento $s_i \in S$ possui $|A|$ elementos correspondentes em $M^{|S|}$, é necessário estabelecer uma estratégia para selecionar as linhas de $\mathbf{W}$ no espaço markoviano. Uma possibilidade seria adicionar à matriz $\mathbf{W}$ o vetor $\mathbf{m}_i^a$ que melhor representasse os outros $|A| - 1$ elementos associados com um arquétipo q_i. Note, entretanto, que em geral os elementos $\mathbf{m}_i^a$ referentes a ações a diferentes são muito dissimilares entre si. Basta pensar no exemplo do carro preso no vale, em que as ações de marcha-à-ré e acelerar têm efeitos opostos. Portanto, uma estratégia simples e que soluciona o problema consiste em simplesmente adicionar à $\mathbf{W}$ *todos* os $|A|$ elementos $\mathbf{m}_i^a$ associados com o arquétipo q_i. Note, no entanto, que essa abordagem multiplica o número de arquétipos por $|A|$.

O Algoritmo 4.1 mostra uma maneira de implementar a fatoração estocástica em S com a atribuição proporcional representada por um *kernel*, como na expressão (4.7). Com essa técnica é possível calcular as matrizes $\mathbf{D}$ e $\mathbf{W}$ a um custo computacional razoável. O número exato de operações necessárias para fazê-lo depende da estratégia adotada para selecionar os arquétipos no espaço de estados. O cálculo da matriz com as distâncias entre todos os elementos de S é $O(|S|^2)$. Para o caso em que se deseja agrupar esses

elementos, existem variantes do algoritmo k-medoids cujo custo por iteração é aproximadamente linear em $|S|$ [109]. Note no entanto que se o espaço de estados S for o resultado de uma discretização regular, como no exemplo do carro preso no vale, os arquétipos podem ser determinados de maneira muito eficiente: basta selecionar m pontos igualmente espaçados na malha resultante da discretização. Nesse caso, o *custo total* da fatoração estocástica seria $O(|S|m)$, que é o número de operações necessárias para se calcular as distâncias dos estados aos arquétipos usadas na construção da matriz $\mathbf{D}$.

Algoritmo 4.1 Fatoração estocástica em S com atribuição baseada em *kernels*

Requer S, $\mathbf{M} \in \mathbb{R}^{|S||A|\times|S|+1}$, $m \in [1, |S|]$, $\tau \in \mathbb{R}_*^+$
Retorna $\mathbf{D} \in \mathbb{R}^{|S||A|\times m}$ e $\mathbf{W} \in \mathbb{R}^{m\times|S|+1}$ tais que $\mathbf{DW} \approx \mathbf{M}$
  $m_2 \leftarrow \lfloor m/|A| \rfloor$, onde $\lfloor x \rfloor$ retorna a parte inteira de x
  $m \leftarrow m_2 \times |A|$
  Selecione m_2 arquétipos $q_i \in S$ usando um algoritmo qualquer
  Inicialize a matriz $\mathbf{D}$ com zeros apenas
  para $a \leftarrow 1$ **até** $|A|$ **faça**
    para $j \leftarrow 1$ **até** m_2 **faça**
      $c \leftarrow (a-1) \times m_2 + j$
      para $i \leftarrow 1$ **até** $|S|$ **faça**
        $l \leftarrow (a-1) \times |S| + i$
        $d_{lc} \leftarrow \kappa(s_i, q_j)$ ▷ Determinação dos elementos da matriz $\mathbf{D}$
      fim para
      $k \leftarrow$ linha de $\mathbf{M}$ correspondente ao arquétipo q_j e à ação a
      $\mathbf{w}_c \leftarrow \mathbf{m}_k$ ▷ Determinação dos vetores-linha da matriz $\mathbf{W}$
    fim para
  fim para

Dependendo do valor de m, um custo na ordem de $|S|m$ pode tornar a fatoração estocástica uma alternativa atraente para o cálculo da função de valor de um processo de Markov induzido por uma política π. No entanto, é combinada com o algoritmo PISF que a fatoração estocástica em S oferece os maiores ganhos em termos de tempo de computação, como será visto nos experimentos à frente. Note que da forma como está apresentado o Algoritmo 4.1 pode ser prontamente utilizado para fatorar estocasticamente um processo de decisão de Markov. É importante ressaltar que os resultados teóricos do Capítulo 3 ainda são válidos nesse caso, assim como no caso em que o Algoritmo 4.1 é usado para fatorar um processo de Markov $\mathbf{M}^\pi$.

4.2.4 Análise empírica

Nesta seção o problema do carro preso no vale é usado para avaliar a efetividade da fatoração estocástica realizada tanto no espaço $M^{|S|}$ quanto no espaço S. Para tal, este último foi discretizado como descrito na Seção 4.1.2, com $\iota = 30$. As matrizes de transições e os vetores-recompensa foram calculados a partir de 100 transições iniciadas em cada um dos 900 estados com cada uma das 3 ações do problema. Ambos os espaços foram munidos com a medida de distância induzida pela norma euclidiana (no caso do espaço S, a dissimilaridade entre dois estados s_i e s_j foi definida como a distância entre os pontos médios das respectivas partições). O algoritmo PISF foi usado para determinar a política de decisão nos dois casos. O método adotado para realizar a fatoração estocástica em $M^{|S|}$ foi o algoritmo k-means. No caso do espaço S, os arquétipos foram selecionados como pontos igualmente espaçados na malha bidimensional e a atribuição proporcional foi calculada pelo Algoritmo 4.1 usando a função gaussiana com $\tau = 0.1$. O valor de τ foi selecionado do conjunto $\{10, 1, 0.1, 0.01\}$ a partir de uma bateria preliminar de experimentos. Com exceção de $\tau = 0.01$, todas as larguras do *kernel* gaussiano resultaram em um desempenho razoável do algoritmo.

Em todos os experimentos deste capítulo os diferentes algoritmos são avaliados segundo o desempenho das políticas de decisão encontradas por eles. No caso do carro preso no vale, as políticas foram avaliadas através de um conjunto de teste composto por uma série de estados a partir dos quais o agente deveria escapar. Os estados foram selecionados como 25 pontos distribuídos de maneira regular no conjunto $[-1.0, 0.15] \times [-0.07, 0.02]$. Essa seleção foi feita de forma a evitar estados triviais a partir dos quais o carro atingiria o topo da montanha independentemente das ações selecionadas. Especificamente, na versão determinística do problema o número mínimo de passos necessários para atingir o objetivo a partir de qualquer estado selecionado do produto cartesiano acima é 33. O número máximo correspondente é 109.[2] Por isso, nos experimentos desta seção se o agente não atingisse o objetivo em 200 passos o episódio era considerado um fracasso e interrompido. Como foi adotada uma versão não-determinística do problema, com $\sigma = 1$, em alguns casos os agentes foram capazes de atingir o topo da montanha em um número

[2] Valores calculados a partir de uma discretização regular do problema com $\iota = 250$.

de passos inferior a 33. Para amenizar o efeito da estocasticidade na aplicação das ações, a avaliação das políticas foi repetida 20 vezes a partir de cada um dos 25 estados do conjunto de teste.

A Figura 4.6 mostra o conjunto de teste sobreposto à função de valor do problema do carro preso no vale. Também nesta figura é possível ver os resultados médios obtidos a partir de cada estado desse conjunto pela política de decisão derivada da discretização do espaço de estados original do problema. Observe como os resultados estão relacionados com o valor de V^*, como era de se esperar. Em particular, note como o número de passos necessários para alcançar o topo da montanha cresce quando se cruza a fronteira no sentido de crescimento de x. Além disso, é interessante chamar a atenção para o fato de o número máximo de 109 passos ter sido superado em quatro estados do conjunto de teste. Isso sugere que uma maior resolução na discretização do espaço S poderia trazer algum benefício. Por outro lado, das 20 tentativas de alcançar o objetivo feitas pelo agente a partir dos 25 pontos do conjunto de teste, apenas três terminaram em falhas, o que indica uma boa qualidade da política de decisão derivada da discretização. Coincidentemente, em 3 ocasiões o número de passos esteve abaixo do mínimo de 33 necessários para escapar do vale na versão determinística do problema. O número médio de passos executados pelo agente considerando todo o conjunto de teste foi 74.01, com um desvio-padrão de 33.00 passos.

Note que os vetores $\mathbf{m}_i^a$ que surgem da discretização de um problema determinístico são justamente aqueles que causam os maiores problemas para a fatoração estocástica. Como discutido extensivamente no capítulo anterior, isso ocorre porque os vetores de transições $\mathbf{p}_i^a$ estão localizados nos vértices do simplex $\Delta^{|S|-1}$. No caso do carro preso no vale, o acréscimo de um ruído às ações executadas pelo agente tende a aliviar o problema, mas infelizmente não chega a resolvê-lo. Como mostra a Figura 4.7, os resultados do algoritmo PISF combinado com a fatoração estocástica em $M^{|S|}$ deixam a desejar. Com $m = 400$ arquétipos usados na aproximação, a política derivada falha reiteradamente em praticamente todos os pontos do conjunto de teste. Embora o acréscimo no número de arquétipos melhore os resultados do algoritmo, um desempenho razoável só é alcançado quando $m = 700$, o que resulta em uma economia de tempo de computação modesta em relação à versão original do problema, em que $|S| = 900$. Pode-se dizer, portanto, que

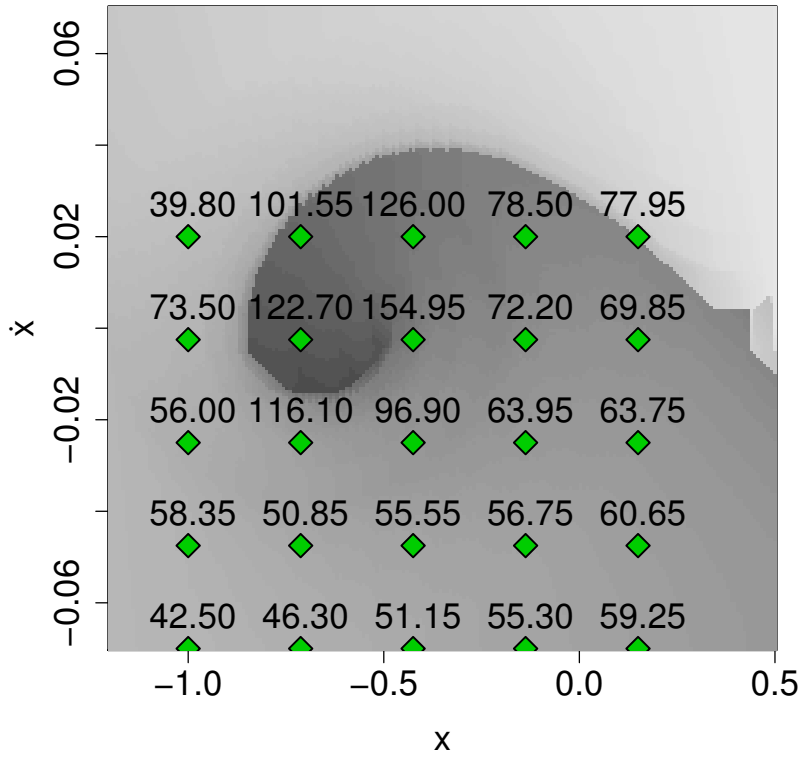


Figura 4.6: Conjunto de teste com os resultados médios obtidos pelo algoritmo de iteração de política no problema do carro preso no vale. O MDP foi construído a partir da discretização do espaço de estados com $\iota = 30$ e $\sigma = 1$.

nesse problema o uso da fatoração estocástica não é aconselhável, pelo menos quando ela é realizada no espaço $M^{|S|}$ pelo algoritmo k-means.

Seria possível que uma fatoração estocástica em S resultasse em políticas de decisão de melhor qualidade? Surpreendentemente, a resposta é sim. Observe na Figura 4.8a como o algoritmo PISF usando essa configuração é capaz de obter resultados bem melhores do que os anteriores, mesmo utilizando um número muito menor de arquétipos na aproximação. O fato de a figura estar um pouco confusa é bom sinal: ele indica que os resultados do algoritmo PISF aproximam-se daqueles encontrados pelo algoritmo de iteração de política na sua versão original. A Figura 4.8b compara o desempenho das políticas encontradas por esses algoritmos no bloco formado por todo o conjunto de teste. Note que quando $m \geq 120$ os resultados do algoritmo PISF-S são praticamente idênticos aos do algoritmo PI. As diferenças nesse caso, tanto em um sentido quanto em outro, são

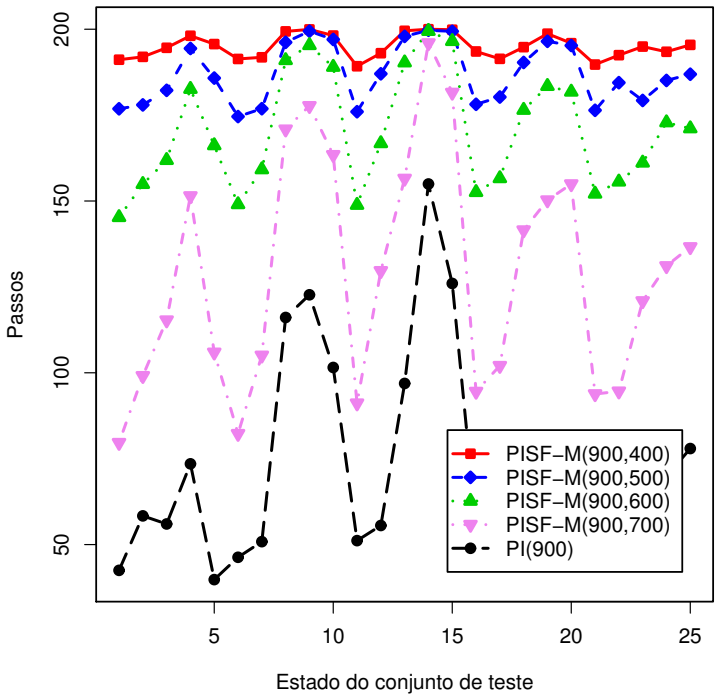
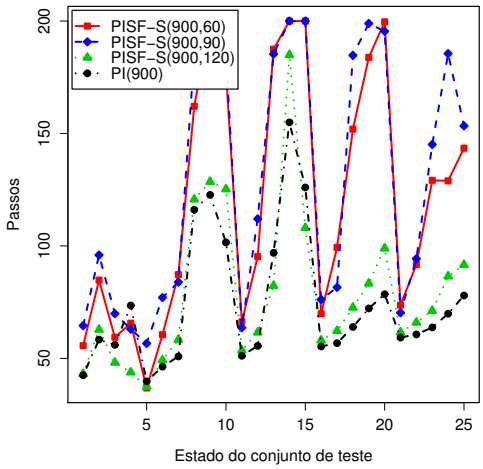
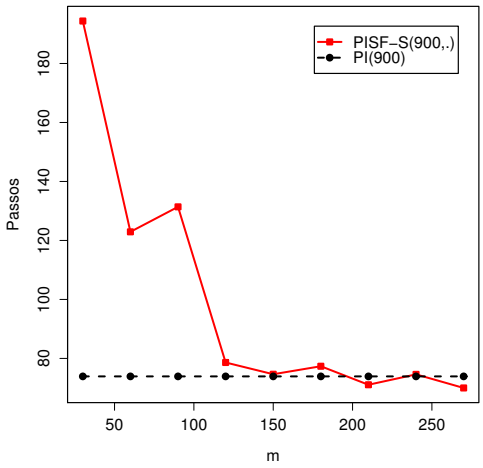


Figura 4.7: Resultados obtidos pelo algoritmo PISF no problema do carro preso no vale com a fatoração estocástica realizada em $M^{|S|}$ pelo algoritmo k-means. Cada ponto corresponde a uma média de 20 execuções desse algoritmo. Em cada execução a política encontrada foi testada 20 vezes em cada estado do conjunto de teste. O eixo das abscissas corresponde aos 25 pontos desse conjunto. Os valores entre parênteses após os nomes dos algoritmos são a dimensão do MDP original e o número m de arquétipos usados na fatoração, respectivamente. A sigla "PI" se refere ao algoritmo de iteração de política sem a fatoração estocástica, cujos resultados são mostrados aqui como referência.

provavelmente uma conseqüência do caráter estocástico do problema (já que $\sigma = 1$).



(a) Resultados discriminados por estado do conjunto de teste



(b) Resultados médios obtidos em todos os estados do conjunto de teste. Os resultados do algoritmo PI independem do valor de m, e sua média é mostrada aqui apenas como referência.

Figura 4.8: Resultados obtidos pelo algoritmo PISF no problema do carro preso no vale com a fatoração estocástica realizada em S. Os arquétipos foram selecionados uniformemente na malha bidimensional e a atribuição proporcional foi calculada pelo Algoritmo 4.1 usando a função gaussiana e $\tau = 0.1$. As políticas de decisão foram avaliadas 20 vezes a partir de cada estado do conjunto de teste.

A Tabela 4.1 mostra os resultados dos algoritmos PISF-M e PISF-S de forma detalhada. Além do desempenho dos algoritmos no conjunto de teste, esta tabela traz também o número médio de passos executados pelas políticas correspondentes a partir da posição de repouso $s = [0, 0]$, que tem interesse particular por representar uma situação mais provável. Observe como de fato os resultados do algoritmo PISF-M não são muito animadores. Com $m = 700$ arquétipos usados na fatoração estocástica, as políticas encontradas por esse algoritmo precisaram de em média 129.04 passos para escapar dos estados do conjunto de teste, um número 74.36% superior àquele requerido pelo algoritmo PI para realizar a mesma tarefa. Quando se considera apenas o estado de repouso, a discrepância nos resultados é ainda maior: embora usando 77.78% dos vetores $\mathbf{m}_i^a$ na fatoração, os agentes do PISF-$M(900, 700)$ precisaram de em média 145.66 passos para escapar do vale, mais do que o dobro do número de passos executados pela política encontrada pelo

algoritmo de iteração de política convencional.

Algoritmo	**Conjunto de teste**			**Posição de repouso**			
	Média	Máx.	Min.	Média	Máx.	Min.	DP
PI(900)	74.01	200	31	71.20	75	67	2.53
PISF-$M(900, 400)$	194.96	200	39	195.75	200	77	19.87
PISF-$M(900, 500)$	186.96	200	38	192.91	200	80	23.61
PISF-$M(900, 600)$	170.39	200	33	178.29	200	74	40.53
PISF-$M(900, 700)$	129.04	200	31	145.66	200	67	52.86
PISF-$S(900, 30)$	194.17	200	56	200.00	200	200	0.00
PISF-$S(900, 60)$	120.40	200	25	186.05	200	85	27.19
PISF-$S(900, 90)$	128.72	200	34	198.05	200	177	6.11
PISF-$S(900, 120)$	78.39	200	19	84.30	95	78	3.56
PISF-$S(900, 150)$	74.46	200	17	79.15	87	75	3.41
PISF-$S(900, 180)$	77.18	200	24	81.40	86	76	3.45
PISF-$S(900, 210)$	70.86	200	24	76.35	89	71	4.26
PISF-$S(900, 240)$	74.46	200	21	77.85	89	75	3.62
PISF-$S(900, 270)$	69.73	169	22	76.25	83	71	3.58

Tabela 4.1: Resultados obtidos pelos algoritmos PI, PISF-M e PISF-S no problema do carro preso no vale. Os valores correspondem ao número de passos usados pelas respectivas políticas de decisão para alcançarem o objetivo do problema. Os detalhes das configurações usadas pelos dois últimos podem ser encontrados nas legendas das Figuras 4.7 e 4.8. A sigla "DP" é usada para denotar o desvio-padrão dos resultados.

Em contraste, note como a partir de $m = 120$ arquétipos o desempenho do algoritmo PISF-S praticamente se iguala ao do algoritmo PI, confirmando a tendência mostrada na Figura 4.8. Particularmente, quando $m = 270$ o número máximo de passos executados pelas políticas do algoritmo PISF-S no conjunto de teste foi 169, o que indica sucesso em *todas* as 10.000 tentativas de escapar do vale (20 execuções $\times$ 25 estados no conjunto de teste $\times$ 20 tentativas a partir de cada estado). Esse resultado não foi alcançado pelo algoritmo PI usando uma discretização com 900 estados. Quando se analisa o desempenho das políticas de decisão partindo da posição de repouso, os resultados do algoritmo PISF-S não chegam a se igualar aos do algoritmo PI, mas também não chegam a decepcionar. Note que com 120 ou mais arquétipos usados na fatoração, as políticas geradas pelo PISF-S não falham *uma única vez* ao tentarem atingir o topo da montanha a partir da

posição $s = [0, 0]$.

Quando analisados assim, de forma superficial, os resultados do algoritmo PISF-S talvez não recebam o crédito merecido. Para que se tenha uma idéia do alcance desses resultados, observe que uma redução do número de estados de $|S| = 900$ para $m = 120$ acarreta uma economia de aproximadamente 99.76% no tempo de computação necessário para avaliar uma política de decisão. Isso significa que se a avaliação de uma política pelo algoritmo PI levasse um dia em uma determinada plataforma computacional, a avaliação correspondente feita pelo algoritmo PISF-$S(900, 120)$ consumiria apenas 3.4 minutos na mesma configuração! Note que no caso em que S é uma malha regular essa redução drástica no tempo de execução vem a um custo relativamente baixo, que são as operações gastas para se calcular as distâncias entre estados e arquétipos. No exemplo hipotético acima, isso poderia ser feito em aproximadamente 25.6 segundos.[3]

Discussão

Os resultados do algoritmo PISF-S tornam-se particularmente impressionantes quando contrastados com aqueles obtidos pelo algoritmo PISF-M. Como a estratégia usada por este último para definir a matriz $\mathbf{W}$ é mais robusta do que a usada pelo primeiro (já que os vetores $\mathbf{w}_i$ não ficam restritos às linhas de $\mathbf{M}$), é natural supor que a diferença no desempenho dos dois algoritmos esteja na determinação da matriz $\mathbf{D}$—o que equivale a dizer que o sucesso do PISF-S é mérito principalmente da atribuição proporcional. Uma maneira de verificar se isso de fato é verdade seria implementar a atribuição proporcional em $M^{|S|}$. Note entretanto que quando os vetores $\mathbf{m}_i^a$ são determinísticos ou quase, como no caso estudado aqui, a atribuição proporcional fica praticamente reduzida à atribuição simples. Uma maneira alternativa de verificar a importância da atribuição proporcional é realizá-la no espaço S com os arquétipos selecionados pelo algoritmo k-medoids em $M^{|S|}$. Usando essa configuração e $m = 120$ arquétipos, o algoritmo PISF obteve uma média de 110.95 passos no conjunto de teste, uma melhoria considerável em relação aos resultados do algoritmo PISF-M mostrados na Tabela 4.1.

Por que isso acontece? Como discutido, a atribuição proporcional caracteriza uma es-

[3]Considerando que a dimensão do espaço S seja $\dim(S) = 2$, que é o caso no problema estudado.

tratégia de associação entre estados e arquétipos mais robusta do que a atribuição simples. Ela pode ser vista como uma maneira de incorporar um certo nível de estocasticidade ao modelo. Imagine por exemplo que os vetores $\mathbf{p}_i^a$ do MDP original sejam todos determinísticos; caso a atribuição proporcional seja adotada, as aproximações $\mathbf{D}^a\mathbf{K}$ darão origem a matrizes de transições $\tilde{\mathbf{P}}^a$ não-determinísticas, já que suas linhas serão combinações convexas dos vetores $\mathbf{k}_i$. Por mais paradoxal que isso possa parecer, a adição de um certo nível de ruído ao modelo pode acabar por torná-lo uma descrição mais fiel do fenômeno original. Isso porque o acréscimo de uma componente aleatória ao processo de modelagem tende a amenizar os efeitos da perda de informação inerente a esse processo. Por exemplo, quando se reduz o número de estados de um MDP perde-se informação a respeito das transições e recompensas associadas com cada estado original. Essa questão torna-se particularmente grave quando um conjunto de estados muito heterogêneo passa a ser reconhecido como um único estado s_i no modelo reduzido. A atribuição proporcional ameniza esse problema "diluindo" uma atribuição *crisp* em várias pertinências no intervalo $[0, 1]$.

Evidentemente, existe uma tensão entre o nível de ruído acrescentado ao modelo e a sua fidelidade ao processo original. Um nível de ruído excessivo pode descaracterizar completamente a dinâmica do problema, tornando o modelo uma representação distorcida sem nenhuma utilidade prática. Por outro lado, uma modelagem puramente *crisp* tende a ser menos robusta em relação à perda de informação. No caso em que a atribuição proporcional é implementada por um *kernel*, como em (4.7), a largura dessa função define o grau de estocasticidade acrescentado ao modelo. Assim como o parâmetro k do algoritmo k-NN, a largura τ pode ser vista como um mecanismo para controlar a tensão entre o *bias* e a variância da aproximação.

Note, no entanto, que mesmo com a atribuição proporcional os resultados do algoritmo PISF-M não se igualam ao do seu concorrente PISF-S. Isso ocorre provavelmente por dois motivos. Em primeiro lugar, pode não fazer muito sentido agrupar dados em espaços de alta dimensionalidade, porque o conceito de vizinhança se esmaece [23, 60]. Além disso, o quase-determinismo dos vetores $\mathbf{p}_i^a$ afeta o funcionamento do algoritmo k-medoids, ou seja, a seleção dos arquétipos usados na aproximação. Portanto, além da matriz $\mathbf{D}$, faz sentido determinar $\mathbf{W}$ no espaço de estados. A transposição da fatoração

estocástica de $M^{|S|}$ para S, além de ser uma solução para a o crescimento superficial, é também uma maneira de contornar a dificuldade de se fatorar estocasticamente matrizes de transições determinísticas.

4.2.5 Aplicabilidade

Estas são as duas condições que devem ser atendidas quando se pretende usar o algoritmo PISF-S para resolver um problema de tomada de decisão seqüencial:

1. É possível representar o espaço de estados S com um número reduzido de variáveis. Isso significa que existe um pequeno grupo de variáveis que originam um conjunto S finito a partir do qual é possível determinar o mapeamento (4.2). O número de variáveis usadas para descrever o problema é considerado "pequeno" se $\dim(S) \ll |S|$. O conjunto S pode ser o resultado de uma discretização, como no caso do carro preso no vale, ou ser naturalmente finito, como no caso em que o MDP representa um jogo de tabuleiro, por exemplo. Obviamente, o MDP resultante do mapeamento (4.2) só é considerado válido se ele descrever o fenômeno de interesse com um grau de precisão tal que as políticas resultantes apresentem um bom desempenho no sistema real.

2. É possível definir uma métrica em S que esteja relacionada com a noção de distância adotada em $M^{|S|}$. Esse requisito é atendido quando os estados s_i puderem ser organizados em um espaço de dimensão $\dim(S)$ de tal forma que uma maior proximidade entre dois estados implique em uma maior semelhança entre as suas probabilidades de transições e recompensas. Na verdade, o que está implícito nessa condição é que os vetores $\mathbf{m}_i^a$ pertençam a um hiper-plano de dimensão $\dim(S)$. A princípio, pode-se pensar em verificar essa condição diretamente, mas o objetivo aqui é justamente evitar a manipulação dos vetores $\mathbf{m}_i^a$, que acarreta um alto custo computacional. Além disso, como discutido, mesmo que a condição acima seja violada esporadicamente, ainda é possível aplicar a fatoração estocástica em S usando a atribuição proporcional.

Resta saber quão realística é a expectativa de que as condições acima sejam atendidas. Quando a dinâmica do sistema é descrita por um conjunto de equações, como é o caso em muitos problemas de controle, é razoável supor que seja possível trabalhar em S diretamente. Como as equações de movimento são normalmente contínuas, existe uma relação topológica natural entre o espaço de estados e o espaço markoviano (veja discussão na próxima seção). Por outro lado, em algumas situações a dimensão do espaço de estados pode ser muito alta (imagine um robô com inúmeros sensores), ou S pode ser desprovido de uma métrica natural (como no caso em que cada estado do sistema representa uma etapa no andamento de um projeto, por exemplo). O que pode ser dito aqui é que quando as condições acima são atendidas e o problema é grande demais para ser resolvido com os recursos computacionais disponíveis, o algoritmo PISF-S pode ser uma alternativa interessante.

4.3 Fatoração Estocástica Baseada em *Kernels*

Nesta seção discutirei em maiores detalhes o caso em que o espaço de estados do fenômeno de interesse é originalmente contínuo. Como será visto, nesse contexto a fatoração estocástica pode ser usada não apenas na resolução de um MDP, como também na sua própria *definição*. Para evitar confusão com o espaço S, que é finito, vou me referir ao espaço de estados contínuo como X, e os estados correspondentes serão denotados por x_i. No caso em que $X \subseteq \mathbb{R}^n$, a dinâmica de um sistema é descrita por uma família de funções $P^a : X \times X \mapsto [0, 1]$ correspondendo às ações $a \in A$. Para todo estado $x_i \in X$, a função de densidade $P^a(\cdot|x_i)$ define as probabilidades de transição a partir de x_i quando a ação a é executada. As recompensas são determinadas por uma função limitada e contínua $R^a : X \times X \mapsto \mathbb{R}$, em que $R^a(x_i, x_j)$ é a recompensa recebida na transição $x_i \xrightarrow{a} x_j$. Na presente discussão considerarei que as funções P^a e R^a são desconhecidas.

De maneira análoga ao caso discreto, cada estado x_i pode ser associado a $|A|$ elementos $\mathbf{m}_i^a \equiv [R^a(x_i, \cdot), P^a(\cdot|x_i)]$. Isso origina um espaço markoviano de dimensão infinita,

que pode ser denotado por M^∞. Note que a continuidade das funções P^a e R^a implica que

$$\left.\begin{array}{l} |P^a(x_k|x_i) - P^a(x_k|x_j)| \to 0 \\ |R^a(x_i,x_k) - R^a(x_j,x_k)| \to 0 \end{array}\right\} \text{ quando } \rho_X(x_i,x_j) \to 0,$$

para todo $x_k \in X$ e todo $a \in A$. Ou seja, existe uma correspondência topológica entre os espaços X e M^∞ que é naturalmente induzida pela continuidade das funções de transição e recompensa. É comum que se imponha condições mais restritivas sobre as funções P^a e R^a de forma a garantir a regularidade do MDP [33, 136, 111], o que em geral resulta em uma relação ainda mais forte entre os espaços X e M^∞. Se essas funções forem Lipschitz contínuas, por exemplo, pode-se escrever

$$\begin{array}{l} |P^a(x_k|x_i) - P^a(x_k|x_j)| \leq K^P \rho_X(x_i,x_j), \\ |R^a(x_i,x_k) - R^a(x_j,x_k)| \leq K^R \rho_X(x_i,x_j), \end{array} \tag{4.10}$$

onde $K^P, K^R \in \mathbb{R}^+$. Não serei muito rigoroso em relação às condições de suavidade das funções de transição e recompensa. O importante aqui é notar que, em geral, quando se trabalha em espaços de estados contínuos, a suposição de que existe uma regularidade topológica de X em relação à M^∞ está implícita na formulação do problema.

Como discutido no Capítulo 2, uma das maneiras de resolver um problema de tomada de decisão seqüencial com espaço de estados contínuo é discretizar X e resolver o MDP resultante. Caso a solução do MDP requeira recursos computacionais além daqueles disponíveis, pode-se pensar em utilizar o algoritmo PISF-S para reduzir o tempo de computação. No entanto, nesta seção discuto uma técnica alternativa para resolver um problema cujo espaço de estados seja contínuo. Essa técnica utiliza a regressão baseada em *kernels* para aproximar as funções P^a e R^a a partir de uma amostra de transições. Como será discutido, tal abordagem pode ser vista como a derivação de um MDP finito que descreve o fenômeno original. Entretanto, em contraste com a discretização convencional, em que a definição da malha é feita *a priori*, na aprendizagem baseada em *kernels* os estados do MDP são definidos a partir das amostras de transições. Uma das vantagens dessa postura é que ela pode ser muito naturalmente combinada com a fatoração estocástica, cuja contribuição para o processo de aprendizagem não fica restrita a um estágio posterior à definição do MDP. Como será visto, a fusão das duas técnicas permite a incorporação de uma grande quantidade de informação a um modelo de tamanho fixo, o que acaba por originar uma descrição compacta do fenômeno em questão.

4.3.1 Aprendizagem por reforço baseada em *kernels*

A aprendizagem baseada em *kernels* foi originalmente proposta por Rust [136] no contexto da programação dinâmica e mais tarde estendida por Ormoneit e Sen [111] para o cenário estudado pela aprendizagem por reforço. Foram estes últimos autores que cunharam o termo *aprendizagem por reforço baseada em kernels* (KBRL[†]). O KBRL permite criar um modelo aproximado de um MDP com espaço de estados contínuo. A sua idéia fundamental consiste em usar um esquema de aproximação não-paramétrico para representar as funções P^a e R^a do MDP. Isso significa que a arquitetura do aproximador não é definida de antemão, e depende inerentemente do conjunto de dados usado na aprendizagem (nesse caso, amostras de transições).

O KBRL apresenta duas propriedades que o colocam em posição de destaque em relação a outros esquemas de aproximação. Em primeiro lugar, ele *sempre converge para uma solução*. Isso contrasta com o comportamento de algumas técnicas de aproximação paramétrica, incluindo escolhas populares como as redes neurais não-lineares, que podem divergir quando usadas para aproximar a função de valor [166, 137, 27, 169, 170]. Além disso, o KBRL é um dos poucos esquemas de aproximação em aprendizagem por reforço que são *consistentes* no sentido estatístico, ou seja, em que um aumento no tamanho do conjunto de treinamento sempre resulta em uma melhoria na qualidade da aproximação.[4] Isso significa que, assintoticamente, a política retornada pela aprendizagem baseada em *kernels* apresentará um desempenho ótimo no problema em questão.

A proposta de Ormoneit e Sen se baseia na cisão do operador básico da programação dinâmica, Υ, em dois operadores mais simples. O operador Γ^a "extrai" a função de valor de ação $Q(x,a)$ de $V(x)$:

$$Q(x,a) = \Gamma^a V(x) = E\left[R^a(x,x') + \gamma V(x')\right], \tag{4.11}$$

em que o valor esperado $E[\cdot]$ é calculado em relação à função de transição P^a. Dada a função de valor Q, o operador Ω faz a operação inversa, através da maximização do valor

[†] *Kernel-based reinforcement learning*

[4] Para uma abordagem semelhante que também apresenta a propriedade de consistência veja o trabalho de Munos e Moore [107].

de x em relação às ações a:

$$V(x) = \Omega Q(x,a) = \max_{a \in A} Q(x,a). \tag{4.12}$$

Fazendo $Q = \Gamma V$, fica claro que a função de valor ótima de um problema corresponde ao ponto fixo da equação $V = \Omega\Gamma V$, ou seja, $\Upsilon = \Omega\Gamma$. A expressão (4.12) pode ser facilmente calculada para qualquer $x_i \in X$. Infelizmente, o mesmo não pode ser dito a respeito de (4.11), cujo cálculo depende das funções P^a e R^a (que supostamente não são conhecidas aqui). A proposta do KBRL é substituir o operador Γ^a por uma versão aproximada $\tilde{\Gamma}^a$, que pode ser calculada a partir de um conjunto de transições.

A aprendizagem por reforço baseada em *kernels* parte de uma premissa simples: para cada ação $a \in A$, existe disponível um conjunto de transições

$$X^a = \{(x_i^a, r_i^a, y_i^a) | i = 1, 2, ..., n_a\},$$

em que y_i^a é um estado resultante da aplicação da ação a em x_i^a e r_i^a é a recompensa associada. As transições em X^a podem ser coletadas através de amostragem ou por uma política de exploração adequada (note que o número de transições n_a referentes a cada ação a pode ser diferente) [111, 110]. Com base nas amostras de transições, pode-se definir o operador $\tilde{\Gamma}^a$ da seguinte maneira:

$$\tilde{\Gamma}^a V(x) = \sum_{i=1}^{n_a} \kappa^a(x, x_i^a) \left[r_i^a + \gamma V(y_i^a) \right], \tag{4.13}$$

onde o *kernel* $\kappa^a(x, x_i^a)$ é definido de forma análoga à (4.7):

$$\kappa^a(x, x_i^a) = \frac{\phi\left(\frac{\rho_X(x, x_i^a)}{\tau_a}\right)}{\sum_{j=1}^{n_a} \phi\left(\frac{\rho_X(x, x_j^a)}{\tau_a}\right)}. \tag{4.14}$$

Intuitivamente, a integral implícita em (4.11) é aproximada em (4.13) por uma soma finita em que a parcela correspondendo à transição iniciada em x_i^a recebe um peso inversamente proporcional à distância $\rho_X(x, x_i^a)$. Essa interpretação deixa claro que, da mesma forma que o algoritmo PISF-S, o KBRL explora a correspondência topológica entre os espaços X e M^∞.

A partir de (4.12) e (4.13) é possível definir o operador de programação dinâmica aproximado $\tilde{\Upsilon} = \Omega\tilde{\Gamma}$. Pode-se mostrar facilmente que, assim como Υ, o operador $\tilde{\Upsilon}$

possui apenas um ponto fixo para $0 \leq \gamma < 1$ [111]. Mas, ao contrário do primeiro, este último não depende do conhecimento de P^a e R^a, e pode ser prontamente aplicado a qualquer função V a partir das amostras de transições X^a. Além disso, $\tilde{\Upsilon}$ apresenta uma grande vantagem do ponto de vista computacional: como o seu cálculo depende apenas dos valores $V(y_i^a)$, qualquer algoritmo iterativo derivado desse operador pode representar a função V armazenando apenas o seu valor nos estados y_i^a. Segue abaixo um exemplo de como isso pode ser feito.

Apenas por uma questão de comodidade, suponha que as amostras de transições tenham sido coletadas executando-se todas as ações $a \in A$ em um conjunto fixo de n estados x_i, ou seja, suponha que

$$n_a = n \;\text{ e }\; x_i^a = x_i \;\text{ para todo }\; a \in A. \tag{4.15}$$

Seja $\mathbf{V}^y$ uma matriz em $\mathbb{R}^{n\times|A|}$ com os valores $V(y_i^a)$, isso é, $v_{ia}^y = V(y_i^a)$. Defina uma matriz $\mathbf{R}^y \in \mathbb{R}^{n\times|A|}$ de maneira análoga. Finalmente, seja $\mathbf{U}$ um tensor em $\mathbb{R}_+^{n\times|A|\times n}$ tal que $u_{iaj} = \kappa^a(y_i^a, x_j)$. Então, a iteração de valor baseada em *kernels* consiste em executar

$$\mathbf{V}_{t+1}^y \leftarrow \Pi\left[\mathbf{U}\left(\mathbf{R}^y + \gamma\mathbf{V}_t^y\right)\right] \tag{4.16}$$

até que $\parallel \mathbf{V}_{t+1}^y - \mathbf{V}_t^y \parallel < \varepsilon$. Na expressão acima, o operador Π aceita como argumento um tensor $\mathbf{Q}^y \in \mathbb{R}^{n\times|A|\times|A|}$ e retorna uma matriz $\mathbf{V}^y \in \mathbb{R}^{n\times|A|}$ em que

$$v_{ij}^y = \max_{a=1,2,\ldots,|A|} q_{iaj}^y.$$

Uma vez encontrada a solução de (4.16), pode-se determinar o valor de qualquer estado $x_i \in X$ a partir da expressão (4.13), ou seja,

$$\tilde{Q}(x,a) = \sum_{i=1}^{n_a} \kappa^a(x, x_i^a)\left[r_i^a + \gamma v_{ia}^y\right]. \tag{4.17}$$

Note que, quando se adota uma função-núcleo ϕ diferenciável, a aproximação (4.17) realizada pelo KBRL é uma função suave. Em geral esse tipo de aproximação gera melhores resultados do que uma função $\tilde{Q}$ contínua por partes, que é o tipo de função de valor originada de uma discretização convencional de X [111].

A aprendizagem por reforço baseada em *kernels* transforma um problema contínuo em uma iteração matricial que converge para uma solução com probabilidade um. Uma

pergunta que surge naturalmente neste contexto se refere à qualidade das soluções encontradas pela iteração (4.16). O leitor provavelmente não ficará surpreso ao ser informado de que isso depende diretamente das amostras de transições X^a. De acordo com Ormoneit e Sen [111], se $n_a \rightarrow \infty$ para todo $a \in A$ e a largura dos *kernels* τ_a diminui a uma taxa de decrescimento "adequada," então a função de valor encontrada pelo KBRL converge em probabilidade para a função ótima do problema, ou seja,

$$\| \tilde{V} - V^* \|_\infty \xrightarrow{P} 0,$$

onde $\tilde{V} = \tilde{\Upsilon}\tilde{V}$.[5] Além disso, os autores mostram que a probabilidade de se escolher uma ação sub-ótima baseando-se em $\tilde{Q} = \tilde{\Gamma}\tilde{V}$ converge para zero nas condições acima. Ormoneit *et al.* caracterizam formalmente o que seria uma taxa admissível para o decrescimento de τ_a para várias escolhas de função ϕ [111, 110].

4.3.2 Derivação de um MDP finito

A aprendizagem por reforço baseada em *kernels* é uma abordagem poderosa para lidar com problemas de tomada de decisão seqüencial em que o espaços de estados é contínuo. No entanto, esse potencial vem a um custo—nesse caso, um custo computacional. O cálculo de cada iteração (4.16) envolve $O(|A|n^2)$ operações aritméticas, uma complexidade computacional inaceitável para muitas aplicações práticas. Se a função-núcleo $\phi(x)$ for não-nula apenas na vizinhança de x, pode-se reduzir esse custo significativamente explorando a esparsidade do tensor $\mathbf{U}$ resultante. Por exemplo, um *kernel* do tipo k-NN reduziria o número de operações em cada execução de (4.16) para $O(|A|kn)$ [111]. Embora esse custo seja mais razoável, a sua dependência em relação ao tamanho n das amostras de transições pode torná-lo proibitivo na prática. Isso porque, dependendo da complexidade do problema em questão, o número de transições requeridas para representar a sua dinâmica satisfatoriamente pode ser muito grande. Além disso, a dependência em relação a n torna o KBRL impraticável para aplicações *on-line*—em que o agente coleta as transições enquanto interage com o ambiente—, porque o desempenho do algoritmo degeneraria a

[5] Além do decrescimento da largura τ_a, Ormoneit e Sen [111] impõem algumas condições sobre a função-núcleo usada para definir o *kernel*, sobre as funções R^a e P^a e sobre a estratégia de amostragem adotada para coletar transições. Nenhuma dessas condições é muito restritiva.

cada iteração.

Nesta seção discuto como a fatoração estocástica pode ser usada para tornar o custo computacional do KBRL independente do tamanho da amostra de dados. As idéias que serão apresentadas aqui estão de tal maneira relacionadas com o exemplo dado na Seção 3.1 que a sua releitura seria aconselhável neste ponto. Como discutido naquela seção, o número de estados de um processo de Markov pode ser reduzido simplesmente redirecionando-se as suas transições para um subconjunto de S. Feito isso, a função de valor de todos os estados do processo passam a depender apenas do valor dos estados pertencentes a esse subconjunto—ou seja, do valor dos arquétipos. A expressão (4.17) deixa claro que quando se adota o operador $\tilde{\Upsilon}$ no lugar de Υ obtém-se um efeito semelhante em um MDP contínuo, pois $\tilde{Q}(x,a)$ passa a depender apenas do subconjunto finito formado pelos estados y_i^a. Assim como feito na Seção 3.1, pode-se pensar em definir um modelo compacto em que figurem apenas as transições e recompensas entre os arquétipos selecionados.

O redirecionamento de transições para os estados y_i^a pode ser formalizado a partir da definição de funções de probabilidade e recompensa auxiliares. Seja $\bar{P}^a$ uma função de transição dada por [110, 69]:

$$\bar{P}^a(x_j|x_i) = \begin{cases} \kappa^a(x_i, x_k^a), \text{ se } x_j = y_k^a, \\ 0, \text{ caso contrário .} \end{cases} \tag{4.18}$$

A função de transição $\bar{P}^a$ pode ser interpretada de uma maneira bastante intuitiva: a probabilidade de transição para o estado y_k^a ao se executar a ação a no estado x_i aumenta à medida que diminui a distância deste último para x_k^a, de onde, sabe-se, já ocorreu essa transição. Note que, por definição, $\sum_{k=1}^{n_a} \bar{P}^a(y_k^a|x_i) = 1$ para todo $x_i \in X$. De maneira análoga a (4.18), pode-se definir a função de recompensa $\bar{R}^a$:

$$\bar{R}^a(x_i, x_j) = \begin{cases} r_k^a, \text{ se } x_j = y_k^a, \\ 0, \text{ caso contrário .} \end{cases} \tag{4.19}$$

Como todas as transições definidas pela função $\bar{P}^a$ terminam no subconjunto de X formado pelos estados y_i^a e, de acordo com $\bar{R}^a$, apenas nessas transições podem ocorrer recompensas não-nulas, é possível derivar um MDP finito composto somente pelos estados y_i^a. Nesse caso, as matrizes de transições $\bar{\mathbf{P}}^a$ e os vetores-recompensa $\bar{\mathbf{r}}^a$ podem ser

facilmente determinados através das expressões (4.18) e (4.19). O cálculo da função de valor $\bar{\mathbf{v}}$ desse modelo reduzido pode ser feito a partir de qualquer algoritmo de programação dinâmica convencional. O algoritmo de iteração de valor, por exemplo, corresponde à expressão (4.16). Uma vez calculada a função $\bar{\mathbf{v}}$, os valores dos estados originais do problema podem ser determinados através de (4.17).

Fixando o tamanho do modelo

Suponha que o número de amostras de transições nos conjuntos X^a seja tal que o MDP resultante não possa ser resolvido com os recursos computacionais disponíveis. Uma opção nesse caso é simplesmente descartar algumas das transições segundo um critério razoável, de forma que o modelo originado seja pequeno o suficiente para ser acomodado na arquitetura computacional em questão. A desvantagem dessa postura é que, por melhor que seja o critério adotado para eliminar transições, existe sempre uma perda de informação ao se descartar dados do problema. Uma outra alternativa para reduzir o custo computacional de se resolver o MDP seria usar o algoritmo PISF-S discutido na última seção. Nesse caso a fatoração estocástica poderia ser efetuada pelo Algoritmo 4.1, ou seja, alguns dos estados y_i^a seriam selecionados como arquétipos e a matriz de desvio $\mathbf{D}$ seria determinada pela atribuição proporcional. Note, no entanto, que como as funções $\bar{P}^a$ e $\bar{R}^a$ permitem o cálculo das probabilidades de transição e das recompensas a partir de *qualquer* estado $x_i \in X$, não há motivo para restringir os arquétipos aos pontos y_i^a.

Considere que um conjunto de m arquétipos $q_j \in X$ tenha sido determinado segundo um critério qualquer. Por exemplo, os arquétipos podem estar uniformemente espalhados pelo espaço X ou serem o resultado de uma clusterização dos estados y_i^a. Uma outra opção seria adotar como arquétipos um subconjunto dos estados coletados por uma política de exploração que visitasse áreas importantes do espaço X. A maneira exata de se selecionar os arquétipos não importa. Uma vez que isso tenha sido feito, no entanto, a concretização da fatoração estocástica depende da determinação das matrizes de desvio $\mathbf{D}^a$—cujos elementos representam as probabilidades de transições dos estados y_i^a para os arquétipos q_j—e das matrizes de retorno $\mathbf{K}^a$, que guardam as probabilidades de transição no sentido contrário. A matriz $\mathbf{K}^a$ pode ser facilmente obtida através da expressão (4.18).

A matriz $\mathbf{D}^a$, por sua vez, pode ser determinada pela atribuição proporcional, dada a correspondência topológica entre os espaços X e M^∞. As probabilidades de transições entre os arquétipos ficam definidas pelas matrizes $\bar{\mathbf{P}}^a = \mathbf{K}^a\mathbf{D}^a$, com $a \in A$ (veja Figura 3.1). Falta ainda determinar as recompensas associadas a essas transições. Por definição, a j-ésima componente de $\bar{\mathbf{r}}^a$ é o valor esperado da recompensa recebida ao se executar a ação a no arquétipo q_j. Logo, esse elemento pode ser calculado como

$$\bar{r}_j^a = \sum_{i=1}^{n_a} k_{ji} r_i^a,$$

o que conclui a definição do novo MDP.

O Algoritmo 4.2 traz uma descrição passo-a-passo do procedimento discutido acima. A única novidade em relação à discussão do último parágrafo diz respeito à definição de dois *kernels* distintos, κ^q e κ^a. Os superescritos se referem ao conjunto de estados usado no cálculo de cada *kernel*. No caso de κ^a são considerados os estados nos conjuntos X^a, como mostra a expressão (4.14). O cálculo de κ^q é feito em relação aos arquétipos, como em (4.7). Note que, como os conjuntos de pontos usados no cálculo das funções κ^q e κ^a são diferentes, as larguras dos *kernels* podem também ser diferentes. Chamarei o Algoritmo 4.2 de *fatoração estocástica baseada em kernels*, ou simplesmente KBSF[†].

Algoritmo 4.2 Fatoração estocástica baseada em *kernels* (KBSF)

Requer X^a, com $a \in A$, $m \in [1, \min_a n_a]$, $\tau_q, \tau_a \in \mathbb{R}_*^+$
Retorna $\bar{\mathbf{P}}^a \in \mathbb{R}^{m\times m}$ e $\bar{\mathbf{r}}^a \in \mathbb{R}^m$, para todo $a \in A$

```
Selecione m arquétipos q_j ∈ X usando um algoritmo qualquer
para a←1 até |A| faça
    r̄^a←0 ∈ ℝ^m
    para i←1 até n_a faça
        para j←1 até m faça
            d^a_ij←κ^q(y^a_i, q_j)          ▷ Determinação dos elementos da matriz D^a
            k^a_ji←κ^a(q_j, x^a_i)          ▷ Determinação dos elementos da matriz K^a
            r̄^a_j←r̄^a_j + k^a_ji r^a_i      ▷ Determinação dos elementos do vetor r̄^a
        fim para
    fim para
    P̄^a←K^a D^a
fim para
```

[†] *Kernel-based stochastic factorization*

Ao contrário do Algoritmo 4.1—que retorna uma fatoração estocástica do MDP que deve ser resolvida pelo algoritmo PISF—, o Algoritmo 4.2 origina um modelo completo, cujas políticas de decisão ótimas podem ser encontradas por qualquer algoritmo de programação dinâmica convencional. A partir da função de valor do MDP reduzido é possível calcular o valor de qualquer estado $y_i^a \in X^a$, como feito na Proposição 3.1:

$$\tilde{V}(y_i^a) = (\mathbf{d}_i^a)^{\mathrm{t}}\, \bar{\mathbf{v}} = \sum_{j=1}^{m} \kappa^q(y_i^a, q_j)\bar{v}_j. \tag{4.20}$$

Note que como os elementos da matriz de desvio $\mathbf{D}^a$ são calculados a partir do *kernel* κ^q, não há motivo para restringir a expressão (4.20) aos pontos y_i^a. Assim, é possível generalizar a idéia de uma matriz de desvio para uma "função de desvio" que fornece o valor de qualquer um dos estados $x_i \in X$:

$$\tilde{V}(x_i) = \sum_{j=1}^{m} \kappa^q(x_i, q_j)\bar{v}_j. \tag{4.21}$$

O esquema de aproximação acima pode ser interpretado como uma rede de funções de base radiais [121]. Nesse caso, os centros das RBFs coincidiriam com os arquétipos e os pesos da camada de saída corresponderiam ao vetor $\bar{\mathbf{v}}$.

O custo computacional do algoritmo KBSF é dominado pelo cálculo das matrizes $\bar{\mathbf{P}}^a$, que é $O(m^2 n_{\max}|A|)$, onde $n_{\max} = \max_{a\in A} n_a$. A definição do MDP realizada pelo KBRL, por outro lado, requer $O(n_{\max}^2|A|)$ operações aritméticas para o cálculo da expressão (4.18) para cada estado $y_i^a \in X^a$ e cada ação $a \in A$. A diferença fundamental é que o número de estados no modelo gerado pelo KBRL também cresce com $n_{\max}$, ao passo que o Algoritmo 4.2 gera um modelo de tamanho fixo, com m arquétipos, independentemente do número de transições nas amostras X^a.

A fatoração estocástica baseada em *kernels* permite que uma quantidade arbitrária de informação seja incorporada ao MDP sem que haja um aumento exorbitante do custo computacional de se resolvê-lo. Intuitivamente, pode-se dizer que à medida que $n_a \to \infty$ para todo $a \in A$, as probabilidades de transições e as recompensas entre os arquétipos q_j tornam-se cada vez mais próximas dos seus valores no MDP original, contínuo. Isso, é claro, se as larguras dos *kernels* κ^a e κ^q forem ajustadas de acordo, nos moldes do que ocorre com o KBRL. Evidentemente, se os arquétipos não forem representativos da dinâmica do problema original, o resultado da aproximação (4.21) será insatisfatório,

por maior que seja o número de amostras de transições disponíveis. Por outro lado, um conjunto de arquétipos bem escolhido pode resultar em uma aproximação pobre se os exemplos de transições não forem suficientes para caracterizar a dinâmica entre eles.

Fica claro, portanto, que no algoritmo KBSF os arquétipos e as amostras de transições cumprem papéis diferentes, porém complementares. Enquanto os primeiros são responsáveis por criar um modelo compacto do MDP contínuo, as amostras de transições servem para incorporar informação a esse modelo, tornando-o cada vez mais fiel ao problema original. Pode-se dizer que os arquétipos determinam a estrutura do MDP reduzido, ao passo que os exemplos de transições ficam responsáveis pela definição da sua dinâmica. No KBRL as duas funções são exercidas pelo mesmo conjunto de estados. Ao desacoplar "forma" e "conteúdo," a fatoração estocástica baseada em *kernels* permite que se lide com o problema de aproximação de uma maneira mais racional, já que a estrutura do aproximador torna-se independente da definição dos seus parâmetros. Em um certo sentido, essa estratégia pode ser vista como uma maneira de atribuir ao KBRL as características desejáveis de uma aproximação paramétrica. Note, no entanto, que em contraste com esta última, a técnica de aproximação descrita nesta seção sempre converge para uma solução.

4.3.3 Análise empírica

Nesta seção os algoritmos KBRL e KBSF são comparados. O problema escolhido para avaliar os dois métodos foi o carro preso no vale com $\sigma = 1$. Isso permite comparações diretas com os resultados mostrados na seção anterior. No caso do KBRL, o algoritmo de aprendizagem se reduz à aplicação da iteração (4.16) aos estados y_i^a. Já no caso da fatoração estocástica baseada em *kernels*, o Algoritmo 4.2 foi usado para gerar um MDP composto pelos arquétipos q_j, que foi posteriormente submetido ao algoritmo de iteração de valor convencional. Tanto no KBRL quanto no KBSF o processo iterativo foi interrompido com base em (2.16), usando $\varepsilon = 0.01$ e $\gamma = 0.995$. Também em ambos os casos a função gaussiana cumpriu o papel de função-núcleo, e a métrica adotada foi aquela induzida pela norma euclidiana em X.

As mesmas amostras de transições foram compartilhadas por ambos os algoritmos.

Os conjuntos X^a foram gerados de acordo com (4.15), com os estados x_i^a amostrados de uma distribuição uniforme em $[-1.2, 0.5] \times [-0.07, 0.07]$. Para cada valor de n em $\{100, 200, 300, 400, 500\}$ foram geradas 20 amostras X^a diferentes, com $a \in A$. Os resultados do algoritmo KBRL correspondem a uma média sobre esses 20 conjuntos. O algoritmo KBSF foi executado 3 vezes em cada conjunto de amostras X^a, cada uma com um valor diferente para o parâmetro m. Em particular, dado um n fixo, foram utilizados valores de m correspondendo a 50%, 30% e 10% do número de transições em X^a. Os arquétipos foram posicionados de maneira uniforme no espaço de estados bidimensional. Mais precisamente, $m_2 = \lfloor \sqrt{m} \rfloor$ arquétipos foram organizados em uma malha e os $m - m_2^2$ restantes foram distribuídos de maneira aleatória em X.

Como o tamanho n das amostras X^a variou ao longo dos experimentos, foi necessário desenvolver uma estratégia para determinar a largura dos *kernels* de acordo com a distribuição dos pontos x_i^a. Isso foi feito de forma a garantir alguma sobreposição das funções gaussianas no espaço X. Especificamente, a largura τ_a foi definida para cada X^a da seguinte maneira: primeiro, foi calculada a distância média de todos os estados x_i^a aos seus 3 vizinhos mais próximos no conjunto X^a (note que $3 = \dim(S) + 1$). Chamando essa distância de $\bar{\rho}$, a largura τ_a foi determinada de forma que $\kappa^a(x_i^a, x_j) = \varpi_a$ quando $\rho_S(x_i^a, x_j) = \bar{\rho}$, em que ϖ_a é um parâmetro dado. Essa estratégia permite que o mesmo nível de sobreposição dos *kernels* seja usado independentemente do número e da distribuição dos estados no conjunto X^a. O parâmetro τ_q foi determinado de maneira análoga, utilizando os arquétipos q_j no lugar dos pontos x_i^a e ϖ_q no lugar de ϖ_a. Foram testados valores de ϖ_a e ϖ_q no conjunto $\{0.01, 0.1, 0.3\}$. Após alguns experimentos preliminares, ficou constatado que os algoritmos apresentam o melhor desempenho quando $\varpi_a = 0.3$ e $\varpi_q = 0.01$. É importante frisar, no entanto, que resultados razoáveis foram obtidos com *todas* as combinações de valores de ϖ_a e ϖ_q no conjunto acima.

A Tabela 4.2 mostra os resultados obtidos pelos algoritmos KBRL e KBSF no problema do carro preso no vale. Várias observações interessantes podem ser feitas a respeito desta tabela. Inicio com as triviais. Note como o desempenho do algoritmo KBRL(n) melhora com o aumento do número n de transições nos conjuntos X^a. Isso está de acordo com a afirmação anterior de que esse algoritmo é consistente no sentido estatístico. O algoritmo KBSF(n, m) também apresenta um comportamento bastante estável. Ao se fi-

xar a proporção m/n, os resultados desse algoritmo melhoram monotonicamente com o aumento de n. Da mesma forma, dado um n fixo, um aumento de m resulta em geral em uma melhoria do desempenho do KBSF (com apenas duas exceções).

No entanto, o que salta aos olhos na Tabela 4.2 é o fato de em alguns casos os resultados do KBSF serem *melhores* do que os do KBRL. Em particular, dado um n fixo, isso é sempre verdade quando $m = 0.5n$ ou $m = 0.3n$. Esse fenômeno é de certa maneira surpreendente, uma vez que ambos os algoritmos utilizaram os mesmos conjuntos de dados e os MDPs gerados pelo KBSF são bem menores do que aqueles construídos pelo KBRL. O que parece estar acontecendo aqui é que os modelos gerados pelo KBSF são mais bem estruturados do que os do KBRL original. Note que os arquétipos q_j foram distribuídos de maneira regular sobre o espaço de estados X, ao passo que os pontos x_i^a estão distribuídos de maneira aleatória.[6] Como discutido, tanto os estados x_i^a quanto os arquétipos q_j servem como "pontos de referência," para o cálculo de $V(x_i)$, para todo $x_i \in X$ (veja equações (4.17) e (4.21)). Aparentemente, em alguns casos é vantajoso sacrificar alguns desses pontos de referência em nome de uma melhor distribuição dos remanescentes. Além disso, uma melhor distribuição dos estados do modelo facilita a definição da largura dos *kernels*.

Como observa Gordon [54] em um contexto ligeiramente diferente, existe uma tensão na definição da largura dos *kernels* usados na aproximação que depende da distribuição dos estados do modelo reduzido. Larguras maiores tendem a originar aproximações mais suaves, mas distorcem mais a dinâmica do processo original, uma vez que há uma probabilidade razoável de transições ocorrerem entre qualquer par de estados (veja expressão (4.18)). Por outro lado, larguras pequenas tendem a aumentar a variância da aproximação, além de originarem MDPs com estados auto-recorrentes. Quando os estados do modelo estão separados por uma distância regular, como no caso dos arquétipos q_j, é razoável esperar que exista um valor para o parâmetro τ_q que ofereça um bom compromisso entre as duas situações acima. Entretanto, quando os estados estão posicionados de maneira não-uniforme, como é o caso dos estados x_i^a, a escolha de um valor específico

[6] Observe a diferença entre "estados distribuídos de maneira regular" e "estados amostrados de uma distribuição uniforme." Enquanto no primeiro caso os estados encontram-se a uma distância regular um do outro, no segundo caso eles foram amostrados de uma distribuição em que todo $x_i \in X$ tem a mesma probabilidade de ser selecionado.

Algoritmo	**Conjunto de teste**			**Posição de repouso**			
	Média	Máx.	Min.	Média	Máx.	Min.	DP
KBRL(100)	107.67	200	18	153.29	200	70	53.81
KBSF$(100, 10)$	113.99	200	18	192.87	200	83	23.68
KBSF$(100, 30)$	104.40	200	18	166.43	200	76	48.99
KBSF$(100, 50)$	112.69	200	18	183.95	200	70	37.41
KBRL(200)	100.77	200	18	118.57	200	67	53.62
KBSF$(200, 20)$	110.22	200	18	178.24	200	79	41.97
KBSF$(200, 60)$	91.79	200	18	121.98	200	68	54.61
KBSF$(200, 100)$	85.24	200	18	99.19	200	70	39.80
KBRL(300)	82.38	200	18	85.92	200	66	29.04
KBSF$(300, 30)$	99.17	200	15	156.85	200	71	54.03
KBSF$(300, 60)$	77.64	200	18	87.26	200	65	28.56
KBSF$(300, 150)$	74.61	200	19	78.23	189	66	13.44
KBRL(400)	79.83	200	16	83.57	200	67	27.09
KBSF$(400, 40)$	87.65	200	17	123.17	200	67	52.41
KBSF$(400, 120)$	74.07	200	18	77.50	191	67	10.38
KBSF$(400, 200)$	72.75	200	16	76.04	200	67	8.62
KBRL(500)	76.46	200	17	77.38	175	68	8.70
KBSF$(500, 50)$	84.74	200	17	107.80	200	66	46.90
KBSF$(500, 150)$	70.33	200	17	75.08	91	66	4.32
KBSF$(500, 250)$	72.48	200	18	75.38	152	66	5.83

Tabela 4.2: Resultados obtidos pelos algoritmos KBRL e KBSF no problema do carro preso no vale. Os valores correspondem ao número de passos usados pelas respectivas políticas de decisão para alcançarem o objetivo do problema. O primeiro valor entre parênteses após o nome dos algoritmos representa o número n de transições em cada conjunto X^a. O segundo número é m, a quantidade de arquétipos usados na aproximação. Para cada combinação de valores de n e m foram realizadas 20 execuções independentes dos algoritmos. Após cada execução, a política encontrada foi testada 20 vezes em cada estado do conjunto de teste.

para τ_a terá implicações diferentes sobre cada um deles, o que pode acabar por prejudicar a aproximação. Pode-se evidentemente pensar em usar um *kernel* diferente para cada ponto x_i^a, mas a determinação de uma largura adequada para cada um deles complica de maneira significativa a definição do modelo.

Para verificar se a distribuição dos estados é mesmo a explicação para os melhores resultados do KBSF em relação ao algoritmo KBRL, o experimento anterior foi repetido, agora com os estados $x_i^a \in X^a$ posicionados de maneira uniforme. De fato, nesse caso os resultados do KBRL foram sempre superiores aos do KBSF.[7] Note, no entanto, que nem sempre é possível gerar estados x_i^a distribuídos de maneira regular pelo espaço de estados. Isso ocorre, por exemplo, quando não se tem um modelo do sistema e os estados têm que ser coletados diretamente por uma política de exploração. Nessa situação, pode-se esperar que o uso do algoritmo KBSF resulte não apenas em uma economia de tempo de computação, como também em uma melhoria no desempenho das políticas de decisão encontradas.

Uma pergunta natural quando se adota o KBSF em lugar do KBRL é quão longe pode ir a redução do modelo sem que o desempenho da política resultante seja comprometido. Essa questão claramente depende do domínio da aplicação e da distribuição dos dados. No caso do carro preso no vale com transições amostradas de uma distribuição uniforme, um número de arquétipos correspondendo a 30% do tamanho da amostra não ocasiona qualquer degeneração dos resultados, como pode ser verificado na Tabela 4.2. Note que isso resulta em um algoritmo aproximadamente 11 vezes mais rápido do que o original. No entanto, quando o número de arquétipos m é reduzido para 10% do valor de n, há uma queda nítida de desempenho. A partir desse ponto existe uma tensão entre economia de tempo de computação e qualidade dos resultados obtidos. A expectativa é que essa questão surja mais cedo ou mais tarde, qualquer que seja o domínio da aplicação.

Embora a forma como os resultados do algoritmo KBSF foram apresentados seja instrutiva, ela não reproduz com fidelidade o que ocorreria em um cenário real. Em uma situação real, só faz sentido pensar em reduzir o MDP gerado pelo KBRL se o tamanho das amostras X^a tornar proibitiva a sua solução direta. Portanto, seria mais realístico

[7] Não mostro os resultados deste experimento para evitar uma proliferação de valores que acabe por dificultar a análise.

supor que os resultados do KBRL(n) não são alcançáveis. Para reproduzir este cenário apresento um experimento que ilustra de maneira mais clara a contribuição potencial do algoritmo KBSF. Suponha que os recursos computacionais disponíveis limitem o tamanho do modelo a no máximo $n = 100$ estados. Caso os estados y_i^a sejam amostrados de uma distribuição uniforme, o melhor desempenho que se pode esperar das políticas geradas pelo algoritmo KBRL(100) no problema do carro preso no vale é aquele mostrado na Tabela 4.2. No entanto, se for possível gerar amostras extras de transições, pode-se melhorar substancialmente esses resultados mantendo o tamanho do MDP fixo em 100 estados. Isso é feito, obviamente, utilizando-se o algoritmo KBSF.

A Figura 4.9 mostra os resultados do algoritmo KBSF($\cdot, 100$) no problema do carro preso no vale utilizando amostras X^a de tamanhos crescentes. Note como o desempenho das políticas de decisão encontradas melhora significativamente com o aumento de n. Em particular, quando $n = 500$ e $m = 100$, o número médio de passos executados por essas políticas é 72.23, um resultado melhor do que aquele que seria encontrado pelo KBRL(500) de acordo com a Tabela 4.2. Só para que se tenha uma idéia da economia computacional alcançada aqui, o número de operações executadas pelo algoritmo KBSF($500, 100$) a cada iteração corresponde a apenas 4% das operações executadas pelo KBRL(500). Além disso, como o modelo gerado pelo primeiro é substancialmente menor, é razoável esperar que a convergência desse algoritmo ocorra em um número inferior de iterações [92].

Discussão

A aprendizagem por reforço baseada em *kernels* pode ser vista como uma técnica de discretização suave de um MDP contínuo. Em contraste com a discretização convencional, em que normalmente a definição dos estados determina a coleta posterior de transições, no KBRL as transições é que definem os estados do MDP discreto. A largura do *kernel* usado pelo KBRL está claramente relacionada com a área de cada célula usada na discretização convencional. No entanto, a região associada com cada arquétipo não tem necessariamente limites bem definidos, o que equivale a dizer que um estado x_i pode ser representado por vários estados y_i^a. A analogia com a atribuição proporcional é

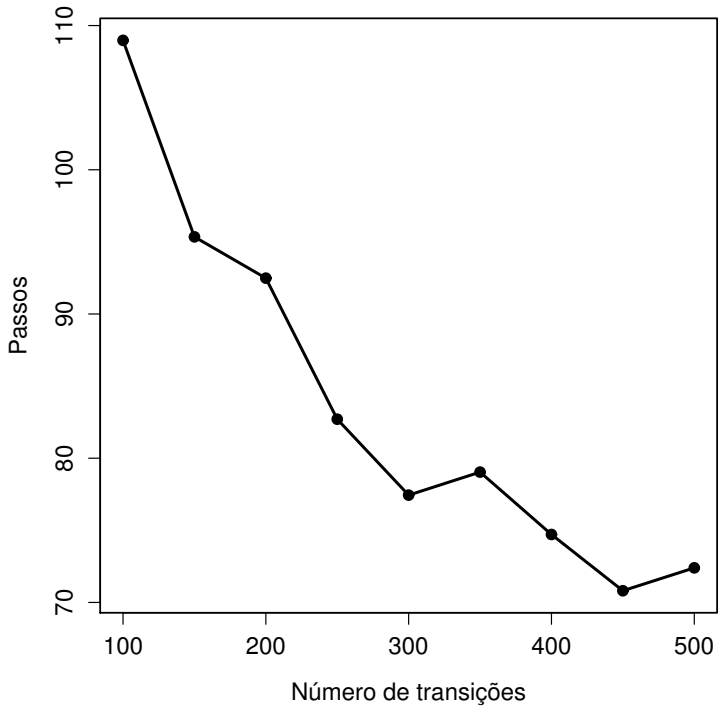


Figura 4.9: Resultados do algoritmo KBSF no problema do carro preso no vale usando um conjunto fixo de 100 arquétipos e um número crescente de transições. Cada ponto corresponde ao número médio de passos usados pelas políticas de decisão para escapar dos 25 estados do conjunto de teste. As políticas foram testadas 20 vezes a partir de cada um desses estados. Os pontos correspondem a uma média obtida em 20 execuções independentes do algoritmo.

inevitável. Como discutido na seção anterior, o acréscimo de um certo grau de estocasticidade ao processo de modelagem pode servir como um fator atenuante dos efeitos de uma eventual perda de informação. Isso permite a definição de modelos mais compactos. De fato, quando se observa os resultados do algoritmo PI(900), mostrados na Tabela 4.1, percebe-se que eles representam uma melhoria modesta em relação ao KBRL(500), cuja discretização foi feita com pouco mais do que a metade da resolução.

Da mesma forma que o KBRL está relacionado com a discretização convencional, existe um nítido paralelo entre o algoritmo KBSF e o algoritmo PISF. Em particular, ambos utilizam a fatoração estocástica para gerar um modelo compacto do fenômeno de interesse. A diferença fundamental é que o PISF trabalha sobre uma discretização pré-existente, enquanto que o KBSF gera uma discretização original. Considerando a

discussão do último parágrafo, seria concebível aplicar o algoritmo PISF à discretização efetuada pelo KBRL. A vantagem dessa abordagem seria que os resultados teóricos do Capítulo 3 forneceriam uma cota superior para a diferença entre as soluções encontradas pelo PISF e aquelas que seriam retornadas pelo KBRL usando todas as transições disponíveis—garantia essa inexistente para o algoritmo KBSF. Note, no entanto, que empiricamente este último parece gerar resultados melhores do que o próprio KBRL, o que torna a opção pelo algoritmo PISF um tanto questionável. De fato, quando se compara os valores mostrados nas Tabelas 4.1 e 4.2, percebe-se por exemplo que o algoritmo KBSF(300,60) apresenta um desempenho similar ao do algoritmo PISF(900,120), mesmo usando a metade dos arquétipos sobre uma discretização de qualidade inferior.

4.3.4 Aplicabilidade

Ao longo desta seção considerei que o espaço de estados em questão era contínuo. Essa é uma restrição imposta pela aprendizagem por reforço baseada em *kernels*, que serviu de inspiração para o algoritmo KBSF. Portanto, se existe qualquer expectativa de estender os resultados teóricos de Ormoneit e Sen [111] para os modelos gerados pela fatoração estocástica, seria prudente ater-se a esse cenário. Pragmaticamente falando, no entanto, a continuidade do espaço de estados não é crucial: uma análise do Algoritmo 4.2 revela que a única restrição em relação ao uso do KBSF é que seja possível calcular uma medida de dissimilaridade entre estados e arquétipos (veja equações (4.7) e (4.14)). Isso aumenta consideravelmente o número de casos em que o KBSF pode ser aplicado. Por exemplo, esse algoritmo pode ser usado quando S é um espaço métrico e os arquétipos representam um subconjunto desse espaço. Ou, mais genericamente, quando os arquétipos são selecionados de um espaço $S' \supset S$ com uma métrica bem definida. Não custa lembrar que nesses casos as observações feitas na Seção 4.2.5 também se aplicam.

4.4 Experimentos Computacionais

Uma brincadeira comum entre crianças consiste em tentar equilibrar uma vassoura de cabeça para baixo sobre uma das mãos espalmada. Em geral, a brincadeira se desenvolve da mesma maneira: após algumas tentativas iniciais totalmente frustradas, o aspirante a equilibrista melhora sucessivamente o seu desempenho até o ponto em que consegue manter a vassoura no ar pelo tempo que desejar. O problema estudado nesta seção é uma reprodução grosseira desse cenário, com a vassoura substituída por um bastão e a criança por uma política de decisão.

4.4.1 Equilibrando um bastão

A tarefa de equilibrar um bastão é mostrada de forma esquemática na Figura 4.10. Como pode ser visto, nessa formulação do problema o bastão encontra-se afixado a um carro que se move sobre trilhos. Em geral, considera-se que o bastão esteja preso em uma polia que permite que ele se movimente em um plano paralelo aos trilhos. O objetivo é manter o bastão equilibrado pelo máximo de tempo possível sem deixar o carro atingir os limites laterais. A cada instante de tempo existem três ações possíveis: empurrar o carro para a esquerda com uma força de $10\mathrm{N}$, empurrá-lo para a direita com a mesma força ou não aplicar força nenhuma. O espaço de estados do problema é constituído por quatro variáveis contínuas representando a posição x e a velocidade $\dot{x}$ do carro, o ângulo θ entre o bastão e o carro e a velocidade angular $\dot{\theta}$ do bastão. Aqui a tarefa foi modelada como um problema descontado: a recompensa em todos os passos é nula, a não ser quando $|x| > 2.4\mathrm{m}$ ou $|\theta| > 36^o$, quando uma recompensa de -1 é entregue e o episódio é interrompido. O fator de desconto adotado foi $\gamma = 0.99$. Para os experimentos desta seção e da próxima foi utilizado um simulador realístico que inclui a fricção exercida pelos componentes do sistema uns nos outros. Os detalhes da implementação podem ser encontrados no Apêndice B.

Embora tenha uma formulação extremamente simples, a tarefa de equilibrar um bastão apresenta uma dinâmica altamente não-linear, o que constitui um obstáculo para muitas técnicas tradicionais de controle. Além disso, trata-se de um sistema inerentemente

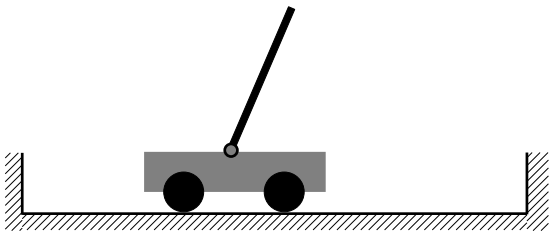


Figura 4.10: O problema de equilibrar um bastão sobre um carro.

instável que é representativo de uma grande classe de problemas. Por esses motivos a tarefa de equilibrar um bastão vem despertando o interesse de pesquisadores de diversas áreas por mais de 40 anos [183, 130]. Atualmente, a quantidade acumulada de trabalhos tratando das mais diferentes versões desse problema é extensa demais para ser citada de uma maneira exaustiva. Para o leitor interessado no assunto, um bom começo é o artigo de Wieland [184], que traz um breve histórico e diversas referências para outros trabalhos. Em se tratando da aprendizagem por reforço especificamente, o problema de equilibrar um bastão sobre um carro foi estudado em trabalhos que hoje já podem ser considerados clássicos da área [99, 10, 1, 2]. A versão *on-line* do problema também foi abordada recentemente com técnicas de aprendizagem por reforço [161, 7]. Além disso, é interessante mencionar que o problema de equilibrar um bastão é um dos favoritos entre os pesquisadores que estudam o uso de algoritmos evolucionários para o projeto de controladores, que hoje acumulam uma grande quantidade de trabalhos sobre o assunto [184, 179, 141, 106, 154, 67, 51, 180].

O cenário que proponho para os experimentos desta seção é o mais geral possível: eu considero que a única informação disponível sobre o problema seja um conjunto de transições coletadas por uma política de exploração π_e. Note que essa estratégia pressupõe apenas a possibilidade de interação com o problema ou com um simulador. Nesse caso especificamente os dados foram gerados a partir de 1000 tentativas de equilibrar o bastão feitas por uma política π_e que seleciona cada uma das três ações disponíveis com a mesma probabilidade. Cada tentativa foi iniciada em um estado sorteado uniformemente de:

$$\begin{array}{ll} x \in [-1.8, 1.8] m & \dot{x} \in [-1.8, 1.8] m/s \\ \theta \in [-3\pi/20, 3\pi/20]\, rad & \dot{\theta} \in [-3\pi/20, 3\pi/20]\, rad/s. \end{array} \tag{4.22}$$

Os limites dos intervalos acima correspondem a 75% dos valores extremos permitidos para as variáveis x e θ. Os detalhes da amostra de transições usada nos experimentos

são mostrados na Tabela 4.3. Como pode ser observado, o número médio de passos executados por π_e foi 94.79, que no simulador usado corresponde a aproximadamente 2 segundos de tempo simulado (veja detalhes no Apêndice B). Ficou determinado então que o objetivo do problema seria equilibrar o bastão por um minuto, ou 3000 passos. Para avaliar as políticas de decisão encontradas foi usado um conjunto de teste formado por 81 estados distribuídos de maneira regular sobre o espaço definido pelos intervalos (4.22). Através de simulações ficou constatado que é possível equilibrar o bastão a partir de todos os estados desse conjunto.

Número de episódios	1000
Número de transições	94787
Número médio de transições por episódio	94.79
Número máximo de transições por episódio	476
Número mínimo de transições por episódio	19
Desvio-padrão do número de transições por episódio	65.67
Transições em que a ação $-10N$ foi executada	33.19%
Transições em que a ação $0N$ foi executada	33.55%
Transições em que a ação $+10N$ foi executada	33.26%

Tabela 4.3: Informações sobre a amostra de transições usada no problema de equilibrar um bastão.

Discretização convencional

A primeira tentativa de resolver o problema foi discretizar o espaço de estados de maneira uniforme. Aqui considerei não haver qualquer informação a respeito do problema além das amostras de transições, o que impede discretizações mais inteligentes como por exemplo a usada por Barto *et al.* [10]. As probabilidades de transições entre as partições e o valor esperado das recompensas foram calculadas como em (2.17), ou seja:

$$\tilde{P}^a(s_j|s_i) = \frac{|X_{ij}^a|}{|X_i^a|} \quad \text{e} \quad \tilde{r}^a(s_i) = \frac{1}{|X_i^a|} \sum_{r^a \in X_i^a} r^a \ ,$$

onde o conjunto X_i^a contém as transições em X^a iniciadas na região de X correspondendo ao estado s_i e X_{ij}^a é o subconjunto de X_i^a formado pelas transições terminadas na partição s_j.

Foram realizados experimentos com discretizações de diversas resoluções. Uma vez definidos os MDPs, o algoritmo de iteração de política foi utilizado para encontrar uma política de decisão. Esse algoritmo foi adotado em todos os experimentos desta seção, e em todos os casos ele foi executado por um número máximo de 15 iterações. A Tabela 4.4 mostra o desempenho das políticas de decisão correspondentes a diferentes discretizações do espaço de estados X. Note que os valores entre parênteses correspondem ao número de estados resultante de discretizações uniformes usando $\iota = 3, 4, 5, 6, 7$ intervalos em cada dimensão de X.

Algoritmo	**Ep.** $(\%)$	**Média**	**Máx.**	**Mín.**	**DP**
PI(81)	0.00	16.64	46	13	3.63
PI(256)	76.54	2316.58	3000	56	1242.34
PI(625)	13.58	1275.88	3000	20	1027.60
PI(1296)	79.01	2383.14	3000	15	1204.47
PI(2401)	18.52	1389.58	3000	17	1049.52

Tabela 4.4: Resultados obtidos no problema de equilibrar um bastão pelas políticas derivadas de diferentes discretizações do espaço de estados. Os valores correspondem ao número de passos executados no conjunto de teste. A coluna "Ep." se refere à proporção de episódios em que o bastão foi equilibrado por 3000 passos.

O que mais chama a atenção na Tabela 4.4 é o fato de o desempenho das políticas não melhorar monotonicamente com o aumento do número n de estados dos MDPs. No entanto, uma análise mais cuidadosa mostra que neste contexto isso não é absurdo: como a mesma amostra de transições foi usada em todos os casos, um aumento de n não significa incorporação de mais informação ao modelo. Nesse caso o padrão usado na discretização pode ser mais importante do que o número de partições utilizadas para fazê-lo. Os dados mostrados na Tabela 4.4 indicam que discretizações em que o número ι de intervalos usados é par tendem a gerar melhores resultados no problema de equilibrar um bastão. Isso sugere que a capacidade de distinguir estados fronteiriços aos eixos de X é uma característica importante para derivar uma boa estratégia de controle.

Um outro fenômeno interessante quando se usa uma amostra de transições fixa é a ocorrência de partições sem nenhuma informação associada. Quando $n = 2401$, por exemplo, cerca de 69.76% das partições em que foi dividido o espaço X são "vazias" (ou

seja, nenhum dos estados x_i que ocorrem nas 94787 transições da amostra pertencem à região de X correspondente a essas partições). Isso resulta em "estados-fantasmas," em que a probabilidade de transição para qualquer outro estado é zero, independentemente da ação executada. Embora esse problema seja mais grave quando se usa uma resolução alta na discretização, ele ocorre mesmo quando o número de intervalos usados é pequeno: quando $n = 81$, os estados-fantasmas ainda representam cerca de 37.04% dos estados do modelo.

A presença de estados-fantasmas claramente distorce a dinâmica do MDP, e é razoável supor que ela também prejudique a qualidade da política resultante. Note que simplesmente eliminar os estados-fantasmas do modelo pode não ser uma boa solução, porque embora não existam transições "saindo" desses estados, provavelmente existem transições terminando em muitos deles. Além disso, caso os estados-fantasmas sejam excluídos, surge uma dúvida em relação à ação que deve ser executada nas regiões do espaço de estado correspondentes às partições eliminadas. A fatoração estocástica oferece uma solução simples e elegante para essa questão.

Nos exemplos anteriores deste capítulo a seleção dos arquétipos para a fatoração estocástica foi feita sempre segundo um critério espacial. Nesta seção eu mostro que existem outras possibilidades, através de um exemplo em que o critério de seleção é a "confiabilidade" dos estados do modelo original. A idéia é a seguinte: quanto maior o número de transições associadas com cada partição s_i, mais confiáveis são os vetores $\mathbf{p}_i^a$ correspondentes, e o mesmo pode ser dito em relação às recompensas r_i^a. Portanto, faz sentido priorizar a presença desses estados no modelo reduzido. Assim, pode-se usar uma versão do algoritmo PISF em que a seleção de m arquétipos se reduz à identificação dos m estados s_i que apresentem mais transições associadas. Note que essa estratégia concentra os recursos do aproximador nas regiões mais visitadas do espaço de estados, excluindo naturalmente os estados-fantasmas do modelo. Observe também que nesse caso a ação a ser executada em cada estado excluído pode ser facilmente determinada ao final do processo, através da aplicação direta da Proposição 3.1.

Embora nesse caso a seleção dos arquétipos seja feita independentemente do espaço de estados S, a atribuição proporcional ainda ocorre nesse espaço, e portanto parece ra-

zoável referenciar o algoritmo como PISF-S. Nos experimentos desta seção a função gaussiana foi utilizada como função-núcleo. A largura dos *kernels* foi determinada como descrito na Seção 4.3.3, com $\varpi_q = 0.01$ e 5 vizinhos usados no cálculo de τ_q. Os resultados obtidos pelo algoritmo PISF-$S(n,m)$ sobre algumas das discretizações anteriores podem ser vistos na Tabela 4.5. Para cada valor de n, foram gerados modelos em que $m = 0.8n, 0.6n, 0.4n, 0.2n$.

Algoritmo	**Ep.**(%)	**Média**	**Máx.**	**Mín.**	**DP**
PISF-S(256,51)	0.00	105.62	1280	16	180.34
PISF-S(256,102)	71.60	2200.19	3000	46	1279.13
PISF-S(256,154)	92.59	2785.43	3000	46	763.46
PISF-S(256,205)	80.25	2439.04	3000	46	1138.26
PISF-S(625,125)	0.00	31.56	92	14	14.72
PISF-S(625,250)	0.00	35.95	158	15	24.45
PISF-S(625,375)	76.54	2381.17	3000	18	1150.25
PISF-S(625,500)	96.30	2889.74	3000	20	565.72
PISF-S(1296,259)	0.00	28.19	97	13	17.72
PISF-S(1296,518)	66.67	2022.09	3000	15	1392.16
PISF-S(1296,778)	81.48	2452.80	3000	16	1155.06
PISF-S(1296,1037)	100.00	3000.00	3000	3000	0.00
PISF-S(2401,480)	0.00	60.01	358	13	74.3
PISF-S(2401,960)	49.38	1582.31	3000	16	1445.94
PISF-S(2401,1441)	90.12	2705.49	3000	15	895.18
PISF-S(2401,1921)	91.36	2742.22	3000	15	843.35

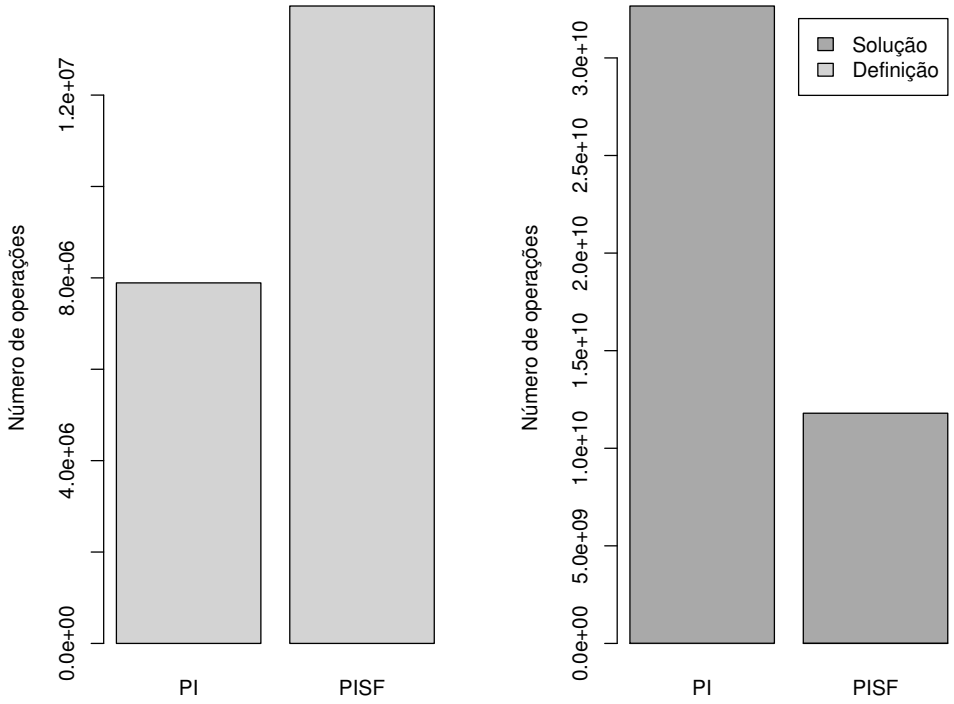
Tabela 4.5: Resultados obtidos pelo algoritmo PISF-$S(n,m)$ no problema de equilibrar um bastão. O parâmetro n se refere ao número de estados da discretização original e m é o número de arquétipos do modelo reduzido. Os valores correspondem ao número de passos executados no conjunto de teste.

Nos casos em que $m = 0.2n$ os resultados do algoritmo PISF degeneram completamente, com as políticas resultantes sendo incapazes de equilibrar o bastão a partir de um único estado do conjunto de teste. Por outro lado, quando $m \geq 0.6n$ o desempenho do algoritmo PISF-$S(n,m)$ é *melhor* do que o do algoritmo PI(n) original, para todos os valores de n. Em alguns casos essa melhoria é significativa, como por exemplo o aumento no número médio de passos de 1275.88 para 2889.74 quando se compara o algoritmo

PISF-S(625,500) com o PI(625). Embora tenha reduzido o tamanho dos MDPs, a fatoração estocástica com $m \geq 0.6n$ elevou a taxa de sucesso da discretização para um patamar acima de 75%. Isso significa que *todas* as políticas de decisão geradas pelo algoritmo PISF-S foram capazes de equilibrar o bastão em pelo menos 60 dos 81 estados do conjunto de teste. Em especial, quando $n = 1296$ e $m = 1037$ esse algoritmo atingiu o *melhor desempenho possível*, não deixando o bastão cair uma única vez. Não custa lembrar que alguns dos estados do conjunto de teste representam situações-limite, em que o bastão encontra-se em uma inclinação razoável e com aceleração no sentido do ângulo de inclinação. Nesses casos um pequeno erro pode levar à queda do bastão.

Como a avaliação de uma política é $O(|S|^3)$, uma redução no número de estados do problema resulta em uma economia de tempo de computação também dessa ordem. Por exemplo, uma redução de 60% no tamanho do MDP equivale a uma economia de cerca de 93.6% no número de operações aritméticas executadas a cada avaliação. Não se pode esquecer, entretanto, que a redução do modelo em si consome um certo número de operações—é daí que surge o problema do crescimento superficial, discutido na Seção 3.3.4. É razoável portanto questionar a respeito do ganho efetivo de processamento quando se leva em conta todas as etapas do processo de redução do MDP. Note que essa análise depende das características particulares dos algoritmos em questão, como o número de iterações executadas e a técnica adotada para a fatoração. No caso específico estudado aqui, é possível fazer uma estimativa baseada no fato de que tanto o algoritmo PI como o algoritmo PISF foram executados por 15 iterações. A Figura 4.11 mostra o número aproximado de operações executadas pelos algoritmos PI(1296) e PISF-S(1296,778), que apresentaram desempenhos semelhantes no problema de equilibrar um bastão.

O número total de operações executadas pelos algoritmos corresponde à soma das operações consumidas na definição do modelo com aquelas utilizadas na sua solução. Como mostra a Figura 4.11b, tanto no algoritmo PI quanto no PISF o primeiro número é desprezível em relação ao segundo (veja a Figura 4.12b para entender melhor a ilustração). A Figura 4.11 evidencia também a estratégia utilizada pelo algoritmo PISF: gastar mais tempo na definição do MDP para obter uma economia posterior na determinação da política de decisão. No caso mostrado na figura essa estratégia gera uma redução em



(a) Definição do MDP (b) Total (definição + solução do MDP)

Figura 4.11: Número aproximado de operações aritméticas executadas pelos algoritmos PI(1296) e PISF-S(1296,778) no problema de equilibrar um bastão. Os valores foram calculados com base no custo computacional de cada fase dos algoritmos.

torno de 60% no tempo total de processamento. Note que a política encontrada pelo algoritmo PISF-S(1296,778) apresenta um desempenho *superior* ao daquela retornada pelo algoritmo PI(1296). Em algumas situações reais pode ser desejável sacrificar um pouco a qualidade da política encontrada para obter uma economia ainda maior no tempo de computação.

Discretização baseada em *kernels*

Nesta seção descrevo a experiência de aplicar os algoritmos KBRL e KBSF ao problema de equilibrar um bastão. Note que a situação descrita acima, em que há um conjunto de transições descrevendo a dinâmica de um sistema, é exatamente o cenário para

que foi projetado o algoritmo KBRL. No entanto, ao tentar usar esse algoritmo depareime com um obstáculo: o número de transições na amostra de dados do problema daria origem a um MDP grande demais para ser resolvido com os recursos computacionais disponíveis. Além disso, um modelo com 94787 estados parece um imenso desperdício quando se sabe que o problema pode ser resolvido de maneira exata com apenas 1037. Foi necessário, portanto, estabelecer uma estratégia para selecionar um subconjunto das transições do problema a serem usadas na aproximação. O algoritmo k-medoids foi adotado para clusterizar os estados iniciais das transições.[8] A idéia era encontrar os estados mais representativos da amostra de dados. O único cuidado que teve que ser tomado diz respeito à inclusão de transições terminais, em que o ângulo formado pelo bastão superou os 36^o permitidos, porque essas são justamente as transições em que a recompensa não é nula—e portanto contêm informação valiosa. Assim, 10% das transições usadas pelo KBRL foram selecionadas de maneira uniforme entre as 1000 em que $r \neq 0$, e as demais foram determinadas pelo algoritmo k-medoids. A Tabela 4.6 mostra os resultados obtidos pelo algoritmo KBRL utilizando um *kernel* gaussiano cuja largura foi definida com $\varpi_a = 0.3$. Os valores referem-se ao desempenho das políticas encontradas em 15 iterações do algoritmo de iteração de política sobre o MDP definido pelo KBRL.

Algoritmo	**Ep.**(%)	**Média**	**Máx.**	**Mín.**	**DP**	**Exec.**(%)
KBRL(600)	26.36	1069.61	3000	13	1243.83	5
KBRL(900)	62.28	1931.43	3000	13	1380.41	10
KBRL(1200)	74.57	2360.12	3000	17	1145.55	30
KBRL(1500)	55.19	1866.84	3000	26	1319.84	5

Tabela 4.6: Resultados obtidos pelo algoritmo KBRL(n) no problema de equilibrar um bastão. O parâmetro n se refere ao número de transições usadas na aproximação. Os valores correspondem ao número médio de passos executados no conjunto de teste pelas políticas de decisão retornadas em 20 execuções independentes do algoritmo. A coluna "Exec." indica a porcentagem das execuções 100% bem-sucedidas, ou seja, a proporção de execuções em que o bastão foi equilibrado por 3000 passos a partir de todos os estados do conjunto de teste.

Como o resultado do algoritmo k-medoids pode variar a cada execução, os experi-

[8] Mais especificamente, foi utilizado o algoritmo CLARA, que é uma variante mais rápida do algoritmo k-medoids [72].

mentos com o KBRL(n) foram repetidos 20 vezes para cada valor de n. Note pelo alto desvio-padrão dos resultados mostrados na Tabela 4.6 como a qualidade da política retornada pelo KBRL varia bastante, dependendo dos dados selecionados para a aproximação. Observe também que a taxa média de sucesso desse algoritmo é relativamente baixa. Os melhores resultados foram obtidos com $n = 1200$, possivelmente porque o valor de ϖ_a usado para definir a largura dos *kernels* é mais compatível com essa quantidade de dados. Mesmo nesse caso a proporção de episódios bem-sucedidos é de apenas 74.57%, um resultado ruim quando se considera que é possível atingir uma taxa de 100% utilizando um modelo com aproximadamente a mesma dimensão (Tabela 4.5). A coluna "Exec." da Tabela 4.6 mostra a proporção de execuções bem-sucedidas do algoritmo KBRL. Uma execução é considerada um sucesso quando a política resultante é capaz de equilibrar o bastão a partir de todos os estados do conjunto de teste. Note que segundo esse critério o desempenho do KBRL é ainda pior, com uma expectativa de sucesso de apenas 5% quando $n = 1500$.

Por que os resultados do algoritmo KBRL são tão inferiores aos encontrados anteriormente com aproximadamente os mesmos recursos? A resposta para esta pergunta é simples: porque, efetivamente falando, esse algoritmo *não* está utilizando os mesmos recursos dos algoritmos descritos na seção anterior. Observe que, em contraste com o que ocorre com o algoritmo PI(n), no algoritmo KBRL(n) o parâmetro n é de fato uma medida da quantidade de informação incorporada ao modelo. A política encontrada pelo algoritmo PI(1296), por exemplo, utilizou *todas* as transições da amostra de dados, embora tenha sido derivada de um MDP com apenas 1296 estados. Em contraste, as políticas encontradas pelo algoritmo KBRL(1200) usaram apenas 1200 transições, ou aproximadamente 1.27% da informação utilizada pelo algoritmo anterior. Sob esta perspectiva, os resultados do KBRL não são tão ruins assim.

Obviamente, a qualidade das políticas retornadas pelo algoritmo KBRL pode ser melhorada aumentando-se o número n de transições usadas na aproximação. Infelizmente, no caso desse algoritmo isso significa um aumento considerável do custo computacional do processo. Isso é uma conseqüência direta de uma questão já discutida anteriormente: como no algoritmo KBRL o mesmo conjunto de transições é utilizado para definir a estrutura e a dinâmica do MDP, um aumento do poder descritivo deste último vem neces-

sariamente acompanhado da sua complexificação. A fatoração estocástica baseada em *kernels*, apresentada na Seção 4.3.2, é uma proposta para resolver essa questão. Segue uma descrição da sua aplicação ao problema de equilibrar um bastão.

Em contraste com o que acontece no algoritmo KBRL, no algoritmo KBSF há total liberdade para definir os arquétipos usados na aproximação. Para que uma comparação mais direta dos dois métodos fosse possível, no entanto, foi utilizada aqui a mesma estratégia do caso anterior—ou seja, os arquétipos foram determinados pelo algoritmo k-medoids. As demais configurações usadas com o KBRL também foram mantidas. A única diferença no caso do algoritmo KBSF é a determinação de um *kernel* extra, como discutido na Seção 4.3.2. Mais uma vez foi adotada uma função-núcleo gaussiana, e a largura do *kernel* resultante foi definida com $\varpi_q = 0.01$. Os resultados do algoritmo KBSF podem ser vistos na Tabela 4.7.

Algoritmo	**Ep.**(%)	**Média**	**Máx.**	**Mín.**	**DP**	**Exec.**(%)
KBSF(94787, 25)	92.59	2779.15	3000	17	781.07	0
KBSF(94787, 50)	99.20	2976.09	3000	21	265.87	35
KBSF(94787, 75)	100.00	3000.00	3000	3000	0.00	100
KBSF(94787, 100)	100.00	3000.00	3000	3000	0.00	100

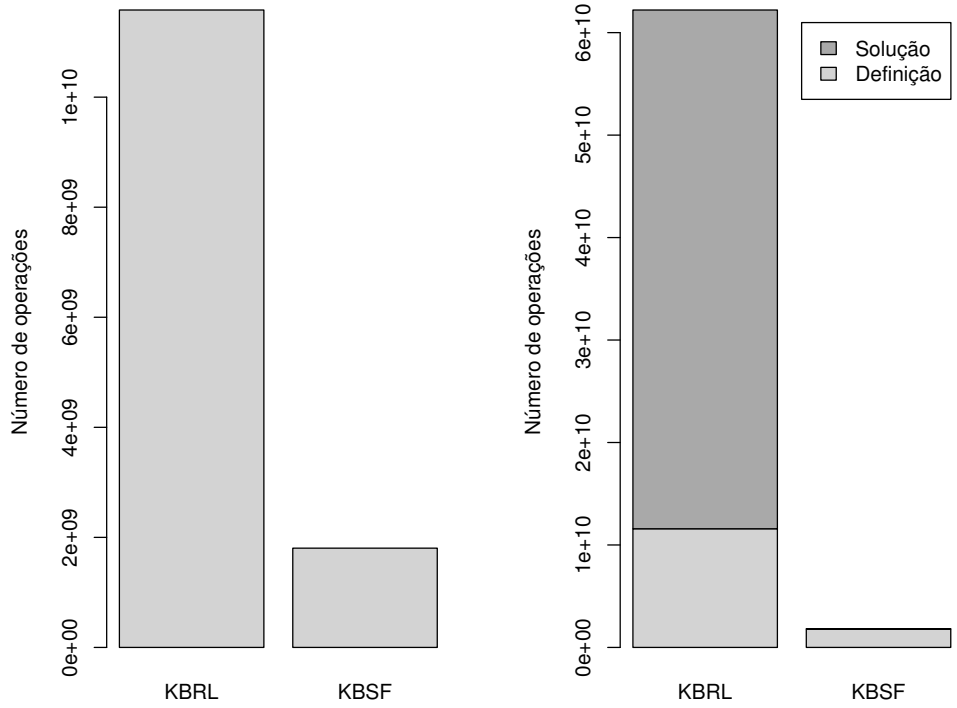
Tabela 4.7: Resultados obtidos pelo algoritmo KBSF(n,m) no problema de equilibrar um bastão. Os parâmetros n e m se referem respectivamente ao número de transições e arquétipos usados na aproximação. Os valores correspondem ao número médio de passos executados no conjunto de teste pelas políticas de decisão retornadas em 20 execuções independentes do algoritmo.

Observe como a possibilidade de incorporar informação a um modelo de tamanho fixo permite ao KBSF gerar políticas de decisão sensivelmente superiores àquelas encontradas pelo seu antecessor. A discrepância entre os resultados chega a ser impressionante: o desempenho do algoritmo KBSF(94787,25), por exemplo, é melhor do que o do algoritmo KBRL(1500), mesmo usando um MDP 60 vezes menor. Observe também como a taxa de sucesso do algoritmo KBSF(n,m) cresce rapidamente com o aumento de m, atingindo o *melhor desempenho possível* quando $m = 75$ apenas. Não custa lembrar que nesse caso as políticas geradas pelo algoritmo KBSF foram capazes de atingir o objetivo a partir dos 81 estados do conjunto de teste em todas as 20 execuções, totalizando 1620 tentativas

bem-sucedidas de equilibrar o bastão. A julgar por este experimento, a expectativa de sucesso do algoritmo KBSF(94787,m) com $m \geq 75$ é algo bem próximo de 100%.

Como ambos se baseiam na fatoração estocástica, a filosofia do algoritmo KBSF é a mesma do algoritmo PISF: investir um grande número de operações aritméticas na definição do MDP para posteriormente economizar um número ainda maior de operações na sua solução. Em relação ao algoritmo KBRL, o custo extra na definição do modelo por parte do KBSF se refere basicamente à multiplicação das matrizes $\mathbf{D}^a$ e $\mathbf{K}^a$, como mostra o Algoritmo 4.2. No caso estudado aqui, no entanto, esse custo é muito pequeno em relação ao custo computacional do algoritmo k-medoids. Como o número de *clusters* usados pelo KBSF é muito menor do que o usado pelo KBRL, o primeiro algoritmo é mais rápido mesmo na definição do MDP, como ilustra a Figura 4.12a. Quando se trata da solução do processo de decisão de Markov, a diferença no tempo de computação dos dois algoritmos é ainda maior. No caso mostrado na Figura 4.12, a cada operação executada pelo algoritmo KBSF na solução do MDP o algoritmo KBRL executou aproximadamente 10000 operações. A economia no tempo total de processamento foi algo em torno de 97%.

A Figura 4.12b deixa claro que os métodos baseados em *kernels* gastam um número de operações na definição do MDP que representa uma fatia significativa do tempo total de computação. Note, no entanto, que aqui o inchaço desse número é uma conseqüência da estratégia adotada para definir as transições usadas na aproximação. Se técnicas mais simples do que o agrupamento de dados fossem usadas para determinar a estrutura do aproximador, provavelmente se veria um padrão mais próximo daquele visto na Figura 4.11. Nesse caso o custo extra requerido pelo KBSF poderia torná-lo mais lento do que o KBRL na definição do MDP. É interessante notar, no entanto, que a liberdade na definição dos arquétipos oferecida pelo KBSF pode ser explorada para tornar a construção do MDP muito eficiente. Pode-se, por exemplo, usar arquétipos distribuídos de forma regular no espaço de estados, como feito na Seção 4.3.3. Nesse caso, o tempo de computação gasto com a definição do modelo tornaria-se desprezível em relação àquele gasto com a sua solução. De fato, essa alternativa foi tentada, com resultados animadores: com apenas 256 arquétipos distribuídos regularmente sobre o espaço X, a política de decisão encontrada pelo algoritmo KBSF foi capaz de equilibrar o bastão a partir de 80 dos 81 estados do conjunto de teste.



(a) Definição do MDP (b) Total (definição + solução do MDP)

Figura 4.12: Número aproximado de operações aritméticas executadas pelos algoritmos KBRL(1500) e KBSF($94787{,}75$) no problema de equilibrar um bastão. Valores calculados considerando 20 iterações do algoritmo k-medoids.

O leitor atento terá notado que evitei fazer comparações diretas entre os métodos baseados em *kernels*, KBRL e KBSF, e aqueles estudados na última seção (PI e PISF). Isso porque esse tipo de análise é muito influenciado por peculiaridades que representam mais decisões de projeto do que características inerentes aos métodos. Um ponto que ficou evidente nos experimentos é a superioridade dos algoritmos baseados na fatoração estocástica sobre os demais, tanto em relação ao tempo de computação quanto em relação à qualidade das soluções encontradas. Quando se compara os algoritmos PISF e KBSF diretamente, os experimentos desta seção e aqueles das seções 4.2.4 e 4.3.3 parecem indicar uma pequena vantagem em favor do segundo. Uma possível explicação para isso é o fato do algoritmo KBSF trabalhar sobre uma discretização suave do espaço de estados, em contraste com o algoritmo PISF, que opera sobre uma discretização convencional.

4.4.2 Equilibrando dois bastões simultaneamente

A tarefa estudada na seção anterior torna-se consideravelmente mais difícil quando se anexa um segundo bastão ao carro, agora com $1/10$ do comprimento do bastão original (veja detalhes no Apêndice B). Como o bastão menor ganha velocidade mais facilmente do que o maior, nessa configuração do problema o controlador deve ser capaz de fazer manobras contra-intuitivas, como por exemplo aumentar a velocidade angular do segundo para compensar a inclinação do primeiro no sentido oposto [184]. Só para que se tenha uma idéia da dificuldade dessa versão do problema, em uma série de experimentos realizados recentemente por Gomez *et al.* [51], apenas um algoritmo de aprendizagem por reforço foi capaz de executar a tarefa, e mesmo assim de maneira insatisfatória. O problema de equilibrar dois bastões é usado nesta seção para comparar o desempenho do algoritmo KBSF com o do *algoritmo de iteração de política baseado nos mínimos quadrados*, daqui pra frente referenciado simplesmente como LSPI[†] [82].

O LSPI é uma iteração de política aproximada que deve a sua origem ao *algoritmo das diferenças temporais baseado nos mínimos quadrados* (LSTD[†]) [28]. Como os nomes sugerem, a idéia dos dois algoritmos é encontrar uma aproximação da função de valor que minimize a soma de uma medida quadrática de erro. Em ambos os casos a avaliação de uma política fica reduzida à solução de um problema dos mínimos quadrados linear. A diferença entre os dois é que o algoritmo LSTD aproxima a função de valor V^π, e portanto depende de um modelo completo para ser incorporado ao esquema da programação dinâmica. O LSPI, por outro lado, produz uma aproximação da função de valor de ação Q^π—o que permite a definição de um poderoso algoritmo de iteração de política.

Não seria exagero dizer que o LSPI constitui uma das escolhas mais sensatas para lidar com problemas de aprendizagem por reforço na atualidade. Em primeiro lugar, esse algoritmo faz um uso muito eficiente dos dados, que podem ser coletados a partir de qualquer distribuição razoável [82]. Além disso, o LSPI não depende da definição de nenhum parâmetro; a única decisão de projeto se refere à determinação da estrutura do aproximador. Como se não bastasse, o algoritmo LSPI oferece garantias teóricas de

[†] *Least squares policy iteration*

[†] *Least-squares temporal-difference learning*

desempenho [82] e já foi bastante testado na prática [83]. Os detalhes do funcionamento do algoritmo LSPI fogem ao escopo deste trabalho. O leitor interessado deve se dirigir ao artigo original de Lagoudakis e Parr [82], que descreve o algoritmo com extrema clareza e é bastante auto-contido.

Nos experimentos relatados nesta seção a tarefa dos algoritmos LSPI e KBSF era equilibrar os dois bastões a partir de um conjunto de teste formado por 81 estados distribuídos de forma regular em

$$\begin{array}{ll} x \in [-0.72, 0.72]\mathrm{m} & \dot{x} \in [-0.72, 0.72]\mathrm{m/s} \\ \theta \in [-3\pi/50, 3\pi/50]\,\mathrm{rad} & \dot{\theta} \in [-3\pi/50, 3\pi/50]\,\mathrm{rad/s} \\ \theta_2 = 0\,\mathrm{rad} & \dot{\theta}_2 = 0\,\mathrm{rad/s}. \end{array} \tag{4.23}$$

Um episódio foi considerado bem-sucedido quando os bastões puderam ser equilibrados por 3000 passos. Note que os limites para os intervalos de x e θ correspondem a 30% dos valores extremos dessas variáveis, em contraste com a seção anterior, em que esses intervalos representavam 75% dos valores factíveis. Essa redução da região de inicialização foi necessária para compensar o grau de dificuldade acrescentado ao problema. Observe também que o bastão mais curto foi inicializado sempre na posição vertical com aceleração angular nula. Isso porque uma pequena variação do ângulo θ_2 pode colocar o sistema em um estado irrecuperável. Para se ter uma idéia, mesmo sendo iniciado na posição de repouso o bastão mais curto foi o responsável pelo término da grande maioria dos episódios, como será discutido à frente.

Assim como o KBSF, o algoritmo LSPI opera sobre um conjunto de transições do tipo (x_i, a_i, r_i, y_i). Portanto, os experimentos desta seção foram configurados como os da seção anterior, com uma política de exploração π_e coletando os dados usados nas aproximações. Foram usadas transições coletadas a partir de 1000 tentativas de equilibrar os bastões. Os estados iniciais de cada tentativa foram amostrados de uma distribuição uniforme sobre o conjunto definido em (4.23). Os detalhes da amostra de dados usada no experimento podem ser vistos na Tabela 4.8. Comparando os dados desta tabela com aqueles mostrados na Tabela 4.3, percebe-se uma queda significativa do número médio de transições executadas pela política π_e, de 94.79 para 17.32. Isso dá uma idéia da dificuldade acrescentada ao problema com a inclusão de um novo bastão. Um outro dado importante que fica evidente na Tabela 4.8 é a exclusão da ação $a = 0\mathrm{N}$ do problema.

Além de simplificar a tarefa, essa modificação reduz o tempo de processamento do algoritmo LSPI, cujo custo computacional cresce rapidamente com $|A|$. O custo do LSPI será discuto em detalhes à frente.

Número de episódios	1000
Número de transições	17319
Número médio de transições por episódio	17.32
Número máximo de transições por episódio	106
Número mínimo de transições por episódio	5
Desvio-padrão do número de transições por episódio	12.19
Transições em que a ação $-10N$ foi executada	50.07%
Transições em que a ação $+10N$ foi executada	49.93%

Tabela 4.8: Informações sobre a amostra de transições usada no problema de equilibrar dois bastões simultaneamente.

Os experimentos desta seção foram realizados da seguinte maneira. Primeiramente, os dados foram agrupados pelo algoritmo k-means, de forma a determinar as coordenadas de m arquétipos. Esses arquétipos foram usados como os centros das funções gaussianas utilizadas na aproximação. Os mesmos aproximadores foram usados pelos algoritmos LSPI e KBSF para aproximar a função de valor (veja a equação (4.21)). No caso deste último, a largura dos *kernels* foi definida como descrito na Seção 4.3.3, com $\varpi_a = 0.3$, $\varpi_q = 0.01$ e 7 vizinhos usados no cálculo de τ_a e τ_q. Não foram tentadas outras combinações de valores para esses parâmetros. A definição da largura do *kernel* usado pelo algoritmo LSPI exigiu um pouco mais de cuidado. Foram testados valores de ϖ_q no conjunto

$$\varpi_q \in \{0.1, 0.2, ..., 0.9\} \cup \{0.01, 0.95, 0.99\}. \tag{4.24}$$

Para cada valor de ϖ_q no conjunto acima o experimento com o algoritmo LSPI foi repetido 5 vezes (o Apêndice C traz os resultados dessa bateria preliminar de experimentos). Os valores 0.95 e 0.99 foram acrescentados à (4.24) a partir da constatação de que valores mais altos para ϖ_q geravam políticas de decisão de melhor qualidade. De fato, as melhores médias obtidas pelo algoritmo LSPI correspondem ao caso em que $\varpi_q = 0.95$. Esses são os resultados relatados aqui.

A Tabela 4.9 mostra os resultados obtidos pelos algoritmos LSPI e KBSF no problema

de equilibrar dois bastões simultaneamente. Esses resultados representam o desempenho das políticas de decisão encontradas após 15 iterações do algoritmo de iteração de política. Note que, apesar do esforço investido na definição dos seus aproximadores, o LSPI gerou resultados no mínimo insatisfatórios. Na sua versão mais bem-sucedida, quando $m = 150$, esse algoritmo foi capaz de equilibrar os dois bastões em apenas 146 das 1620 tentativas de fazê-lo (20 execuções $\times$ 81 estados no conjunto de teste). No entanto, o que torna os resultados do LSPI particularmente preocupantes é o fato de o desempenho desse algoritmo não melhorar com o aumento do número m de arquétipos usados na aproximação. Se esse fosse o caso, o problema se reduziria a uma questão computacional: dados recursos de armazenamento e processamento suficientes, as políticas de decisão retornadas pelo LSPI atingiriam eventualmente o nível de qualidade desejado. Infelizmente, a julgar pela Tabela 4.9, um aumento de m não é garantia de melhoria do desempenho do algoritmo LSPI.

Algoritmo	**Ep.**(%)	**Média**	**Máx.**	**Mín.**	**DP**	**Exec.**(%)
LSPI(17319,50)	0.86	168.91	3000	5	371.82	0
LSPI(17319,100)	4.44	296.93	3000	5	646.01	0
LSPI(17319,150)	9.01	396.77	3000	6	841.03	5
LSPI(17319,200)	0.00	71.01	1075	5	79.91	0
LSPI(17319,250)	1.11	120.59	3000	5	359.44	0
KBSF(17319,50)	20.99	996.29	3000	5	1149.71	0
KBSF(17319,100)	42.47	1568.57	3000	5	1321.06	0
KBSF(17319,150)	61.98	2128.85	3000	6	1246.90	0
KBSF(17319,200)	71.23	2348.42	3000	5	1140.52	0
KBSF(17319,250)	75.06	2381.21	3000	6	1161.26	0

Tabela 4.9: Resultados obtidos pelos algoritmos LSPI(n,m) e KBSF(n,m) no problema de equilibrar simultaneamente dois bastões. Os parâmetros n e m referem-se respectivamente ao número de transições e arquétipos usados na aproximação. Os valores correspondem ao número médio de passos executados no conjunto de teste pelas políticas de decisão retornadas em 20 execuções independentes dos algoritmos.

Em contraste, os resultados do algoritmo KBSF melhoram monotonicamente com o aumento do número de arquétipos usados da aproximação. Note que isso é verdade não apenas em relação ao número de episódios bem-sucedidos, como também em relação

ao número médio de passos executados por episódio. Esse comportamento previsível é extremamente bem-vindo, pelas razões discutidas acima. Note que, no caso em que $m = 250$, o algoritmo KBSF foi capaz de equilibrar os bastões em 1216 das 1620 tentativas realizadas, uma proeza considerável dada a dificuldade do problema.

Apesar do bom desempenho do algoritmo KBSF na tarefa de equilibrar simultaneamente dois bastões, pode-se destacar um ponto negativo em relação aos valores mostrados na Tabela 4.9. Observe como o desvio-padrão dos resultados desse algoritmo não exibe uma tendência de queda com o aumento de m, como era de se esperar. Em um primeiro momento pode-se interpretar esse fato como um indício de instabilidade do algoritmo, mas uma análise mais cuidadosa mostra que esse não é o caso. Note que a variação dos resultados mostrados na Tabela 4.9 tem duas causas possíveis. Uma delas é a mudança no desempenho do algoritmo KBSF a cada execução, em decorrência da diferença no posicionamento dos arquétipos definido pelo algoritmo k-means. A outra causa possível para o alto desvio-padrão dos resultados é a heterogeneidade do conjunto de teste, que contém estados com diferentes níveis de dificuldade de se equilibrar os bastões.

A Figura 4.13 mostra uma comparação entre os dois tipos de desvio-padrão presentes nos resultados da Tabela 4.9 referentes ao algoritmo KBSF. O desvio "inter-execução" se refere ao desvio-padrão da seqüência de comprimento 20 em que cada elemento é a média de passos executados pelo KBSF em todo o conjunto de teste. Ou seja, cada elemento dessa seqüência sintetiza o desempenho do algoritmo KBSF em uma execução. O desvio "intra-execução," por outro lado, é o desvio-padrão da seqüência de comprimento 81 em que cada elemento é a média obtida pelo algoritmo KBSF em um estado específico do conjunto de teste. Uma maneira simples de entender a diferença entre os dois tipos de desvio-padrão é imaginar os resultados organizados de forma tabular em uma matriz 20×81, em que cada linha corresponde a uma execução do algoritmo e cada coluna representa um estado do conjunto de teste. Nesse caso, a primeira medida seria o desvio-padrão dos valores médios por coluna e a segunda seria a mesma estatística calculada em relação às linhas da matriz.

Observe na Figura 4.13 como o algoritmo KBSF se comporta exatamente como esperado: quando o número de arquétipos m é pequeno, a aproximação realizada por esse

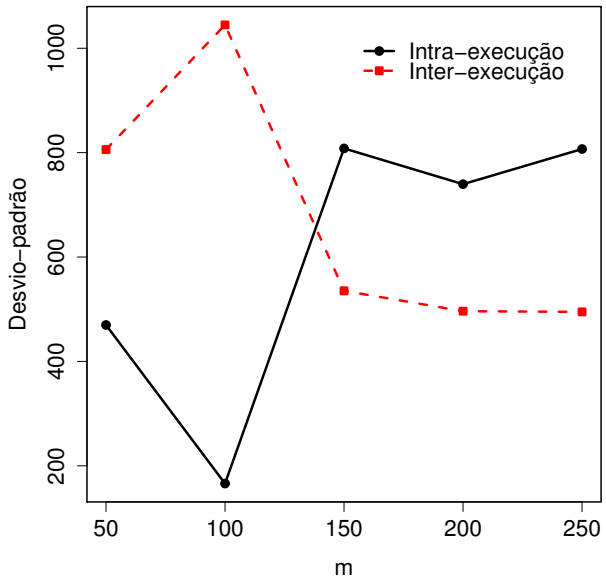


Figura 4.13: Desvio-padrão dos resultados do algoritmo KBSF no problema de equilibrar simultaneamente dois bastões. Os valores se referem aos resultados mostrados na Tabela 4.9. O desvio-padrão "intra-execução" corresponde à variação do número de passos executados pelo algoritmo no conjunto de teste. O desvio "inter-execução" é uma medida da variação decorrente do posicionamento dos arquétipos em cada execução.

algoritmo é mais sensível ao posicionamento deles, o que resulta em uma grande variação dos resultados de uma execução para a outra. Além disso, como as políticas de decisão encontradas são incapazes de equilibrar os bastões por muito tempo, o desvio intra-execução é relativamente pequeno. A expectativa em relação a um algoritmo bem-comportado é que um aumento no número de arquétipos tenha efeitos contrários sobre os dois fenômenos acima. Quando se observa a Figura 4.13, nota-se que isso é justamente o que ocorre com o algoritmo KBSF, como indica a inversão das curvas representando os dois tipos de desvio-padrão. Isso explica porque a variação dos resultados mostrados na Tabela 4.9 permanece alta mesmo com o aumento de m. Note que este raciocínio só faz sentido quando se considera um número fixo de transições usadas na aproximação. Como é possível equilibrar o bastão a partir de todos os estados do conjunto de teste, a expectativa é que ambos os tipos de desvio-padrão desapareçam quando $n \to \infty$. Esse assunto será retomado em breve.

Antes, porém, é interessante analisar com mais cuidado os resultados obtidos pelos

algoritmos LSPI e KBSF no problema de equilibrar dois bastões ao mesmo tempo. Como mostra a Tabela 4.9, o algoritmo KBSF supera de maneira significativa o LSPI nessa tarefa. A diferença no desempenho dos dois algoritmos torna-se ainda mais expressiva quando se leva em consideração o custo computacional de cada um. A avaliação de política no algoritmo LSPI envolve a atualização e a solução de um sistema linear, que requerem respectivamente $O(nm^2)$ e $O(m^3|A|^3)$ operações aritméticas.[9] Isso sem contar a melhoria de política, que exige a determinação da ação ótima *em cada estado* y_i^a *da amostra de transições*. O algoritmo KBSF, em comparação, executa na avaliação de política um número de operações na ordem de m^3, enquanto a melhoria de política é $O(m^2|A|)$ apenas. Fica claro, portanto, que quando se usa os mesmos aproximadores e as mesma amostras de transições para ambos os algoritmos, como nos experimentos desta seção, o número de operações realizadas pelo algoritmo KBSF representa apenas uma pequena fração das operações executadas pelo LSPI. Para que se tenha uma idéia da magnitude da diferença no tempo de processamento dos algoritmos, quando $n = 17319$, $m = 100$ e $|A| = 2$, como em um dos experimentos desta seção, o número de operações realizadas em uma iteração do KBSF representa menos de 1% das operações envolvidas em uma iteração do LSPI.

Pode-se argumentar que o desempenho ruim do LSPI é conseqüência das decisões de projeto tomadas aqui. De fato, existem muitas possibilidades para tentar melhorar os resultados desse algoritmo. É possível que valores para ϖ_q fora do conjunto (4.24) gerem políticas de decisão de melhor qualidade. Pode-se pensar também em usar uma largura específica para cada função de base radial. Indo um pouco mais além, é razoável supor que outras escolhas de função-núcleo—que no caso do LSPI não precisa ser local—resultem em um desempenho melhor do algoritmo. Há de se admitir, no entanto, que um esforço razoável foi investido aqui para encontrar bons resultados com o algoritmo LSPI. Além dos diversos valores usados para o parâmetro ϖ_q, foram testados outros mecanismos de coleta de dados, como será discutido a seguir. Infelizmente, no problema de equilibrar dois bastões nenhuma dessas tentativas resultou em uma melhoria significativa do desem-

[9]Foi adotada a implementação do LSPI em que o mesmo bloco de funções radiais é repetido para cada ação do problema, como sugerido por Lagoudakis e Parr [82]. O custo de $O(nm^2)$ para atualizar o sistema representa uma otimização em relação à versão do algoritmo apresentada pelos autores, na qual esse processo requer um número de operações na ordem de $nm^2|A|^2$.

penho mostrado na Tabela 4.9.

Finalmente, é importante destacar que os resultados obtidos pelo KBSF não devem ser interpretados como o melhor desempenho possível desse algoritmo, uma vez que não houve um ajuste dos seus parâmetros para cada problema estudado. A decisão de usar os mesmos valores para ϖ_a e ϖ_q em todos os experimentos tem como objetivo mostrar a robustez do KBSF, além de evitar uma super-especialização do algoritmo a cada problema—o que causaria uma expectativa distorcida em relação a um cenário real. Não é absurdo supor, no entanto, que em algumas aplicações reais seja possível fazer um pequeno ajuste preliminar de parâmetros, e nesse caso os resultados desta seção seriam uma previsão conservadora do desempenho alcançável com o algoritmo KBSF.

A questão da amostragem de dados

É tranqüilizador saber que o desempenho do algoritmo KBSF(n,m) melhora à medida que $m \to n$. No entanto, para que esse algoritmo se caracterize de fato como um método bem-comportado, é necessário que o mesmo ocorra quando se aumenta o número n de transições usadas na aproximação. Para verificar se isso realmente acontece, repeti o experimento da seção anterior, agora mantendo o número de arquétipos fixo em $m = 100$ e variando o número de transições na amostra de dados. As transições extras foram geradas como descrito na seção anterior e sucessivamente acrescentadas à amostra original. A Figura 4.14 mostra os resultados encontrados nesse experimento. Observe como os comportamentos dos algoritmos LSPI e KBSF seguem o mesmo padrão. Após uma melhoria inicial quando o número de episódios usados na aproximação cresce de 1000 para 2000, o desempenho de ambos começa a degradar com o acréscimo posterior de transições. Embora a taxa de degradação dos resultados seja relativamente lenta, esse fenômeno é preocupante, especialmente quando se considera que o movimento contrário seria o esperado.

O fato de os algoritmos LSPI e KBSF se comportarem de maneira tão parecida sugere que o problema pode estar na estratégia usada para a coleta de informação. Lembre-se que nos experimentos descritos acima os dados usados na aproximação foram obtidos por uma política de exploração. O grande atrativo dessa estratégia de amostragem é a sua

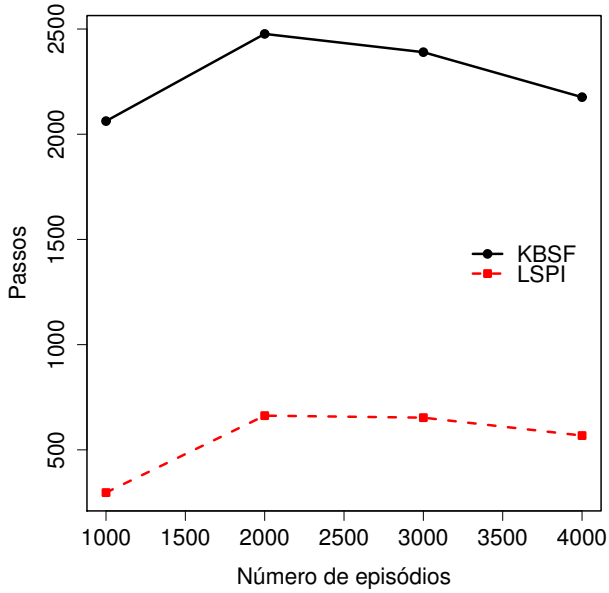


Figura 4.14: Resultados obtidos pelos algoritmos LSPI e KBSF no problema de equilibrar simultaneamente dois bastões utilizando amostras de dados de diferentes tamanhos. Cada amostra de dados corresponde à amostra anterior acrescida das transições coletadas em 1000 episódios pela política de exploração π_e. Os resultados são uma média de 20 execuções dos algoritmos utilizando $m = 100$ arquétipos na aproximação.

generalidade, mas ela também apresenta algumas desvantagens. Note que mesmo que a política de exploração π_e seja estritamente "justa," escolhendo cada ação do problema com a mesma probabilidade, a distribuição dos estados resultantes da sua interação com o ambiente será em geral bastante tendenciosa. Quando se tenta equilibrar dois bastões simultaneamente, por exemplo, os episódios tendem a terminar com a queda do bastão mais curto, uma vez que as forças exercidas sobre o carro têm um efeito muito maior sobre ele do que sobre o outro bastão. Isso concentra os dados em uma região específica do espaço de estados, deixando outras regiões completamente inexploradas. Para que se tenha uma idéia da gravidade do problema, na amostra de dados detalhada na Tabela 4.8 apenas *uma* das 1000 tentativas de equilibrar os bastões não foi encerrada pela queda do bastão mais curto—quando o carro atingiu o limite esquerdo da pista. É correto afirmar, portanto, que os resultados mostrados na Tabela 4.9 foram obtidos por políticas de decisão que simplesmente *ignoravam* o bastão mais longo.

Diante disso, é natural a pergunta se os resultados dos algoritmos LSPI e KBSF po-

deriam ser melhorados utilizando-se outras estratégias de amostragem de dados. De fato, um dos pressupostos de Ormoneit e Sen [111] em relação ao algoritmo KBRL é que as transições usadas na aproximação sejam amostradas de maneira uniforme no espaço de estados.[10] Para verificar se essa estratégia resultaria em políticas de decisão de melhor qualidade, reproduzi o experimento acima utilizando conjuntos de dados gerados da seguinte maneira: primeiramente, os estados iniciais das transições foram amostrados de uma distribuição uniforme definida no espaço de estados X. Como as acelerações do carro e dos bastões não são naturalmente limitadas, utilizei como limite para esses intervalos os valores extremos que apareceram na amostra anterior (Tabela 4.8). Uma vez definidos os estados iniciais, uma das duas ações do problema foi sorteada para ser executada em cada um deles. Cada dupla (x_i, a_i) foi armazenada juntamente com a recompensa r_i e o estado y_i resultantes da execução de a_i em s_i. Assim como no caso anterior, as diferentes amostras foram geradas com o acréscimo sucessivo de novas transições à amostra original.

Note que com essa nova estratégia de amostragem os estados iniciais das transições tendem a estar mais "espalhados" pelo espaço X, o que resulta em um aumento exponencial do volume do envoltório convexo definido pelos dados [60]. O efeito desse fenômeno sobre o algoritmo KBSF é uma diminuição excessiva da sobreposição das funções gaussianas, o que degrada completamente os seus resultados. Utilizando $\varpi_q = 0.01$, como nos experimentos anteriores, esse algoritmo foi incapaz de encontrar uma política que pudesse equilibrar os bastões por mais de 50 passos. Para compensar esse efeito, a largura do *kernel* κ^q usado pelo KBSF foi redefinida utilizando-se a mesma estratégia adotada na seção anterior para determinar a largura do *kernel* correspondente do algoritmo LSPI. Ficou constatado a partir de uma série preliminar de experimentos que a largura que gerava os melhores resultados era aquela derivada de $\varpi_q = 0.6$. O parâmetro ϖ_a não foi alterado. Para ser "justo" com o algoritmo LSPI, uma segunda bateria de experimentos foi realizada para verificar se a largura usada no experimento anterior permanecia a melhor escolha com essa nova estratégia de amostragem. Nesse caso, a escolha por $\varpi_q = 0.95$ foi confirmada. Os experimentos usados para definir a largura dos *kernels* dos algoritmos

[10] Essa restrição foi posteriormente removida em um trabalho relacionado de um dos autores, em que o critério de otimalidade adotado é a recompensa média acumulada [110].

KBSF e LSPI são descritos em detalhes no Apêndice C.

Os resultados dos algoritmos KBSF e LSPI utilizando transições amostradas de maneira uniforme em X podem ser vistos na Figura 4.15. Os valores se referem ao desempenho dos algoritmos no mesmo conjunto de teste usado no experimento anterior. Observe que nesse caso o eixo das abscissas corresponde ao número de transições usadas na aproximação, uma vez que nesse esquema de amostragem a interação com o problema não se dá em episódios. Comparando os resultados dos algoritmos quando o número de transições é 20000 com aqueles mostrados na Figura 4.14 obtidos com 1000 episódios (ou 17319 transições), nota-se uma queda considerável do desempenho tanto do LSPI quanto do KBSF. Esse fenômeno é provavelmente uma conseqüência do aumento exponencial do volume do hipercubo definido pelos dados, como discutido acima. Em contraste com o experimento anterior, em que as amostras de transições estavam concentradas em uma região crítica do espaço de estados, no experimento mostrado na Figura 4.15 os dados encontravam-se espalhados por todo o espaço X. Nesse caso, é necessário um número muito maior de transições para "preencher" de maneira adequada o espaço de estados de seis dimensões. Se por um lado essa estratégia de amostragem resulta em um uso menos eficiente dos dados, por outro lado ela também resulta em um comportamento mais previsível dos algoritmos. Note como as curvas da Figura 4.15 indicam claramente uma tendência de melhora no desempenho do KBSF e do LSPI com o aumento do número de transições usadas na aproximação. Pode-se inferir que com um número suficientemente grande de transições na amostra ambos os algoritmos superariam o desempenho apresentado no experimento anterior.

Seria interessante verificar se a suspeita acima se confirma aumentando sucessivamente o número n de exemplos de transições usados nas aproximações. No entanto, os requisitos de memória e processamento desse experimento tornam a sua realização proibitiva com os recursos computacionais disponíveis. Essa questão ilustra uma tensão envolvida na escolha da estratégia de amostragem. Em geral, uma cobertura mais localizada do espaço de estados resulta em um desempenho satisfatório com um número menor de transições, mas limita assintoticamente a qualidade das políticas de decisão que podem ser encontradas.

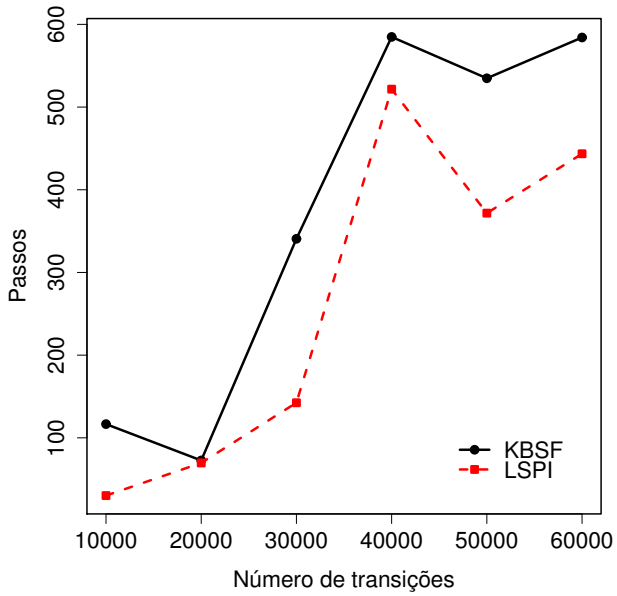


Figura 4.15: Resultados obtidos pelos algoritmos LSPI e KBSF no problema de equilibrar dois bastões utilizando transições amostradas de maneira uniforme no espaço de estados. Cada amostra de dados corresponde à amostra anterior acrescida de 1000 transições. Os resultados são uma média de 20 execuções dos algoritmos utilizando $m = 100$ arquétipos na aproximação. Para esse experimento exclusivamente a largura do *kernel* κ^q usado pelo algoritmo KBSF foi determinada com $\varpi_q = 0.6$.

A coleta de dados é uma questão antiga na aprendizagem por reforço, e constitui uma das suas principais diferenças em relação à aprendizagem supervisionada—em que a distribuição de onde são amostrados os dados é em geral definida *a priori* [147]. Existe na literatura alguma evidência de que uma boa estratégia de amostragem é coletar as transições segundo a política em avaliação [160, 170]. Essa abordagem pode a princípio ser adotada tanto com o algoritmo LSPI quanto com o algoritmo KBSF, desde que haja uma nova coleta de dados a cada política de decisão gerada. O desenvolvimento de técnicas eficientes de amostragem de dados constitui na minha opinião um dos passos mais importantes para o processo de consolidação dos algoritmos modernos de aprendizagem por reforço.

4.5 Resumo

As idéias apresentadas neste capítulo baseiam-se fortemente na distinção entre dois espaços métricos envolvidos na programação dinâmica. O primeiro deles é o espaço markoviano $M^{|S|}$, cujos elementos são vetores compostos por recompensas e probabilidades de transições. Embora em geral não seja explicitamente identificado como tal, o conjunto $M^{|S|}$ constitui o verdadeiro espaço da programação dinâmica. De fato, $M^{|S|}$ contém toda a informação relevante para esta última, já que é nele que ocorre a definição de um processo de decisão de Markov.

Se o espaço markoviano fornece a representação de um MDP, é o espaço de estados S que lhe confere uma interpretação. No espaço S cada estado s_i é uma descrição de uma situação do mundo real. Pode-se dizer portanto que é esse espaço que associa os vetores do espaço $M^{|S|}$ com o fenômeno de interesse. Note que o espaço S é uma necessidade lógica, e a premissa da sua existência não impõe nenhum tipo de restrição.

A distinção entre o espaço markoviano e o espaço de estados pode ajudar a entender as diferenças entre a programação dinâmica e a aprendizagem por reforço: enquanto a primeira se preocupa exclusivamente com o espaço $M^{|S|}$, a segunda estuda as propriedades dos estados de S que são impostas implicitamente pelos elementos do espaço markoviano. Entretanto, entender as relações entre $M^{|S|}$ e S tem também conseqüências de caráter mais prático. Como existe uma associação natural entre os elementos desses espaços, pode-se em alguns casos escolher em qual dos dois trabalhar, de acordo com a conveniência. No caso específico estudado neste trabalho, foi explorado o fato de que, em geral,

$$\dim(S) \ll \dim(M^{|S|}) = |S|.$$

Como discutido, se existir uma relação topológica entre os dois espaços acima—ou seja, se existir uma correspondência entre as métricas adotadas em cada um—, é possível deslocar as operações baseadas na relação espacial entre os elementos de $M^{|S|}$ para o espaço de estados S. Essa propriedade pode ser explorada para contornar o crescimento superficial associado com o espaço markoviano.

Neste capítulo eu apresentei duas alternativas para se realizar a fatoração estocástica

no espaço S. A primeira delas é o algoritmo PISF-S, que nada mais é do que a extensão do algoritmo PISF para o caso em que a fatoração ocorre no espaço de estados. Um ponto positivo do algoritmo PISF-S é que ele pode a princípio ser aplicado a qualquer processo de decisão de Markov, desde que ele possua um espaço de estados que atenda às condições descritas na Seção 4.2.5. Uma outra vantagem desse algoritmo é que ele conta com fortes garantias teóricas, como discutido no capítulo anterior. Isso significa que o desempenho da política de decisão encontrada depende unicamente da qualidade da fatoração estocástica. Em particular, no caso em que a fatoração é exata tem-se a garantia de convergência para a política de decisão ótima, como anuncia o Corolário 3.1.

Uma outra maneira de realizar a fatoração estocástica no espaço de estados é usar o algoritmo KBSF. O algoritmo KBSF não conta com as mesmas garantias teóricas do PISF, mas empiricamente esse algoritmo parece gerar políticas de decisão de melhor qualidade, como pôde ser visto nos experimentos das seções 4.2.4, 4.3.3 e 4.4.1 em particular. Além disso, como mostrado na Seção 4.4.2, os resultados do KBSF no problema de equilibrar simultaneamente dois bastões são muito superiores aos do algoritmo LSPI, um dos mais importantes algoritmos de aprendizagem por reforço modernos.

É importante ressaltar que os algoritmos PISF e KBSF não devem ser interpretados como projetos acabados, mas como "arcabouços teóricos" a serem explorados. Tentei deixar isso claro neste capítulo mostrando diferentes maneiras de utilizá-los e citando na seqüência do texto várias possibilidades de modificações. Na Seção 4.4.2, por exemplo, levantei a possibilidade de uma versão do KBSF em que a coleta de dados ocorresse a cada iteração. Essa mesma idéia se aplica ao algoritmo PISF no caso em que ele é usado com amostras de transições. Pode-se pensar também em versões *on-line* desse algoritmos, como será discutido no Capítulo 5. Uma outra idéia seria desenvolver algoritmos "híbridos" que combinassem conceitos do PISF e do KBSF. As possibilidades são muitas, enfim. Mais importante do que enumerá-las uma-a-uma é enfatizar a interpretação dos algoritmos PISF e KBSF como concretizações específicas de uma abordagem mais genérica, que é a fatoração estocástica no espaço de estados. A partir dessa perspectiva surgem naturalmente idéias como as citadas acima.

Encerro este capítulo com a observação de que a fatoração estocástica em S *não é* de

forma alguma a única solução para o problema da maldição do crescimento superficial. Um exemplo de estratégia alternativa foi dado na Seção 4.4.1, em que os arquétipos foram selecionados segundo a "confiabilidade" dos estados do MDP construído a partir de amostras de transições. Além disso, é plenamente possível, pelo menos em princípio, desenvolver técnicas de fatoração estocástica em $M^{|S|}$ cujo custo computacional justifique a economia posterior no tempo de computação. A vantagem da fatoração estocástica no espaço markoviano é que ao se trabalhar diretamente com recompensas e probabilidades de transições é possível identificar padrões que vão além das relações espaciais entre os estados do MDP. Esse assunto constitui um tema interessante para pesquisas futuras.

Capítulo 5

Conclusão

"Publicamos nuestro libros para librarnos de ellos, para no pasar el resto de nuestras vidas corrigiendo borradores."

Jorge Luis Borges

É um tanto redundante descrever as minhas intenções em um capítulo intitulado "Conclusão." Parto, portanto, diretamente aos comentários a respeito do conteúdo de cada seção. Inicio com uma breve recapitulação de toda a tese, que tem como objetivo fornecer uma visão global do trabalho apresentado. Isso é feito na Seção 5.1. A seguir, nas Seções 5.2 e 5.3, discuto os dois temas que considero as contribuições mais importantes desta pesquisa: a fatoração estocástica e a interpretação da derivação de um MDP como um mapeamento. Em ambas as seções o objetivo é o mesmo: enfatizar os pontos mais importantes a respeito de cada assunto e generalizar as idéias para além do cenário considerado neste trabalho. Na Seção 5.4 discuto algumas possibilidades de trabalhos futuros. Encerro o capítulo e a tese na Seção 5.5, onde apresento meus comentários finais a respeito da pesquisa realizada.

5.1 Duzentas Páginas em Quatro

É difícil imaginar qual seria a reação de Garry Kasparov, considerado o maior enxadrista da história, se lhe dissessem que a sua habilidade extraordinária pode ser modelada como um processo de decisão de Markov. Se essa afirmação causasse alguma frustração de início, o mal-entendido poderia ser esclarecido com a informação de que isso nada tem de desabonador: muitos problemas importantíssimos da atualidade podem ser descritos por um prosaico MDP. Os processos de decisão de Markov, embora simples, são modelos poderosos porque conseguem absorver uma questão inerente a muitos problemas reais: a tensão existente entre benefícios imediatos e aqueles de longo prazo. Em outras palavras, esses modelos permitem um tratamento rigoroso de problemas de tomada de decisão seqüencial.

A busca por uma das políticas ótimas de um MDP é o problema estudado pela programação dinâmica e a sua versão incremental, a aprendizagem por reforço. A promessa dessas abordagens é sedutora: a programação de agentes "inteligentes" através de recompensas apenas. Assim, se a idéia fosse desenvolver um robô capaz de evitar obstáculos, por exemplo, bastaria lhe fornecer uma recompensa negativa—uma "punição"—toda vez que ele se chocasse com algum objeto. O robô ficaria responsável por descobrir *como* executar a tarefa, deixando o projetista com a definição dos objetivos apenas. No entanto, para que o cenário acima se torne realidade, a programação dinâmica também tem seus obstáculos a superar: embora ela funcione bem em problemas pequenos, a sua aplicação a problemas de grande porte depende da solução de uma séria questão de escalabilidade.

Em problemas com espaços de estados grandes ou contínuos, é necessário aplicar a programação dinâmica de maneira aproximada. Existem duas formas de fazê-lo. A primeira delas é aproximar a função de valor. Além de ser simples e direta, essa abordagem se beneficia da imensa massa teórica proveniente da aprendizagem supervisionada. No entanto, ela apresenta uma séria desvantagem: como as operações realizadas na programação dinâmica têm uma natureza recorrente, o uso de aproximadores pode facilmente levar a instabilidades e à divergência. Uma outra maneira de lidar com o problema de escalabilidade da programação dinâmica é criar um modelo aproximado do processo de decisão de Markov. Essa estratégia apenas estende, um passo além, o raciocínio subja-

cente à modelagem de um problema de tomada de decisão: se é possível representar o problema usando um MDP, talvez seja possível representar este último com um modelo reduzido. O objetivo nesse caso passa a ser encontrar um modelo compacto que retenha o máximo de informação a respeito do modelo original. A abordagem proposta neste trabalho visa alcançar este objetivo.

A fatoração estocástica nasceu de uma idéia simples: se for possível encontrar um pequeno conjunto de estados arquetípicos que representem bem a dinâmica de um MDP, pode-se reduzir o espaço de estados deste último simplesmente redirecionando as suas transições para esse conjunto. Como o nome sugere, essa estratégia de redirecionamento pode ser formalizada como a fatoração de uma matriz. Uma das maneiras de lidar com o modelo resultante do redirecionamento de transições é usar o algoritmo de iteração de política baseado na fatoração estocástica, ou simplesmente PISF. O PISF é um algoritmo bem-comportado, no sentido que o desempenho das políticas de decisão encontradas por ele depende unicamente da qualidade da fatoração estocástica do MDP. Em particular, uma fatoração exata leva necessariamente a uma das políticas ótimas do problema de tomada de decisão.

Há no entanto uma ironia nesta história: embora seja bem fundamentada teoricamente, a fatoração estocástica sofre do mesmo mal que ela pretende combater. Isso significa que os algoritmos capazes de fatorar estocasticamente um MDP também não escalam bem para espaços de estados grandes. Surge então uma situação curiosa: dada uma fatoração estocástica de um MDP, sabe-se que é possível derivar uma política de decisão em um número relativamente pequeno de operações aritméticas. No entanto, o *processo* de fatoração envolve ele mesmo um grande número de operações, de forma que dificilmente se justifica a economia computacional posterior no cálculo da política de decisão.

Felizmente, existe uma solução para este dilema. Ela se baseia na constatação de que os processos de decisão de Markov não são entidades abstratas sem qualquer ligação com o mundo real. Muito pelo contrário: todo MDP descreve a dinâmica de um processo de tomada de decisão. Assim, é correto afirmar que cada estado do modelo pode ser descrito por um conjunto de variáveis representando as suas características no fenômeno real. Formalmente, cada uma dessas variáveis corresponde a uma dimensão do espaço

de estados. Na programação dinâmica, esse espaço costuma ser negligenciado porque a descrição de um estado é irrelevante; uma vez que o processo de decisão de Markov tenha sido definido, toda a informação que lhe deu origem pode ser simplesmente descartada.

No entanto, em alguns casos o espaço de estados pode cumprir um papel importante. Quando se aproxima a função de valor, por exemplo, as entradas do aproximador costumam ser as variáveis desse espaço. No caso da fatoração estocástica, interessam espaços de estados que atendam a duas condições. A primeira delas é quase trivial: a dimensão desse espaço deve ser menor do que a sua cardinalidade (ou seja, devem existir mais estados do que variáveis). A segunda condição é um pouco mais restritiva, mas na prática ela é freqüentemente atendida: deve existir uma certa coerência entre o espaço de estados e a dinâmica do MDP. Em outras palavras, estados com características semelhantes devem ter associadas recompensas e probabilidades de transições parecidas.

Se a primeira condição for atendida e a segunda for violada apenas esporadicamente, pode-se realizar a fatoração estocástica no espaço de estados. Nesse caso, a determinação dos estados arquetípicos é feita segundo um critério espacial. Para contornar eventuais violações da segunda condição acima, é aconselhável redirecionar as transições de um estado para vários arquétipos, respeitando o grau de similaridade entre eles. A transposição da fatoração estocástica para o espaço de estados costuma gerar uma redução drástica no número de operações envolvidas no processo. Uma vez fatorado o MDP, pode-se adotar o algoritmo PISF para encontrar uma política de decisão em apenas uma fração do tempo que tomaria a solução do modelo original.

Essa estratégia pode ser estendida a espaços de estados contínuos. Como nesse caso a cardinalidade do espaço é infinita, a primeira condição acima é trivialmente atendida. Embora a segunda condição não seja garantida, ela costuma ser induzida pela continuidade das funções que descrevem a dinâmica do problema. Quando esse é o caso, pode-se utilizar a fatoração estocástica baseada em *kernels*, ou KBSF. O algoritmo KBSF permite a derivação de um MDP finito que descreve a dinâmica de um problema de decisão com espaço de estados contínuo. No atual estágio de desenvolvimento teórico o KBSF não conta com as mesmas garantias de desempenho do algoritmo PISF, mas nos experimentos computacionais realizados esse algoritmo gerou resultados promissores. Uma avaliação

mais detalhada desse algoritmo é necessária antes de se chegar a uma conclusão definitiva a seu respeito.

5.2 Fatoração Estocástica Revisitada

A principal contribuição desta tese para a comunidade científica é sem dúvida a fatoração estocástica, uma abordagem simples, genérica e flexível para a redução da dimensão de um processo de decisão de Markov. Uma característica particularmente desejável dessa abordagem é a sua possibilidade de ser interpretada como um redirecionamento de transições, o que permite que se pense em problemas complexos de uma maneira intuitiva. Embora neste trabalho a fatoração estocástica tenha sido encarada como uma estratégia para lidar com o problema de escalabilidade da programação dinâmica, é possível tratar do assunto em um grau maior de generalidade. Nesta seção apresento uma formulação mais genérica da fatoração estocástica e discuto algumas aplicações potenciais. Aproveito para enfatizar pontos importantes mencionados no trabalho e analisar umas poucas questões que não puderam ser discutidas no fluxo do texto.

Sejam $\mathbf{A} \in \mathbb{R}^{n \times p}$, $\mathbf{B} \in \mathbb{R}^{m \times p}$ e $\mathbf{D} \in \mathbb{R}^{n \times m}$, com $\mathbf{D}$ estocástica, tais que

$$\mathbf{DB} \approx \mathbf{A}. \tag{5.1}$$

Pode-se distinguir pelo menos quatro situações em que a relação acima tem algum interesse prático:

1. A matriz $\mathbf{A}$ é uma cadeia de Markov, ou seja, ela é quadrada e estocástica ($n = p$). As cadeias de Markov são modelos matemáticos genéricos com um enorme potencial de aplicações em áreas como a biologia, a astronomia e a estatística—além, é claro, da pesquisa operacional [25, 98]. Nesse caso, a relação (5.1) pode ser usada para reduzir a dimensão da cadeia. Como discutido, se $\mathbf{B}$ é uma matriz estocástica, então $\mathbf{BD}$ também é uma cadeia de Markov. Quando $m < n$, essa cadeia pode ser interpretada como uma versão compacta da primeira. Ela pode ser usada por exemplo para calcular uma aproximação da distribuição estacionária da cadeia $\mathbf{A}$, um problema importante que surge em muitas aplicações de interesse.

2. A matriz $\mathbf{A}$ é um processo de Markov isolado, isso é, uma cadeia de Markov com recompensas (ou custos) associados às transições ($n = p-1$). Como discutido, nesse caso a relação (5.1) pode ser explorada para se calcular uma aproximação da função de valor do processo a um custo computacional reduzido. Neste trabalho os processos de Markov foram usados para descrever o comportamento de uma política de decisão no contexto da iteração de política generalizada, mas é possível imaginar outras aplicações para esses modelos. Qualquer sistema dinâmico que apresente a propriedade markoviana e que tenha um custo associado com cada transição pode a princípio ser descrito por um processo de Markov. Pode-se por exemplo usar esse modelo para descrever a série temporal representando as ações de mercado de uma determinada companhia. Como nesse caso não é possível intervir na dinâmica do sistema, a função de valor serviria principalmente como uma ferramenta de análise.

3. Existem vários processos de Markov $\mathbf{A}_i$, todos construídos a partir da mesma cadeia. Este cenário ilustra como a fatoração estocástica pode ser útil também na *definição* de um modelo. Por exemplo, ao se modelar um sistema dinâmico como um processo de Markov, pode ser necessário avaliar vários sistemas de recompensas diferentes. Se esse for o caso, o cálculo da função de valor de cada sistema candidato envolveria $O(n^3)$ operações aritméticas. Uma alternativa seria fatorar a cadeia de Markov como descrito no item 1 acima e a seguir projetar cada vetor-recompensa no espaço alcançado pelos vetores-linha de $\mathbf{D}$, como em (3.43). Dependendo da qualidade da fatoração e das projeções, as funções de valor dos modelos reduzidos poderiam ser usadas como estimativas das funções reais.

4. A matriz $\mathbf{A}$ é um processo de decisão de Markov ($n = (p-1) \times |A|$). Nesse caso, a fatoração estocástica resulta em uma redução do modelo se e somente se $m < p-1$.

Neste trabalho eu me concentrei no item 4, ou seja, no uso da fatoração estocástica para a redução de um processo de decisão de Markov. A fatoração estocástica de um MDP pode se concretizar de duas maneiras: pode-se fatorar separadamente os processos de Markov associados com cada ação do problema, o que remete ao item 2 acima, ou pode-se fatorá-los conjuntamente, como visto na Seção 3.4. No primeiro caso o resultado é um novo MDP de tamanho reduzido, que pode ser resolvido por qualquer algoritmo de

programação dinâmica convencional. O segundo caso resulta em um modelo que precisa de um tratamento especial. Na seção abaixo discuto os prós e contras de cada uma dessas alternativas, usando os algoritmos PISF e KBSF como ilustração.

5.2.1 Duas maneiras de se fatorar um MDP

Existe uma diferença "filosófica" entre os algoritmos PISF e KBSF. Enquanto este último gera um MDP de tamanho reduzido, o algoritmo PISF manipula um modelo com uma estrutura particular. É natural questionar os motivos dessa complicação adicional.

No algoritmo PISF os processos $\mathbf{M}^a$ do MDP são fatorados em conjunto, ou seja, nesse caso a expressão (5.1) seria:

$$\mathbf{DW} \approx \mathbf{M},$$

onde $\mathbf{M}$ é uma matriz $|S||A| \times |S|+1$ descrevendo o processo de decisão de Markov (veja (3.45)). Quando se está trabalhando no espaço $M^{|S|}$, como no Capítulo 3, faz sentido realizar a fatoração estocástica dos processos $\mathbf{M}^a$ conjuntamente, porque o algoritmo que determinará as linhas de $\mathbf{W}$ é capaz de "enxergar" todo o MDP. De fato, uma vez que os processos $\mathbf{M}^a$ tenham sido combinados para formar um MDP $\mathbf{M}$, como em (3.45), não há razão para fazer qualquer distinção em relação à ação associada com cada linha $\mathbf{m}_i$ no modelo original. Além disso, é a fatoração conjunta dos processos $\mathbf{M}^a$ que torna possível, pelo menos em tese, contornar o crescimento superficial de $M^{|S|}$, como discutido na Seção 3.4.

No entanto, quando a fatoração estocástica ocorre no espaço S, a situação muda, porque o algoritmo escolhido para fatorar o MDP não manipulará os vetores-linha $\mathbf{m}_i$ diretamente (veja por exemplo o Algoritmo 4.1). Nesse caso, cada estado s_i representa $|A|$ vetores $\mathbf{m}_i^a$, e não há nenhuma vantagem aparente em se fatorar os processos $\mathbf{M}^a$ conjuntamente. Pode-se pensar portanto em adotar uma estratégia bem mais simples no algoritmo PISF-S: ao invés de manter uma matriz $\mathbf{W}$ global, é possível realizar a fatoração dos processos de Markov $\mathbf{M}^a$ isoladamente. Isso daria origem a $|A|$ matrizes de transições $\bar{\mathbf{P}}^a \approx \mathbf{K}^a\mathbf{D}^a$ e ao mesmo número de vetores $\bar{\mathbf{r}}^a$, que poderiam então ser submetidos a algoritmos convencionais da programação dinâmica. Além do ganho em

"portabilidade"—já que não seria mais necessário um algoritmo especializado para lidar com o modelo resultante—, haveria um ganho em tempo de computação, uma vez que os modelos gerados seriam mais compactos (lembre-se que a cada arquétipo q_i selecionado no algoritmo PISF-S são adicionadas $|A|$ linhas à matriz $\mathbf{W}$). Como se não bastasse, essa nova versão do algoritmo convergiria garantidamente para uma função de valor específica, ao passo que a versão atual pode ficar indefinidamente visitando as proximidades de $\mathbf{v}^*$.

Existiria alguma vantagem em se realizar a fatoração estocástica conjunta dos processos $\mathbf{M}^a$? Do ponto de vista teórico, sim. Note que no algoritmo PISF, embora a avaliação das políticas se dê em um modelo reduzido, a *definição* de uma política ainda ocorre nos estados originais do problema. Isso torna relativamente simples determinar o erro incorporado ao processo de aprendizagem nas etapas de avaliação e melhoria de política (neste último caso, zero). É a possibilidade de mensurar precisamente o erro envolvido em cada fase do algoritmo o que torna possível a derivação da cota superior apresentada no Corolário 3.1—que se traduz em uma garantia de desempenho em relação ao MDP original. Em particular, quando a fatoração estocástica é exata, tem-se a garantia de convergência do algoritmo PISF para uma das políticas ótimas do problema. Quando os processos $\mathbf{M}^a$ são fatorados isoladamente, as políticas de decisão são definidas nos arquétipos, e não nos estados do modelo original. Nesse caso, não é tão claro como calcular o erro introduzido pela avaliação e melhoria de política realizadas no modelo reduzido—e portanto fica difícil estabelecer limites teóricos baseados na Proposição 3.4. Esta é, na minha opinião, a grande lacuna teórica a ser preenchida neste trabalho, como será discutido à frente.

Se a estratégia de fatoração conjunta dos processos $\mathbf{M}^a$ oferece melhores garantias teóricas, é natural inverter o raciocínio e perguntar se ela não poderia ser adotada no algoritmo KBSF. A resposta aqui é mais uma vez afirmativa: para fazê-lo, bastaria "empilhar" as matrizes $\mathbf{K}^a$ retornadas pelo Algoritmo 4.2 e definir as matrizes $\mathbf{D}^a$ como feito no Algoritmo 4.1. Nesse caso, o número de arquétipos usados pelo KBSF seria multiplicado por $|A|$, mas por outro lado ter-se-ia o Corolário 3.1 como uma garantia teórica em relação ao MDP que seria definido pelo algoritmo KBRL original.

Fica claro, portanto, que a estratégia usada para realizar a fatoração estocástica é uma decisão de projeto, e não uma característica indissociável dos algoritmos PISF e KBSF.

Em particular, pode-se pensar em uma versão do primeiro em que os processos $\mathbf{M}^a$ são fatorados isoladamente ou uma versão do segundo em que eles o são de maneira conjunta. A escolha sobre qual estratégia adotar envolve no atual estágio de desenvolvimento teórico a seguinte tensão: uma fatoração isolada gera modelos mais portáteis e compactos, enquanto a fatoração conjunta oferece garantias de desempenho em relação ao modelo original.

5.2.2 As dimensões da fatoração estocástica

No início desta seção discuti diferentes versões da fatoração estocástica que são caracterizadas pelas propriedades da matriz $\mathbf{A}$ na expressão (5.1). É possível identificar outras dimensões de variação do problema, de acordo com as restrições impostas sobre as matrizes de desvio e de retorno:

Matriz de desvio : No caso mais geral, a única condição que se impõe sobre a matriz $\mathbf{D}$ é que ela seja uma matriz estocástica. No entanto, é possível restringir o número de elementos não-nulos por linha, o que equivale a determinar o número k de arquétipos usados para representar cada estado. No caso extremo, em que $k = 1$, a fatoração se reduz a um problema de alocação de estados a arquétipos. Como discutido no texto, o parâmetro k pode ser visto como uma estratégia para controlar a tensão entre o *bias* e a variância na aproximação do MDP. Valores maiores para esse parâmetro tendem a gerar aproximações mais suaves, mas também distorcem mais a dinâmica do MDP original.

Matriz de retorno : As linhas da matriz $\mathbf{B}$ na expressão (5.1) podem ser livres ou representar um subconjunto dos vetores-linha de $\mathbf{A}$. Neste último caso, a definição dos arquétipos se reduz a um problema de otimização combinatória. Se por um lado essa simplificação tende a facilitar a fatoração, por outro lado ela também limita o poder de expressão do modelo.

Portanto, a versão mais restrita do problema é aquela em que $\mathbf{D}$ é uma matriz determinística e as linhas de $\mathbf{B}$ são um subconjunto das linhas de $\mathbf{A}$. O algoritmo k-medoids,

por exemplo, encontra uma fatoração desse tipo. No outro extremo, tem-se apenas a restrição de estocasticidade sobre as matrizes $\mathbf{D}$ e $\mathbf{B}$.[1] Esse é o tipo de fatoração retornada pelo método adaptado de Lee e Seung [87], mostrado no Algoritmo 3.1. Existem entre esses dois extremos várias versões intermediárias do problema. O algoritmo k-means, por exemplo, retorna uma matriz $\mathbf{D}$ determinística, mas não restringe as linhas de $\mathbf{B}$ aos vetores-linha de $\mathbf{A}$.

Note que nos exemplos de algoritmos dados acima as matrizes $\mathbf{D}$ e $\mathbf{B}$ são determinadas conjuntamente. É possível desacoplar o problema em duas fases distintas: determinação dos arquétipos—ou definição da matriz $\mathbf{B}$—e redirecionamento de transições, ou determinação da matriz $\mathbf{D}$. Esse tipo de abordagem permite que se concentre nos aspectos particulares de cada etapa da fatoração estocástica. Por exemplo, pode-se definir de forma independente quais serão as restrições impostas sobre $\mathbf{D}$ e $\mathbf{B}$ ou se é conveniente deslocar uma ou as duas etapas da fatoração para o espaço de estados.

5.3 A Modelagem Vista como um Mapeamento

Além da fatoração estocástica e dos algoritmos derivados, uma outra contribuição importante deste trabalho é a interpretação da modelagem como um mapeamento entre o espaço de estados e um outro conjunto, que chamei de espaço markoviano. Embora a relação entre S e $M^{|S|}$ esteja presente tacitamente em muitos textos da área, acredito que a visão da derivação de um MDP como um mapeamento explícito, como em (4.2), seja uma novidade. Aliás, a própria interpretação de $M^{|S|}$ como um espaço métrico não me parece ainda ter sido explorada.

Como discutido na Seção 4.2, na programação dinâmica o espaço de estados serve apenas para definir o MDP; uma vez que isso tenha sido feito, as variáveis do problema podem ser desconsideradas. De fato, não é raro encontrar textos da área que se referem aos estados apenas pela posição que eles ocupam no MDP (ou seja, o estado $s_i \in S$, que pode ser um vetor multidimensional, se torna o índice $i \in \mathbb{N}$) [22]. Na aprendizagem por

[1] No caso em que $\mathbf{A}$ é um processo de Markov ou um MDP, a restrição de estocasticidade sobre $\mathbf{B}$ é apenas parcial, como discutido na Seção 3.3.2.

reforço, a necessidade de se aproximar a função de valor traz o espaço S para o centro da discussão, mas nesse caso o espaço markoviano é que costuma ser negligenciado.

A distinção entre os espaços S e $M^{|S|}$ e o estudo das suas relações podem trazer benefícios práticos, como discutido extensivamente no Capítulo 4. Mais importante do que isso, no entanto, é a possibilidade de analisar as características de um problema de tomada de decisão seqüencial. Um exemplo nesse sentido é a identificação de problemas em que existe uma certa "localidade" nos mapeamentos $\varphi^a : S \mapsto M^{|S|}$, com $a \in A$. Neste trabalho essa propriedade foi explorada para realizar em S operações baseadas em relações espaciais entre os elementos de $M^{|S|}$. Isso faz sentido porque a dimensão do primeiro costuma ser bem menor do que a do segundo, isso é, $\dim(S) \ll |S|$. No entanto, a relação topológica entre os espaços S e $M^{|S|}$ pode ser explorada de outras formas. Por exemplo, é razoável supor que quando A é um conjunto contínuo a localidade das funções φ^a induz a mesma propriedade no mapeamento $\pi^* : S \mapsto A$. Ou seja: estados parecidos, ações parecidas. Isso claramente levanta a possibilidade de desenvolvimento de técnicas específicas para esse cenário.

Entretanto, a relação entre os espaços S e $M^{|S|}$ não se restringe à sua topologia. Mesmo no caso em que esses espaços não são dotados de uma métrica, a análise das funções φ^a pode ser esclarecedora. A seguir mostro um exemplo de como o estudo dessas funções pode ajudar a entender aspectos fundamentais de um problema de tomada de decisão seqüencial.

5.3.1 A dimensão intrínseca de um problema de tomada de decisão

Uma questão freqüentemente citada como uma limitação fundamental da programação dinâmica é a chamada maldição da dimensionalidade de Bellman [15], discutida na Seção 2.4. Superficialmente, a maldição de Bellman diz que o número de estados de um problema cresce exponencialmente com a dimensão do seu espaço de estados. Isso limitaria a aplicação da programação dinâmica a problemas com um pequeno número de variáveis.

A maldição de Bellman é de fato um obstáculo a ser enfrentado pela programação

dinâmica. Note, no entanto, que há uma sutileza no seu enunciado. Quando se usa o número de variáveis para caracterizar o nível de dificuldade de um problema, supõe-se implicitamente que a modelagem do MDP tenha sido feita de forma ótima. A maneira mais fácil de entender essa questão é observar um contra-exemplo, ou seja, um caso em que a modelagem tenha sido feita de forma inadequada. Suponha que na modelagem do problema do carro preso no vale, discutida na Seção 4.1.2, decida-se por incluir na descrição dos estados a informação sobre a temperatura corrente. Ou seja, cada estado do modelo seria agora um vetor $s_i = [x, \dot{x}, t]$, onde t é a temperatura ambiente medida em uma unidade qualquer. Obviamente, a informação contida na variável t é totalmente irrelevante para a solução do problema. No entanto, ela causaria inevitavelmente um "inchaço" do espaço de estados. No caso de uma discretização com ι intervalos, por exemplo, o número $|S|$ de estados cresceria de ι^2 para ι^3.

Por ser muito improvável, o exemplo acima pode causar a impressão de que uma redundância na definição das variáveis é sempre conseqüência de um erro crasso de modelagem. Infelizmente, esse não é o caso. Em muitos problemas reais, existe uma descrição "natural" dos estados que pode não ser a mais econômica. Além disso, mesmo que se tenha consciência da redundância nas variáveis, pode ser difícil identificá-la ou impossível removê-la. Pense no caso de um robô com vários sensores, por exemplo. Podem haver dependências entre as variáveis que não são evidentes a princípio. Mesmo no caso em que essa dependência é identificada, pode não ser viável do ponto de vista tecnológico construir um sensor que removeria a dependência de outros dois.

Fica óbvio, portanto, que a verdadeira dimensão de um problema não é necessariamente uma função exponencial do seu número de variáveis. O estudo da relação entre S e $M^{|S|}$ permite uma análise mais clara dessa questão. Observe que se existir alguma redundância na definição de S, ela irá inevitavelmente se manifestar nos mapeamentos φ^a. No caso do carro preso no vale, por exemplo, todos os estados s_i que concordam no valor das variáveis x e $\dot{x}$ dão origem aos mesmos vetores $\mathbf{m}_i^a$, independentemente de t. Isso indica claramente que há uma dependência entre as variáveis do problema. Obviamente, a redundância na definição de S não se manifestará sempre de uma maneira tão evidente. Suponha que o robô do exemplo acima tenha sido equipado com três sensores, que podem ser representados pelas variáveis x_1, x_2 e x_3. Imagine que exista uma dependência

entre as variáveis x_1 e x_2; mais precisamente, suponha que seja possível substituí-las por $x' = x_1 + x_2$ sem perda para a descrição do problema. Nesse caso, a redundância na definição de S se manifestaria de maneira diferente nos mapeamentos φ^a: todos os estados s_i com o mesmo valor para x_3 e para a *soma* $x_1 + x_2$ originariam os mesmos vetores $\mathbf{m}_i^a$.

A partir destas idéias, pode-se definir a *dimensão intrínseca* de um problema como o menor número de variáveis necessário para determinar os mapeamentos φ^a. Essa definição pode ser útil de várias maneiras. Em primeiro lugar, ela constitui um critério bem definido para classificar problemas de tomada de decisão seqüencial markovianos. A partir dela fica mais fácil também compreender a maldição de Bellman, que faz mais sentido quando enunciada com essa medida. Finalmente, o conceito de dimensão intrínseca ajuda a entender a própria fatoração estocástica, que pode ser vista como uma tentativa de reduzir um MDP à sua verdadeira dimensão.

5.4 Trabalhos Futuros

A fim de maximizar o alcance das idéias apresentadas, procurei neste trabalho manter a discussão no nível mais alto de generalidade possível. Uma conseqüência disso é que a pesquisa pode ser especializada em inúmeras direções. Dentre as possibilidades de trabalhos futuros, duas me parecem particularmente promissoras: a derivação de garantias teóricas em relação à fatoração de processos de Markov em separado e a incorporação da fatoração estocástica ao processo de aprendizagem. Essas duas idéias são abordadas em detalhes nas Seções 5.4.1 e 5.4.2. Na Seção 5.4.3 discuto brevemente outras possibilidades de extensão da pesquisa.

5.4.1 Garantias teóricas para a fatoração isolada de um MDP

Em várias passagens deste trabalho eu enfatizei a importância do Corolário 3.1 como uma garantia teórica de desempenho para o algoritmo PISF. Como discutido, a qualidade das políticas retornadas por esse algoritmo depende unicamente da qualidade da fatoração estocástica do MDP. Na Seção 5.2.1 afirmei que essa garantia é perdida quando os proces-

sos de Markov de um MDP são fatorados isoladamente, como ocorre no algoritmo KBSF. Isso acontece porque nesse caso as políticas de decisão são definidas nos arquétipos, o que dificulta uma comparação direta com o processo de aprendizagem que ocorreria no MDP original.

Como nos experimentos realizados neste trabalho o algoritmo KBSF se comportou melhor do que o PISF, pode-se pensar que essa seja uma questão menos importante. No entanto, uma análise mais criteriosa revela que isso não é verdade. Quando um algoritmo é bem entendido como o PISF, é possível estabelecer condições sob as quais ele se comportaria de maneira ótima. Uma vez que a configuração ótima seja conhecida, fica fácil identificar e mensurar desvios em relação a esse cenário ideal—e portanto tentar minimizá-los. No caso do algoritmo PISF, a configuração ideal corresponde a uma fatoração estocástica exata. Assim, o erro de fatoração serve como uma medida da "distância" em relação à configuração ótima do algoritmo. Quando os processos $\mathbf{M}^a$ são fatorados individualmente isso não ocorre, porque uma fatoração exata não é garantia de convergência para uma das políticas ótimas de um MDP.

Para ilustrar esta questão, eu realizei uma bateria de experimentos em que os processos de Markov podiam ser fatorados de maneira exata. Para tal, inverti o procedimento que vinha sendo adotado até aqui: ao invés de determinar as matrizes de desvio e de retorno a partir de um MDP, obtive este último a partir das duas primeiras. Especificamente, em cada experimento foram geradas uma matriz de retorno $\mathbf{K} \in \mathbb{R}^{m \times |S|}$ e $|A|$ matrizes de desvio $\mathbf{D}^a \in \mathbb{R}^{|S| \times m}$. A estratégia usada para gerar as matrizes $\mathbf{K}$ foi a mesma utilizada na Seção 3.3.2 para construir as matrizes $\mathbf{P}^a$. As matrizes de desvio foram geradas de maneira aleatória, respeitando, claro, a restrição de estocasticidade. Ao lado de cada matriz $\mathbf{K}$ foi gerado um vetor $\mathbf{r} \in \mathbb{R}^m$, também usando a mesma estratégia adotada na Seção 3.3.2. A partir desses elementos, fica fácil definir os processos de Markov simplesmente fazendo $\mathbf{P}^a = \mathbf{D}^a\mathbf{K}$ e $\mathbf{r}^a = \mathbf{D}^a\mathbf{r}$ para todo $a \in A$. A versão reduzida dos MDPs foi obtida segundo uma estratégia que já não deve causar surpresas neste ponto do trabalho: $\bar{\mathbf{P}}^a = \mathbf{K}\mathbf{D}^a$ e $\bar{\mathbf{r}}^a = \mathbf{r}$. A Figura 5.1 mostra a diferença entre as políticas π^* e $\bar{\pi}^*$ calculadas pelo algoritmo de iteração de política.[2]

[2]Note que a falta de correlação entre χ_m e $|A|$ é provavelmente uma conseqüência da estratégia usada para construir os MDPs, em que uma mesma matriz $\mathbf{K}$ e um mesmo vetor $\mathbf{r}$ foram usados para gerar todos

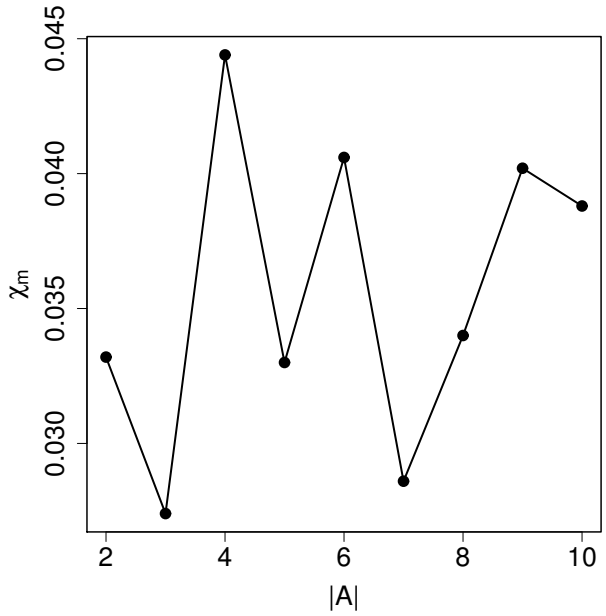


Figura 5.1: Erro na política de decisão obtida a partir da fatoração estocástica de um MDP em que cada processo de Markov é fatorado isoladamente. Os MDPs foram construídos como explicado no texto, com $|S| = 100$ e $m = 20$. As matrizes $\mathbf{K}$ foram geradas com $\vartheta = 20$, $\sigma = 1$ e $\eta = 0$. Os valores se referem a uma média de 50 execuções do algoritmo de iteração de política.

Como no cenário descrito acima a fatoração estocástica é exata, o algoritmo PISF convergiria garantidamente para uma das políticas ótimas dos MDPs. Infelizmente, o mesmo não ocorre quando os processos $\mathbf{M}^a$ são fatorados em separado, como fica claro na Figura 5.1. Nesse caso especificamente a diferença entre as política π^* e $\bar{\pi}^*$ é pequena, mas é difícil estimar qual seria essa diferença em MDPs com outras características, uma vez que a fonte do erro não é conhecida. Este exemplo mostra a importância de se estabelecer limites teóricos que forneçam alguma garantia em relação à fatoração isolada dos processos $\mathbf{M}^a$, nos moldes da cota em (3.49). Mesmo se esses limites tivessem pouca utilidade como uma estimativa de desempenho real, eles pelo menos deixariam claro quais são as variáveis que determinam a qualidade das políticas derivadas dos MDPs reduzidos. Na minha opinião, a derivação de garantias teóricas para a fatoração isolada de um MDP constitui um dos passos mais importantes a serem dados na continuidade deste trabalho.

os processos de Markov $\mathbf{M}^a$.

5.4.2 Fatoração estocástica em tempo real

Para facilitar a análise teórica e a avaliação empírica da fatoração estocástica, neste trabalho eu me concentrei no cenário em que a construção de um modelo reduzido do MDP ocorre *offline*, isso é, antes de se iniciar a busca por uma política de decisão. Na prática, no entanto, pode ser interessante incorporar a fatoração ao processo de aprendizagem. Nesse caso a construção do modelo ocorreria paralelamente à coleta de dados, como no esquema mostrado na Figura 2.5. A possibilidade de intervenção mútua entre esses dois processos é lucrativa para ambos, como discutido na Seção 2.3.4. Por um lado, a exploração do ambiente pode ser feita de uma maneira mais inteligente, com base no modelo aproximado do MDP; por outro, a própria construção do modelo se beneficia da qualidade dos dados que ajudou a gerar.

Esse uso mais eficiente da informação estenderia as possibilidades de aplicação da fatoração estocástica a problemas de tomada de decisão de dimensões maiores. Além disso, o esquema de aprendizagem descrito acima tornaria possível que a fatoração ocorresse em tempo real, enquanto o agente interagisse com o ambiente. Nesse caso, o desafio seria fixar as dimensões das matrizes $\mathbf{D}^a$ e $\mathbf{K}^a$. Uma idéia seria utilizar um esquema semelhante à atribuição proporcional para atualizar todas as linhas dessas matrizes a cada transição. As multiplicações $\mathbf{K}^a\mathbf{D}^a$ poderiam ser calculadas dentro de um cronograma pré-definido ou disparadas por alguma informação disponível em tempo de execução. Como nesse caso o modelo do MDP é sucessivamente refinado, a definição dos arquétipos poderia ser reavaliada esporadicamente, com os recursos sendo gradativamente concentrados nas regiões mais importantes do espaço de estados.

5.4.3 Outras possibilidades de extensão da pesquisa

A pesquisa descrita nesta tese pode ser estendida em várias outras direções além daquelas discutidas acima. Abaixo listo cinco possibilidades. Como será visto, algumas das idéias levantadas constituem extensões genuínas do estudo realizado, ao passo que outras serviriam para preencher lacunas deixadas por este trabalho.

Posto estocástico: Na Seção 3.2.1 introduzi o conceito de posto estocástico de uma matriz, que corresponde ao menor número de arquétipos necessários para a sua fatoração exata. No entanto, não apresentei uma maneira de se *determinar* esse número. Da mesma forma que é possível encontrar uma base vetorial para uma matriz qualquer, seria extremamente útil um método para determinar uma "base estocástica" de uma matriz, ou seja, um conjunto mínimo de arquétipos capaz de representá-la de forma exata. Note que utilizando esse método um MDP poderia ser facilmente reduzido à sua dimensão intrínseca (Seção 5.3.1).

Fatoração estocástica no espaço markoviano: Existem pelo menos duas vantagens em realizar a fatoração (exata ou aproximada) diretamente em $M^{|S|}$. Em primeiro lugar, a necessidade de uma correspondência topológica desse espaço com o espaço de estados é removida. Além disso, quando se fatora o MDP em $M^{|S|}$ é possível identificar relações entre os vetores $\mathbf{m}_i^a$ que independem da sua distribuição espacial. No entanto, como discutido na Seção 3.3.4, o crescimento superficial do espaço $M^{|S|}$ torna a fatoração estocástica computacionalmente inviável nesse espaço. Isso é, pelo menos quando se considera os algoritmos que seriam os candidatos naturais para a tarefa. O desenvolvimento de métodos específicos ou a modificação de algoritmos genéricos que permitissem que a fatoração estocástica fosse realizada diretamente no espaço $M^{|S|}$ seria uma extensão importante deste trabalho.

Correspondência topológica: Na Seção 4.2.1 apresentei as relações (4.3) e (4.4) como exemplos de correspondências topológicas fortes entre os espaços $M^{|S|}$ e S. Logo a seguir, no entanto, afirmei que essas condições raramente se cumprem na prática, e a partir daí adotei a noção vaga de uma correspondência "razoável" entre esses dois espaços. Este assunto claramente requer uma investigação mais aprofundada. Para que a fatoração estocástica (e potencialmente outras operações) possam ser realizadas de maneira confiável no espaço de estados, é importante estabelecer noções mais precisas do que seria uma correspondência topológica entre $M^{|S|}$ e S. Uma outra questão que merece ser analisada com mais cuidado é o papel da atribuição proporcional como um recurso para contornar eventuais inconsistências topológicas entre o espaço de estados e o espaço markoviano.

Definição dos arquétipos: Neste trabalho eu me concentrei no caso em que os arquétipos são determinados segundo a sua posição no espaço, seja esse $M^{|S|}$ ou S. Duas exceções são o algoritmo de Lee e Seung, apresentado na Seção 3.3.2, e o critério de "confiabilidade" dos estados, usado na Seção 4.4.1. É possível pensar em outras estratégias para definir os arquétipos a serem usados na fatoração estocástica. Uma possibilidade é desenvolver heurísticas baseadas em intuição a respeito do problema. Por exemplo, quando se está trabalhando no espaço $M^{|S|}$, pode-se selecionar como arquétipos as linhas das matrizes $\mathbf{P}^a$ que mais se aproximam de um vetor determinístico. A idéia é que esses vetores estão mais próximos dos vértices do simplex que contém as matrizes $\mathbf{P}^a$ (veja discussão na Seção 3.2.1). Uma outra idéia é usar conhecimento prévio a respeito de um problema para identificar estados que sejam representativos da sua dinâmica. Por exemplo, em um sistema para o gerenciamento de múltiplos elevadores, é razoável esperar que configurações em que todas as unidades encontram-se em andares intermediários sejam mais representativas do que aquelas nas quais um subconjunto dos elevadores está no primeiro ou no último andar.

Aplicação real: Embora eu tenha tentado sempre que possível embasar as discussões com experimentos computacionais, algumas das idéias apresentadas neste trabalho não foram suficientemente testadas, e outras nem mesmo chegaram a ser tentadas na prática. Um exemplo neste último grupo que me parece particularmente promissor é a aplicação do algoritmo PISF à discretização suave realizada pelo KBRL. Uma investigação empírica detalhada significaria um avanço no sentido de entender a relevância dos algoritmos propostos na prática. Particularmente importante seria a aplicação da fatoração estocástica a um problema de tomada de decisão real. Esse tipo de experiência serviria não apenas para avaliar o desempenho dos algoritmos em um cenário real, como também para destacar deficiências que não aparecem em experimentos com problemas artificiais de pequeno porte.

5.5 Considerações Finais

Esta tese se aproxima mais de um diário de bordo de uma viagem em curso do que de um relatório de uma jornada encerrada. Assim sendo, ela traz todas as imperfeições inerentes a uma descrição de uma pesquisa ainda em andamento. Em primeiro lugar, como uma grande quantidade de assuntos foi discutida, nem todos receberam a atenção merecida, seja na apresentação, seja na pesquisa propriamente dita. Suspeito, inclusive, que algumas das questões levantadas podem se tratar de problemas já abordados na literatura. Um exemplo: não seria de todo surpreendente se, após o encerramento deste trabalho, eu encontrasse nas páginas de um livro de álgebra linear um algoritmo para determinar o posto estocástico de uma matriz. Tampouco seria uma grande surpresa me deparar com algum resultado da área que permitisse a derivação de garantias teóricas para uma fatoração isolada. Acredito, no entanto, que esses lapsos eventuais não afetem de forma significativa o conteúdo do trabalho.

Um outro problema em escrever de dentro do trem é que apenas na última estação é que se conhece toda a viagem. Ao tentar organizar idéias e conceitos de uma forma coerente, adquiri uma visão global da pesquisa que não tinha antes da redação deste trabalho. Se pudesse começar tudo de novo, faria muita coisa diferente. Mas isso não chega a ser um grande problema: encerro com a convicção de que esta é uma descrição imperfeita de uma contribuição pequena, mas efetiva, para a programação dinâmica e para a aprendizagem por reforço.

Referências Bibliográficas

[1] ANDERSON, C. W. *Learning and Problem Solving with Multilayer Connectionist Systems*. PhD thesis, Computer and Information Science, University of Massachusetts, 1986.

[2] ANDERSON, C. W. Learning to control an inverted pendulum using neural networks. *IEEE Control Systems Magazine 9* (1989), 31–37.

[3] ARTHUR, D., E VASSILVITSKII, S. How slow is the k-means method? In *Proceedings of the twenty-second annual symposium on computational geometry* (New York, NY, USA, 2006), ACM, pp. 144–153.

[4] ATKESON, C., E SANTAMARIA, J. A comparison of direct and model-based reinforcement learning. In *Proceedings of the IEEE International Conference on Robotics and Automation* (1997), vol. 4, pp. 3557–3564.

[5] BAIRD, L. C. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the International Conference on Machine Learning* (1995), pp. 30–37.

[6] BARBER, C. B., DOBKIN, D. P., E HUHDANPAA, H. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software 22*, 4 (1996), 469–483.

[7] BARRETO, A. M. S., E ANDERSON, C. W. Restricted gradient-descent algorithm for value-function approximation in reinforcement learning. *Artificial Intelligence 172*, 4-5 (2008), 454–482.

[8] BARTO, A. G. *Handbook of Brain Theory and Neural Networks*, 2 ed. MIT Press, 2003, ch. Reinforcement learning, pp. 963–968.

[9] BARTO, A. G., E MAHADEVAN, S. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems 13*, 4 (2003), 341–379.

[10] BARTO, A. G., SUTTON, R. S., E ANDERSON, C. W. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics 13* (1983), 834–846.

[11] BAXTER, J., E BARTLETT, P. Direct gradient-based reinforcement learning: I. Gradient estimation algorithms. Relatório técnico, Research School of Information Sciences and Engineering, Australian National University, July 1999.

[12] BAXTER, J., WEAVER, L., E BARTLETT, P. Direct gradient-based reinforcement learning: II. Gradient ascent algorithms and experiments. Relatório técnico, Research School of Information Sciences and Engineering, Australian National University, July 1999.

[13] BELLMAN, R. E. *Dynamic Programming*. Princeton University Press, 1957.

[14] BELLMAN, R. E. A markov decision process. *Journal of Mathematical Mechanics 6* (1957), 679–684.

[15] BELLMAN, R. E. *Adaptive Control Processes*. Princeton University Press, 1961.

[16] BENBRAHIM, H., E FRANKLIN, J. Biped dynamic walking using reinforcement learning. *Robotics and Autonomous Systems Journal* (December 1997).

[17] BERTSEKAS, D. *Nonlinear Programming*. Athena Scientific, 1995.

[18] BERTSEKAS, D., E CASTANON, D. Adaptive aggregation methods for infinite horizon dynamic programming. *IEEE Transactions on Automatic Control 34*, 6 (June 1989), 589–598.

[19] BERTSEKAS, D. P. Distributed dynamic programming. *IEEE Transactions on Automatic Control 27* (1982), 610–616.

[20] BERTSEKAS, D. P. *Dynamic programming: deterministic and stochastic models*. Prentice-Hall, 1987.

[21] BERTSEKAS, D. P., E TSITSIKLIS, J. N. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, 1995.

[22] BERTSEKAS, D. P., E TSITSIKLIS, J. N. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

[23] BEYER, K., GOLDSTEIN, J., RAMAKRISHNAN, R., E SHAFT, U. When is "nearest neighbor" meaningful? *Lecture Notes in Computer Science 1540* (1999), 217–235.

[24] BEZDEK, J. C. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Plenum Press, New York, 1981.

[25] BHARUCHA-REID, A. T. *Elements of the Theory of Markov Processes and Their Applications*. McGraw-Hill, New York, 1960.

[26] BOYAN, J. A. Technical update: Least-squares temporal difference learning. *Machine Learning 49* (2002), 233–246.

[27] BOYAN, J. A., E MOORE, A. W. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in Neural Information Processing Systems* (1995), The MIT Press, pp. 369–376.

[28] BRADTKE, S. J., E BARTO, A. G. Linear least-squares algorithms for temporal difference learning. *Machine Learning 22*, 1/2/3 (1996), 33–57.

[29] BRAFMAN, R. I., E TENNENHOLTZ, M. R-MAX - a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research 3* (2003), 213–231.

[30] BRIN, S., E PAGE, L. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the Seventh International World Wide Web Conference* (1998), Elsevier, p. 107–117.

[31] CHAND, D. R., E KAPUR, S. S. An algorithm for convex polytopes. *Journal of the ACM 17*, 1 (1970), 78–86.

[32] CHEN, J.-C. The nonnegative rank factorizations of nonnegative matrices. *Linear Algebra and its Applications 62* (1984), 207–217.

[33] CHOW, C., E TSITSIKLIS, J. An optimal multigrid algorithm for continuous state discrete time stochastic control. *IEEE Transactions on Automatic Control 36*, 8 (1991), 898–914.

[34] CLARKSON, K. L., E SHOR, P. W. Applications of random sampling in computational geometry, II. *Discrete and Computational Geometry 4*, 1 (1989), 387–421.

[35] CRITES, R. H., E BARTO, A. G. Improving elevator performance using reinforcement learning. In *Advances in Neural Information Processing Systems* (1996), vol. 8, The MIT Press, pp. 1017–1023.

[36] CUTLER, A., E BREIMAN, L. Archetypal analysis. *Technometrics 36*, 4 (1994).

[37] DANTZIG, G., E WOLFE, P. Decomposition principle for dynamic programs. *Operations Research* (1960).

[38] DAYAN, P. The convergence of TD(λ) for general λ. *Machine Learning 8* (1992), 341–362.

[39] DAYAN, P., E SEJNOWSKI, T. TD(λ) converges with probability 1. *Machine Learning 14* (1994), 295–301.

[40] D'EPENOUX, F. A probabilistic production and inventory problem. *Management Science* (1963).

[41] DONOHO, D., E STODDEN, V. When does non-negative matrix factorization give a correct decomposition into parts? In *Advances in Neural Information Processing Systems* (2004).

[42] DORIGO, M., E COLOMBETTI, M. Robot shaping: Developing autonomous agents through learning. *Artificial Intelligence 71* (1994), 321–370.

[43] DORIGO, M., E STÜTZLE, T. *Ant Colony Optimization*. MIT Press, 2004.

[44] DUNN, J. C. A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters. *Journal of Cybernetics* (1973).

[45] EHRGOTT, M. *Multicriteria Optimization*, 2 ed. Springer, 2005.

[46] FARNSTROM, F., LEWIS, J., E ELKAN, C. Scalability for clustering algorithms revisited. *SIGKDD Explorations 2*, 1 (2000), 51–57.

[47] GALTON, F. Co-relations and their measurement, chiefly from anthropometric data. *Royal Society of London Proceedings Series I 45* (1888), 135–145.

[48] GESSFORD, J., E KARLIN, S. *Studies in the Mathematical Theory of Inventory and Production*. Stanford University Press, 1958, ch. Optimal policies for hydroelectric operations, pp. 179–200.

[49] GOLDBERG, D. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley Reading, MA, 1989.

[50] GOLUB, G. H., E LOAN, C. F. V. *Matrix Computations*, 3 ed. The Johns Hopkins University Press, 1996.

[51] GOMEZ, F., SCHMIDHUBER, J., E MIIKKULAINEN, R. Efficient non-linear control through neuroevolution. In *Proceedings of the 17th European Conference on Machine Learning* (2006), Springer.

[52] GOMEZ, F. J. *Robust non-linear control through neuroevolution*. PhD thesis, The University of Texas at Austin, 2003.

[53] GONZALEZ, E. F., E ZHANG, Y. Accelerating the Lee-Seung algorithm for non-negative matrix factorization. Relatório Técnico TR-05-02, Department of Computational and Applied Mathematics at Rice University, 2005.

[54] GORDON, G. Stable function approximation in dynamic programming. Relatório Técnico CMU-CS-95-103, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, January 1995.

[55] GORDON, G. J. Stable function approximation in dynamic programming. In *Proceedings of the International Conference on Machine Learning* (San Francisco, CA, 1995), Morgan Kaufmann, pp. 261–268.

[56] GOULD, S. J. *Darwin e os Grandes Enigmas da Vida*, 2 ed. Livraria Martins Fontes Editora LTDA, 1992. Tradução: Maria Elizabeth Martinez.

[57] GUESTRIN, C., HAUSKRECHT, M., E KVETON, B. Solving factored MDPs with continuous and discrete variables. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence* (2004).

[58] HARTIGAN, J. A. *Clustering Algorithms*. John Wiley and Sons, 1975.

[59] HARTIGAN, J. A., E WONG, M. A. A k-means clustering algorithm. *Applied Statistics 28* (1979), 100–108.

[60] HASTIE, T., TIBSHIRANI, R., E FRIEDMAN, J. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2002.

[61] HAYKIN, S. *Redes Neurais: Princípios e Prática*, 2 ed. Bookman, 2001. Tradução: Paulo Martins Engel.

[62] HO, N.-D., E VAN DOOREN, P. Non-negative matrix factorization with fixed row and column sums. *Linear Algebra and Its Applications* (2007).

[63] HOLLAND, J. H. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

[64] HOWARD, R. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA, 1960.

[65] HOWELL, A. J. *Automatic Face Recognition Using Radial Basis Function Networks*. PhD thesis, University of Sussex, 1997.

[66] HSU, F.-H. *Behind Deep Blue: Building the Computer that Defeated the World Chess Champion*. Princeton University Press, 2002.

[67] IGEL, C. Neuroevolution for reinforcement learning using evolution strategies. In *Proceedings of the Congress on Evolutionary Computation* (2003), vol. 4, IEEE Press, pp. 2588–2595.

[68] JAAKKOLA, T., JORDAN, M. I., E SINGH, S. P. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation 6* (1994).

[69] JONG, N., E STONE, P. Kernel-based models for reinforcement learning in continuous state spaces. In *Proceedings of the International Conference on Machine Learning—Workshop on Kernel Machines and Reinforcement Learning* (June 2006).

[70] KAELBLING, L. P. *Learning in embedded systems*. MIT Press, Cambridge, MA, USA, 1993.

[71] KAELBLING, L. P., LITTMAN, M. L., E MOORE, A. P. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research 4* (1996), 237–285.

[72] KAUFMAN, L., E ROUSSEEUW, P. J. *Finding Groups in Data: an Introduction to Cluster Analysis*. John Wiley and Sons, 1990.

[73] KEARNS, M., E SINGH, S. Near-optimal reinforcement learning in polynomial time. In *Proceedings of the International Conference on Machine Learning* (1998).

[74] KEARNS, M., SINGH, S., E SINGH, S. Finite-sample convergence rates for q-learning and indirect algorithms. In *Advances in Neural Information Processing Systems* (1999), MIT Press, pp. 996–1002.

[75] KEERTHI, S., E RAVINDRAN, B. A tutorial survey of reinforcement learning. *Sadhana 19*, 6 (1994), 851–889.

[76] KIM, D., SRA, S., E DHILLON, I. S. Fast Newton-type methods for the least squares nonnegative matrix approximation problem. In *Proceedings of SIAM Conference on Data Mining* (Minneapolis, Minnesota, USA, 2007).

[77] KLOPF, A. H. Brain function and adaptive systems - a heterostatic theory. In *Proceedings of the 1974 International Conference on Systems, Man and Cybernetics*, IEEE Systems.

[78] KLOPF, A. H. A comparison of natural and artificial intelligence. *SIGART Newsletter*, 53 (1975), 11–13.

[79] KLOPF, A. H. *The Hedonistic Neuron: a Theory of Memory, Learning and Intelligence*. Hemisphere, 1982.

[80] KOLMOGOROV, A. N., E FOMIN, S. V. *Elementos da Teoria das Funções e da Análise Funcional*. Mir, Moscou, 1982. Tradução: M. Dombrovsky.

[81] KULLBACK, S., E LEIBLER, R. A. On information and sufficiency. *Annals of Mathematical Statistics 22* (1951), 79–86.

[82] LAGOUDAKIS, M. G., E PARR, R. Least-squares policy iteration. *Journal of Machine Learning Research 4* (2003), 1107–1149.

[83] LAGOUDAKIS, M. G., PARR, R., E LITTMAN, M. L. Least-squares methods in reinforcement learning for control. In *Proceedings of the Second Hellenic Conference on AI* (London, UK, 2002), Springer-Verlag, pp. 249–260.

[84] LAPLACE, P. S. *Essai philosophique sur les probabilités*. Mme. Ve. Courcier., Paris, France, 1814. Disponível em inglês sob o título *A Philosophical Essay on Probabilities*.

[85] LARSON, H. J. *Introduction to probability theory and statistical inference*, 3 ed. John Wiley and Sons, New York, 1982.

[86] LEE, D. D., E SEUNG, H. S. Unsupervised learning by convex and conic coding. In *Advances in Neural Information Processing Systems* (1997), pp. 515–521.

[87] LEE, D. D., E SEUNG, H. S. Learning the parts of objects by non-negative matrix factorization. *Nature 401* (1999), 788–791.

[88] LEE, D. D., E SEUNG, H. S. Algorithms for nonnegative matrix factorization. In *Advances in Neural Information Processing Systems* (2000), pp. 556–562.

[89] LEITHOLD, L. *O Cálculo com Geometria Analítica, volumes I e II*, 3 ed. Harbra, 1994.

[90] LIN, C.-J. Projected gradient methods for nonnegative matrix factorization. *Neural Computation 19*, 10 (2007), 2756–2779.

[91] LIN, L.-J. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning 8* (August 1992), 293–321.

[92] LITTMAN, M. L., DEAN, T. L., E KAELBLING, L. P. On the complexity of solving Markov decision problems. In *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence* (Montreal, Québec, Canada, 1995), pp. 394–402.

[93] MANSOUR, Y., E SINGH, S. On the complexity of policy iteration. In *Proceedings of the Fifteenth Annual Conference on Uncertainty in Artificial Intelligence* (Stockholm, Sweden, 1999), pp. 401–408.

[94] MARKHAM, T. L. Factorizations of nonnegative matrices. *Proceedings of the American Mathematical Society 32*, 1 (1972), 45–47.

[95] MCLACHLAN, G. J., E BKASFORD, K. *Mixture Models: Inference and Applications to Clustering*. John Wiley and Sons, 1988.

[96] MCQUESTEN, P. H. *Cultural Enhancement of Neuroevolution*. PhD thesis, The University of Texas at Austin, 2002.

[97] MELEKOPOGLOU, M., E CONDON, A. On the complexity of the policy improvement algorithm for Markov decision processes. *ORSA Journal on Computing 6*, 2 (1994).

[98] MEYN, S., E TWEEDIE, R. *Markov chains and stochastic stability*. Springer-Verlag, 1993.

[99] MICHIE, D., E CHAMBERS, R. BOXES: An experiment on adaptivecontrol. *Machine Intelligence 2* (1968), 125–133.

[100] MILENOVA, B. L., E CAMPOS, M. M. O-cluster: Scalable clustering of large high dimensional data sets. *Second IEEE International Conference on Data Mining* (2002), 290–297.

[101] MINSKY, M. L. Steps toward artificial intelligence. In *Proceedings of the Institute of Radio Engineers* (1961), vol. 49, pp. 8–30. Publicado novamente no livro *Computers and Thought*, pp. 406–450, McGraw-Hill, 1963.

[102] MITCHELL, T. M. *Machine Learning*. McGraw-Hill, New York, 1997.

[103] MOORE, A., E ATKESON, C. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning 21* (1995).

[104] MOORE, A. W. Variable resolution dynamic programming: Efficiently learning action maps in multivariate real-valued spaces. In *Proceedings of the Eighth International Machine Learning Workshop* (San Francisco, CA, USA, 1991), Morgan Kaufmann.

[105] MOORE, A. W., E ATKESON, C. G. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning 13* (1993), 103–130.

[106] MORIARTY, D. E., E MIIKKULAINEN, R. Efficient reinforcement learning through symbiotic evolution. *Machine Learning 22*, 1–3 (1996), 11–32.

[107] MUNOS, R., E MOORE, A. Barycentric interpolators for continuous space & time reinforcement learning. In *Advances in Neural Information Processing Systems* (1999), MIT Press, pp. 1024–1030.

[108] NASCIMENTO, S., MIRKIN, B., E MOURA-PIRES, F. Modeling proportional membership in fuzzy clustering. *IEEE Transactions on Fuzzy Systems 11*, 2 (2003).

[109] NG, R. T., E HAN, J. Efficient and effective clustering methods for spatial data mining. In *Proceedings of the 20th International Conference on Very Large Data Bases* (Santiago del Chile, Chile, 1994), Morgan Kaufmann, pp. 144–155.

[110] ORMONEIT, D., E GLYNN, P. Kernel-based reinforcement learning in average-cost problems. *IEEE Transactions on Automatic Control 47*, 10 (October 2002), 1624–1636.

[111] ORMONEIT, D., E SEN, S. Kernel-based reinforcement learning. *Machine Learning 49 (2–3)* (2002), 161–178.

[112] PAATERO, P. Least squares formulation of robust, non-negative factor analysis. *Chemometrics and Intelligent Laboratory Systems 37* (1997), 23–35.

[113] PAATERO, P. The multilinear engine—a table-driven least-squares program for solving multilinear problems, including the n-way parallel factor analysis model. *Journal of Computational and Graphical Statistics 8*, 4 (1999), 854–888.

[114] PAATERO, P., E TAPPER, U. Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics 5* (1994), 111–126.

[115] PARSONS, L., HAQUE, E., E LIU, H. Subspace clustering for high dimensional data: a review. *SIGKDD Explorations 6*, 1 (2004), 90–105.

[116] PENG, J. *Efficient Dynamic Programming-Based Learning for Control*. PhD thesis, Northeastern University, Boston, MA, 1993.

[117] PENG, J., E WILLIAMS, R. J. Efficient learning and planning within the Dyna framework. *Adaptive Behavior 2* (1993), 437–454.

[118] PENG, J., E WILLIAMS, R. J. Incremental multi-step Q-learning. *Machine Learning 22* (1996), 283–290.

[119] PERKINS, T. J., E PRECUP, D. A convergent form of approximate policy iteration. In *Advances in Neural Information Processing Systems 15*. MIT Press, 2003, pp. 1595–1602.

[120] PICCI, G., VAN DEN HOF, J., E VAN SCHUPPEN, J. Primes in several classes of the positive matrices. *Linear Algebra and its Applications 277*, 1 (1998), 149–185.

[121] POGGIO, T., E GIROSI, F. A theory of networks for approximation and learning. Relatório Técnico AIM-1140 CBIP-31, Massachusetts Institute of Technology Artificial Intelligence Laboratory and Center for Biological Information Processing, Whitaker College, 1989.

[122] POLLACK, J. B., E BLAIR, A. D. Why did TD-Gammon work? In *Advances in Neural Information Processing Systems* (1996), pp. 10–16.

[123] POLLACK, J. B., BLAIR, A. D., E LAND, M. Coevolution of a backgammon player. In *Proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems* (Cambridge, MA, 1997), The MIT Press, pp. 92–98.

[124] POOLE, D., MACKWORTH, A., E GOEBEL, R. *Computational Intelligence—A Logical Approach*. Oxford University Press, 1998.

[125] POWELL, M. J. D. Radial basis functions for multivariable interpolation: a review. *Algorithms for Approximation* (1987), 143–167.

[126] PRECUP, D., SUTTON, R. S., E DASGUPTA, S. Off-policy temporal-difference learning with function approximation. In *Proceedings of the 18th International Conference on Machine Learning* (2001), Morgan Kaufmann, San Francisco, CA, pp. 417–424.

[127] PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., E FLANNERY, B. P. *Numerical Recipes in C, 2nd. edition*. Cambridge University Press, 1992.

[128] PUTERMAN, M. L. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. Wiley-Interscience, 1994.

[129] PUTERMAN, M. L., E SHIN, M. Modified policy iteration algorithms for discounted markov decision problems. *Management Science* (1978), 1127–1137.

[130] R. H. CANNON, J. *Dynamics of Physical Systems*. McGraw-Hill, 1967.

[131] REYNOLDS, S. I. The stability of general discounted reinforcement learning with linear function approximation. In *Proceedings of the U.K. Workshop on Computational Intelligence* (2002).

[132] RIBEIRO, C. Reinforcement learning agents. *Artificial Intelligence Review 17*, 3 (2002), 223–250.

[133] ROSS, S. M. *Introduction to Stochastic Dynamic Programming*. Academic Press, Inc., 1983.

[134] RUMMERY, G., E NIRANJAN, M. On-line Q-learning using connectionist systems. Relatório Técnico CUED/F-INFENG/TR 166, Cambridge University—Engineering Department, 1994.

[135] RUMMERY, G. A. *Problem Solving with Reinforcement Learning*. PhD thesis, Cambridge University, 1995.

[136] RUST, J. Using randomization to break the curse of dimensionality. *Econometrica 65*, 3 (1997), 487–516.

[137] SABES, P. N. Approximating Q-values with basis function representations. In *Proceedings of the Connectionist Models Summer School* (Hillsdale, NJ, 1993), Lawrence Erlbaum Assoc. Inc.

[138] SACKS, O. *O Homem que Confundiu sua Mulher com um Chapéu*. Imago Editora LTDA, 1987. Tradução de Talita Macedo Rodrigues.

[139] SAMUEL, A. L. Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development 3* (1959), 211–229.

[140] SAMUEL, A. L. Some studies in machine learning using the game of checkers II—recent advances. *IBM Journal on Research and Development 11* (1967), 601–617.

[141] SARAVANAN, N., E FOGEL, D. B. Evolving neural control systems. *IEEE Expert: Intelligent Systems and Their Applications 10*, 3 (1995), 23–27.

[142] SCARDUA, L. A., DA CRUZ, J. J., E REALI. Optimal control of ship unloaders using reinforcement learning. *Advanced Engineering Informatics 16*, 3 (July 2002), 217–227.

[143] SCHOKNECHT, R., E MERKE, A. Convergent combinations of reinforcement learning with linear function approximation. In *Advances in Neural Information Processing Systems 15*. MIT Press, 2003, pp. 1579–1586.

[144] SCHÖLKOPF, B., E SMOLA, A. *Learning with Kernels*. MIT Press, 2002.

[145] SEIDEL, R. Constructing higher-dimensional convex hulls at logarithmic cost per face. In *Proceedings of the eighteenth annual ACM symposium on theory of computing* (New York, NY, USA, 1986), ACM, pp. 404–413.

[146] SHANNON, C. Programming a computer for playing chess. *Philosophical Magazine 41*, 314 (1950).

[147] SI, J., BARTO, A. G., POWELL, W. B., E WUNSCH, D. *Handbook of Learning and Approximate Dynamic Programming*. Wiley–IEEE Press, 2004.

[148] SINGH, S. P., E BERTSEKAS, D. Reinforcement learning for dynamic channel allocation in cellular telephone systems. In *Advances in Neural Information Processing Systems* (1997), vol. 9, The MIT Press, p. 974.

[149] SINGH, S. P., JAAKKOLA, T., E JORDAN, M. I. Reinforcement learning with soft state aggregation. In *Advances in Neural Information Processing Systems* (1995), vol. 7, The MIT Press, pp. 361–368.

[150] SINGH, S. P., JAAKKOLA, T., LITTMAN, M. L., E SZEPESVARI, C. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning 38*, 3 (2000), 287–308.

[151] SINGH, S. P., E SUTTON, R. S. Reinforcement learning with replacing eligibility traces. *Machine Learning 22*, 1–3 (1996), 123–158.

[152] SINGH, S. P., E YEE, R. C. An upper bound on the loss from approximate optimal-value functions. *Machine Learning 16*, 3 (1994), 227–233.

[153] SRA, S., E DHILLON, I. S. Nonnegative matrix approximation: Algorithms and applications. Relatório Técnico TR-06-27, Department of Computer Sciences, University of Texas at Austin, Austin, TX 78712, USA, 2006.

[154] STANLEY, K. O., E MIIKKULAINEN, R. Efficient reinforcement learning through evolving neural network topologies. In *Proceedings of the Genetic and Evolutionary Computation Conference* (New York, July 2002), Morgan Kaufmann Publishers, pp. 569–577.

[155] STONE, P., E SUTTON, R. S. Scaling reinforcement learning toward RoboCup soccer. In *Proceedings of the 18th International Conference on Machine Learning* (2001), Morgan Kaufmann, San Francisco, CA, pp. 537–544.

[156] SUTTON, R., MCALLESTER, D., SINGH, S., E MANSOUR, Y. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems* (2000), pp. 1057–1063.

[157] SUTTON, R. S. Learning to predict by the methods of temporal differences. *Machine Learning 3* (1988), 9–44.

[158] SUTTON, R. S. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the International Conference on Machine Learning* (1990), pp. 216–224.

[159] SUTTON, R. S. Planning by incremental dynamic programming. In *Proceedings of the Eighth International Workshop on Machine Learning* (1991), Morgan Kaufmann, pp. 353–357.

[160] SUTTON, R. S. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems* (1996), vol. 8, The MIT Press, pp. 1038–1044.

[161] SUTTON, R. S., E BARTO, A. G. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[162] TADIĆ, V. On the convergence of temporal-difference learning with linear function approximation. *Machine Learning 42*, 3 (2001), 241–267.

[163] TESAURO, G. J. Practical issues in temporal difference learning. In *Advances in Neural Information Processing Systems* (1992), vol. 4, Morgan Kaufmann Publishers, Inc., pp. 259–266.

[164] TESAURO, G. J. TD-Gammon, a self-teaching backgammon program achieves master-level play. *Neural Computation 6*, 2 (1994), 215–219.

[165] TESAURO, G. J. Temporal difference learning and TD-gammon. *Communications of the ACM* (1995).

[166] THRUN, S., E SCHWARTZ, A. Issues in using function approximation for reinforcement learning. In *Proceedings of the Fourth Connectionist Models Summer School* (December 1993), Lawrence Erlbaum Associates.

[167] THRUN, S. B. The role of exploration in learning control with neural networks. In *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches* (Florence, Kentucky, 1992), Van Nostrand Reinhold.

[168] TSITSIKLIS, J. N. Asynchronous stochastic approximation and Q-learning. *Machine Learning 16* (1994), 185–202.

[169] TSITSIKLIS, J. N., E ROY, B. V. Feature-based methods for large scale dynamic programming. *Machine Learning 22* (1996), 59–94.

[170] TSITSIKLIS, J. N., E ROY, B. V. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control 42* (May 1997), 674–690.

[171] WALD, A. *Sequential Analysis*. Wiley, New York, 1974.

[172] WATKINS, C. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, England, 1989.

[173] WATKINS, C., E DAYAN, P. Q-learning. *Machine Learning 8* (1992), 279–292.

[174] WERBOS, P. *Handbook of intelligent control*. Van Nostrand Reinhold, 1992, ch. Approximate dynamic programming for real-time control and neural modeling.

[175] WERBOS, P. J. Advanced forecasting methods for global crisis warning and models of intelligence. *General systems yearbook 22* (1977), 25–38.

[176] WERBOS, P. J. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks 1*, 4 (1988), 339–356.

[177] WHITE, D. J. Real applications of Markov decision processes. *Interfaces 15* (1985), 73–83.

[178] WHITE, D. J. Further real applications of Markov decision processes. *Interfaces 18* (1988), 55–61.

[179] WHITLEY, D., DOMINIC, S., DAS, R., E ANDERSON, C. W. Genetic reinforcement learning for neurocontrol problems. *Machine Learning 13*, 2-3 (1993), 259–284.

[180] WHITLEY, D., RICHARDS, M., BEVERIDGE, R., E BARRETO, A. M. S. Alternative evolutionary algorithms for evolving programs: Evolution strategies and steady-state GP. In *Proceedings of the 8^{th} Annual Conference on Genetic and Evolutionary Computation* (Seattle, Washington, USA, July 2006), vol. 1, ACM Press, pp. 919–926. Winner best GP paper.

[181] WHITT, W. Approximations of dynamic programs, I. *Mathematics of Operations Research 3*, 3 (August 1978), 231–243.

[182] WHITT, W. Approximations of dynamic programs, II. *Mathematics of Operations Research 4*, 2 (May 1979), 179–185.

[183] WIDROW, B., E SMITH, F. W. Pattern-recognizing control systems. In *Proceedings of the Computer and Information Sciences Symposium* (Washington, D.C., USA, 1963).

[184] WIELAND, A. P. Evolving neural network controllers for unstable systems. In *Proceedings of the International Joint Conference on Neural Networks* (Seattle, WA, USA, 8–14 July 1991), vol. 2, pp. 667–673.

[185] WILLIAMS, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning 8* (1992), 229–256.

[186] WILLIAMS, R. J., E BAIRD, L. C. Tight performance bounds on greedy policies based on imperfect value functions. Relatório Técnico NU-CCS-93-14, Northeastern University, November 1993.

[187] WITTEN, I. H. The apparent conflict between estimation and control—a survey of the two-armed bandit problem. *Journal of Franklin Institute* (1976), 161–189.

[188] WITTEN, I. H. Exploring, modelling and controlling discrete sequential environments. *International Journal of Man-Machine Studies 9* (1977), 715–735.

[189] XU, W., LIU, X., E GONG, Y. Document clustering based on non-negative matrix factorization. In *Proceedings of the 26th annual international ACM SIGIR conference on research and development in informaion retrieval* (New York, NY, USA, 2003), ACM, pp. 267–273.

[190] XU, X., GEN HE, H., E HU, D. Efficient reinforcement learning using recursive least-squares methods. *Journal of Artificial Intelligence Research 16* (2002), 259–292.

[191] ZADEH, L. A. Fuzzy sets. *Information and Control 8*, 3 (1965), 338–353.

[192] ZHANG, W., E DIETTERICH, T. G. A reinforcement learning approach to job-shop scheduling. In *Proceedings of the International Joint Conference on Artificial Intelligence* (1995).

Apêndice A

Convenções adotadas

Neste apêndice eu descrevo brevemente algumas convenções usadas no trabalho.

Discurso: Normalmente em textos científicos opta-se por um discurso impessoal, com o uso da terceira pessoa, ou por uma autoria coletiva, quando se adota a primeira pessoa do plural. Como uma tese de doutorado é, por definição, um trabalho científico de caráter mais pessoal, acredito que nesse caso o uso da primeira pessoa do singular também seja admissível. Penso inclusive que esse tipo de discurso oferece algumas vantagens, como por exemplo a possibilidade de diferenciar claramente hipóteses bem estabelecidas de opiniões e análises pessoais do autor. Ironicamente, este parágrafo é um bom exemplo nesse sentido. Cumpre enfatizar que a escolha por essa forma de exposição não significa, de maneira alguma, subestimar a enorme contribuição daqueles que estiveram envolvidos direta ou indiretamente na pesquisa que culminou neste trabalho.

Língua estrangeira: Procurei evitar estrangeirismos ao máximo. Em algumas situações, porém, o uso de expressões em língua estrangeira me pareceu inevitável. Um exemplo são os casos em que não há um termo específico em português que traduza uma idéia de maneira precisa. Nessas situações, o uso exclusivo do português prejudicaria o fluxo do texto. Além disso, optei por conservar as siglas em inglês, a fim de facilitar a correspondência com a literatura internacional. Quando uma nova sigla é introduzida, a expressão que lhe deu origem é apresentada em uma nota de rodapé

identificada pelo símbolo "†". Os significados das siglas podem ser consultados sempre que desejado no Índice Remissivo no final do trabalho (página 239).

Matemática: Em relação às expressões matemáticas, adotei as convenções normalmente utilizadas na literatura. Talvez a única observação que se faça necessária diz respeito ao uso de letras em negrito para diferenciar matrizes e vetores de valores escalares. Especificamente, as matrizes são identificadas por letras maiúsculas do alfabeto latino e os vetores por letras minúsculas do mesmo alfabeto. A não ser quando especificado o contrário, os vetores correspondem às linhas das matrizes. Assim, dada a matriz $\mathbf{P}^{\pi}$, o vetor $\mathbf{p}_i^{\pi}$ é a i-ésima linha dessa matriz, e p_{ij}^{π} corresponde ao j-ésimo elemento desse vetor. Os principais símbolos utilizados nesta tese podem ser encontrados na Lista de Símbolos constante na página xviii.

Apêndice B

Simulador do Problema de Equilibrar Bastões

O simulador do problema de equilibrar bastões envolve uma séria de variáveis. A Tabela B.1 traz uma descrição de cada uma delas, bem como os respectivos valores usados nos experimentos da Seção 4.4. Esses valores foram extraídos da tese de Gomez [52], e correspondem à configuração "padrão" do problema.

Nos experimentos com dois bastões a ação $F = 0\mathrm{N}$ foi excluída do problema, pelos motivos discutidos no texto. A partir das variáveis mostradas na Tabela B.1 pode-se definir as equações de movimento para b bastões equilibrados simultaneamente [184, 52]:

$$\ddot{x} = \frac{F - \mu_c \mathrm{sgn}(\dot{x}) + \sum_{i=1}^{b} \hat{F}_i}{M + \sum_{i=1}^{b} \hat{m}_i}, \tag{B.1}$$

$$\ddot{\theta}_i = -\frac{3}{2l_i}\left(\ddot{x}\cos\theta_i + g\mathrm{sen}\theta_i + \frac{2\mu_{pi}\dot{\theta}_i}{m_i l_i}\right), \tag{B.2}$$

onde $\mathrm{sgn}(x)$ retorna o sinal de x ou zero caso essa variável seja nula. Note que adotei a notação "$\dot{x}$" para representar a taxa de variação de x. A variável $\hat{F}_i$ na equação (B.1) é a força efetivamente aplicada no carro pelo i-ésimo bastão, dada por:

$$\hat{F}_i = \frac{1}{2} m_i l_i \dot{\theta}_i^2 \sin\theta_i + \frac{3}{4} m_i \cos\theta_i \left(\frac{2\mu_{pi}\dot{\theta}_i}{m_i l_i} + g\mathrm{sen}\theta_i\right).$$

Símbolo	Descrição	Valor
x	Posição do carro na pista	$[-2.4, 2.4]\mathrm{m}$
θ	Ângulo entre o primeiro bastão e o carro	$[-\frac{\pi}{5}, \frac{\pi}{5}]\mathrm{rad}$
θ_2	Ângulo entre o segundo bastão e o carro	$[-\frac{\pi}{5}, \frac{\pi}{5}]\mathrm{rad}$
F	Força aplicada pelo agente no carro	$\{-10, 0, 10\}\mathrm{N}$
l_1	Comprimento do primeiro bastão	$1\mathrm{m}$
l_2	Comprimento do segundo bastão	$0.1\mathrm{m}$
M	Massa do carro	$1\mathrm{kg}$
m_1	Massa do primeiro bastão	$0.1\mathrm{kg}$
m_2	Massa do segundo bastão	$0.01\mathrm{kg}$
μ_c	Coeficiente de fricção do carro na pista	5×10^{-4}
μ_{p1}	Coeficiente de fricção da polia do primeiro bastão	2×10^{-6}
μ_{p2}	Coeficiente de fricção da polia do segundo bastão	2×10^{-6}
g	Aceleração devido à gravidade	$9.8\mathrm{m/s}$

Tabela B.1: Parâmetros usados no problema de equilibrar bastões.

De forma similar, $\hat{m}_i$ representa a massa efetiva do i-ésimo bastão:

$$\hat{m}_i = m_i \left(1 - \frac{3}{4}\cos^2\theta_i\right).$$

A integração das equações (B.1) e (B.2) foi feita pelo método de Runge-Kutta de quarta ordem com intervalo de $0.02\mathrm{s}$.

Apêndice C

Experimentos Computacionais Extras

Este apêndice traz o resultado de uma série de experimentos preliminares usados para determinar a largura dos *kernels* usados pelos algoritmos no problema de equilibrar simultaneamente dois bastões (Seção 4.4.2). Em todos os casos, a largura τ_q foi definida a partir dos valores de ϖ_q mostrados na tabela, como explicado no texto. Sete vizinhos foram usados como referência para determinar a sobreposição das funções. As Tabelas C.1 a C.4 mostram os resultados utilizados para determinar a largura do *kernel* usado pelo LSPI nos experimentos em que as transições foram coletadas por uma política de exploração π_e. As Tabelas C.5 e C.6 trazem os resultados usados para definir a largura dos *kernels* nos experimentos em que as transições foram amostradas uniformemente no espaço de estados X.

Algoritmo	ϖ_q	**Ep.**(%)	**Média**	**Máx.**	**Mín.**	**DP**	**Exec.**(%)
LSPI(17319,50)	0.01	0.00	4 9.06	448	5	74.08	0
LSPI(17319,50)	0.10	0.00	61.34	308	5	53.10	0
LSPI(17319,50)	0.20	6.17	224.64	3000	5	715.27	0
LSPI(17319,50)	0.30	21.73	752.71	3000	5	1237.32	0
LSPI(17319,50)	0.40	7.90	304.54	3000	5	811.86	0
LSPI(17319,50)	0.50	0.00	19.48	118	5	19.25	0
LSPI(17319,50)	0.60	0.74	81.27	3000	5	354.07	0
LSPI(17319,50)	0.70	0.00	30.64	287	5	40.84	0
LSPI(17319,50)	0.80	11.11	531.52	3000	5	979.77	0
LSPI(17319,50)	0.90	40.00	1562.37	3000	27	1323.84	20
LSPI(17319,50)	0.95	0.25	218.46	3000	12	293.42	0
LSPI(17319,50)	0.99	0.00	6.59	35	5	4.44	0

Tabela C.1: Resultados obtidos pelo algoritmo LSPI(17319,50) no problema de equilibrar dois bastões simultaneamente usando diferentes larguras para o *kernel* gaussiano. Os valores correspondem ao número médio de passos executados no conjunto de teste pelas políticas de decisão retornadas em 5 execuções independentes do algoritmo.

Algoritmo	ϖ_q	**Ep.**(%)	**Média**	**Máx.**	**Mín.**	**DP**	**Exec.**(%)
LSPI(17319,100)	0.01	0.00	18.41	1046	5	56.10	0
LSPI(17319,100)	0.10	0.74	118.27	3000	5	336.34	0
LSPI(17319,100)	0.20	0.00	22.75	168	5	24.49	0
LSPI(17319,100)	0.30	0.00	27.74	441	5	44.70	0
LSPI(17319,100)	0.40	0.00	24.60	908	5	54.46	0
LSPI(17319,100)	0.50	0.00	20.97	93	5	19.32	0
LSPI(17319,100)	0.60	0.00	36.90	2341	5	127.28	0
LSPI(17319,100)	0.70	0.00	60.56	2543	5	180.60	0
LSPI(17319,100)	0.80	0.25	125.93	3000	5	282.37	0
LSPI(17319,100)	0.90	0.00	128.14	402	9	73.80	0
LSPI(17319,100)	0.95	19.75	714.97	3000	5	1144.89	0
LSPI(17319,100)	0.99	0.00	16.23	102	5	28.36	0

Tabela C.2: Resultados obtidos pelo algoritmo LSPI(17319,100) no problema de equilibrar dois bastões simultaneamente usando diferentes larguras para o *kernel* gaussiano. Os valores correspondem ao número médio de passos executados no conjunto de teste pelas políticas de decisão retornadas em 5 execuções independentes do algoritmo.

Algoritmo	ϖ_q	**Ep.**(%)	**Média**	**Máx.**	**Mín.**	**DP**	**Exec.**(%)
LSPI(17319,150)	0.01	0.00	51.37	325	5	73.22	0
LSPI(17319,150)	0.10	0.00	48.39	703	5	70.62	0
LSPI(17319,150)	0.20	0.00	27.28	216	5	30.93	0
LSPI(17319,150)	0.30	2.47	97.22	3000	5	463.19	0
LSPI(17319,150)	0.40	0.74	66.62	3000	5	318.02	0
LSPI(17319,150)	0.50	6.67	224.02	3000	5	743.45	0
LSPI(17319,150)	0.60	0.00	18.03	95	5	19.87	0
LSPI(17319,150)	0.70	0.00	41.13	144	5	32.71	0
LSPI(17319,150)	0.80	0.00	42.77	213	5	45.26	0
LSPI(17319,150)	0.90	0.00	50.20	365	5	64.25	0
LSPI(17319,150)	0.95	18.77	885.64	3000	72	1104.88	0
LSPI(17319,150)	0.99	0.00	5.35	6	5	0.48	0

Tabela C.3: Resultados obtidos pelo algoritmo LSPI(17319,150) no problema de equilibrar dois bastões simultaneamente usando diferentes larguras para o *kernel* gaussiano. Os valores correspondem ao número médio de passos executados no conjunto de teste pelas políticas de decisão retornadas em 5 execuções independentes do algoritmo.

Algoritmo	ϖ_q	**Ep.**(%)	**Média**	**Máx.**	**Mín.**	**DP**	**Exec.**(%)
LSPI(17319,200)	0.01	0.00	19.68	187	5	24.05	0
LSPI(17319,200)	0.10	2.47	100.19	3000	5	463.65	0
LSPI(17319,200)	0.20	0.00	42.42	505	5	55.54	0
LSPI(17319,200)	0.30	0.00	17.85	161	5	18.06	0
LSPI(17319,200)	0.40	0.00	28.93	778	5	51.46	0
LSPI(17319,200)	0.50	0.00	20.06	201	5	25.87	0
LSPI(17319,200)	0.60	0.00	40.00	417	5	59.25	0
LSPI(17319,200)	0.70	0.00	37.72	332	5	41.71	0
LSPI(17319,200)	0.80	0.25	46.21	3000	5	154.57	0
LSPI(17319,200)	0.90	0.00	36.75	146	5	35.23	0
LSPI(17319,200)	0.95	0.00	97.15	333	5	74.03	0
LSPI(17319,200)	0.99	0.00	15.49	154	5	21.47	0

Tabela C.4: Resultados obtidos pelo algoritmo LSPI(17319,200) no problema de equilibrar dois bastões simultaneamente usando diferentes larguras para o *kernel* gaussiano. Os valores correspondem ao número médio de passos executados no conjunto de teste pelas políticas de decisão retornadas em 5 execuções independentes do algoritmo.

Algoritmo	ϖ_q	**Ep.**(%)	**Média**	**Máx.**	**Mín.**	**DP**	**Exec.**(%)
LSPI(20000,100)	0.01	0.00	12.94	40	5	9.15	0
LSPI(20000,100)	0.10	0.00	15.19	115	5	12.88	0
LSPI(20000,100)	0.20	0.00	16.20	90	5	14.21	0
LSPI(20000,100)	0.30	0.00	20.11	96	5	14.12	0
LSPI(20000,100)	0.40	0.00	11.80	85	5	9.91	0
LSPI(20000,100)	0.50	0.00	21.06	98	5	16.44	0
LSPI(20000,100)	0.60	0.00	17.41	96	5	16.50	0
LSPI(20000,100)	0.70	0.00	39.42	337	5	34.66	0
LSPI(20000,100)	0.80	0.00	35.91	570	7	50.28	0
LSPI(20000,100)	0.90	0.00	9.38	32	5	8.35	0
LSPI(20000,100)	0.90	0.00	9.38	32	5	8.35	0
LSPI(20000,100)	0.95	1.98	160.59	3000	6	480.47	0
LSPI(20000,100)	0.99	0.25	114.20	3000	5	155.09	0

Tabela C.5: Resultados obtidos pelo algoritmo LSPI(20000,100) no problema de equilibrar dois bastões usando transições amostradas uniformemente no espaço de estados. Os valores correspondem ao número médio de passos executados no conjunto de teste pelas políticas de decisão retornadas em 5 execuções independentes do algoritmo.

Algoritmo	ϖ_q	**Ep.**(%)	**Média**	**Máx.**	**Mín.**	**DP**	**Exec.**(%)
KBSF(20000,100)	0.01	0.00	11.54	171	5	16.03	0
KBSF(20000,100)	0.10	0.00	9.59	43	5	7.29	0
KBSF(20000,100)	0.20	0.00	24.48	1605	5	87.39	0
KBSF(20000,100)	0.30	0.00	9.49	111	5	9.57	0
KBSF(20000,100)	0.40	0.00	15.52	134	5	16.19	0
KBSF(20000,100)	0.50	0.00	21.73	267	5	35.67	0
KBSF(20000,100)	0.60	0.74	83.18	3000	5	307.05	0
KBSF(20000,100)	0.70	0.25	75.57	3000	5	158.11	0
KBSF(20000,100)	0.80	0.00	130.02	1048	74	59.09	0
KBSF(20000,100)	0.90	0.00	102.37	152	66	20.00	0
KBSF(20000,100)	0.90	0.00	102.37	152	66	20.00	0
KBSF(20000,100)	0.95	0.00	87.32	352	41	34.49	0
KBSF(20000,100)	0.99	0.00	34.81	158	9	42.89	0

Tabela C.6: Resultados obtidos pelo algoritmo KBSF(20000,100) no problema de equilibrar dois bastões usando transições amostradas uniformemente no espaço de estados. Os valores correspondem ao número médio de passos executados no conjunto de teste pelas políticas de decisão retornadas em 5 execuções independentes do algoritmo. O valor de τ^a foi calculado com $\varpi_a = 0.3$.

Índice Remissivo

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo