UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

BRUNO CASTRO DA SILVA

Aprendizado por Reforço em Ambientes Não-Estacionários

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof[a]. Dra. Ana Lúcia Cetertich Bazzan
Orientadora

Porto Alegre, maio de 2007

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Silva, Bruno Castro da

Aprendizado por Reforço em Ambientes Não-Estacionários / Bruno Castro da Silva. – Porto Alegre: PPGC da UFRGS, 2007.

98 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2007. Orientadora: Ana Lúcia Cetertich Bazzan.

1. Não-estacionariedade. 2. Aprendizado por reforço. 3. Múltiplos modelos. I. Bazzan, Ana Lúcia Cetertich. II. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. José Carlos Ferraz Hennemann
Vice-Reitor: Prof. Pedro Cezar Dutra Fonseca
Pró-Reitora de Pós-Graduação: Profª. Valquíria Linck Bassani
Diretor do Instituto de Informática: Prof. Flávio Rech Wagner
Coordenadora do PPGC: Profª. Luciana Nedel Porcher
Bibliotecária-chefe do Instituto de Informática: Beatriz Regina Bastos Haro

Because things are the way they are, things will not stay the way they are.
— BERTOLT BRECHT

AGRADECIMENTOS

Agradeço em primeiro lugar à professora Ana Lúcia Bazzan, pela participação essencial em minha formação como bacharel e pelo apoio incondicional, desde o ano de 2002, como orientadora. Não seria exagero dizer que meu amadurecimento como pesquisador deveu-se exclusivamente ao seu direcionamento pragmático e preciso, ao seu constante empenho e ao excelente modelo de profissionalismo que representou em minha vida. Não imagino como poderia ter encontrado uma orientadora melhor. Agora, ao fim desta etapa, vejo que também tenho uma amiga. Muito obrigado!

Agradeço em especial ao meu colega e amigo Eduardo Basso, sem o qual grande parte dessa dissertação não teria sido possível. A todos outros colegas da UFRGS com os quais tive conversas esclarecedoras e que me ajudaram de milhares de formas, especialmente Filipo Perotto, Renata Palazzo, Denise de Oliveira, Juliano Bittencourt e Licurgo de Almeida. Ao professor Paulo Engel, a quem devo meu primeiro contato acadêmico com a inteligência artificial, no ano de 2001, e depois novamente nas disciplinas do mestrado. Ao professor Roberto Silva, com quem mantive contato apenas no final de meu mestrado, mas que contribuiu, indiretamente, muito mais do que imagina.

A todo pessoal do Instituto de Informática, por fazer o nosso dia-a-dia possível e por ter conseguido construir, muitas vezes por meio de sacrifícios pessoais, um dos centros de computação mais respeitados do Brasil. Só os que convivem com vocês há tantos anos podem entender a beleza do exemplo que vocês dão para o resto da sociedade. Ao CNPq, por ter possibilitado que eu me dedicasse à pesquisa por tantos anos.

A todos meus amigos, pelas noites em claro e pelas praias no inverno: Afonso Araujo, Rafaello Perotto e Saul Farias, entre outros tantos.

Aos meus pais e irmãs, por estarem sempre comigo, incondicionalmente. À Sofia, que sem dúvida terá um futuro brilhante pela frente, por existir e por deixar meus dias mais alegres. E por fim, à mulher que me acompanha desde 2001 e com quem compartilhei todas as grandes mudanças da minha vida. Ana, nosso futuro começa agora. Te amo.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Neste trabalho apresentamos o RL-CD (*Reinforcement Learning with Context Detection*), um método desenvolvido a fim de lidar com o problema do aprendizado por reforço (RL) em ambientes não-estacionários. Embora os métodos existentes de RL consigam, muitas vezes, superar a não-estacionariedade, o fazem sob o inconveniente de terem de reaprender políticas que já haviam sido calculadas, o que implica perda de desempenho durante os períodos de readaptação. O método proposto baseia-se em um mecanismo geral através do qual são criados, atualizados e selecionados um dentre vários modelos e políticas parciais. Os modelos parciais do ambiente são incrementalmente construídos de acordo com a capacidade do sistema de fazer predições eficazes. A determinação de tal medida de eficácia baseia-se no cálculo de qualidades globais para cada modelo, as quais refletem o ajuste total necessário para tornar cada modelo coerente com as experimentações reais. Depois de apresentadas as bases teóricas necessárias para fundamentar o RL-CD e suas equações, são propostos e discutidos um conjunto de experimentos que demonstram sua eficiência, tanto em relação a estratégias clássicas de RL quanto em comparação a algoritmos especialmente projetados para lidar com cenários não-estacionários. O RL-CD é comparado com métodos reconhecidos na área de aprendizado por reforço e também com estratégias RL multi-modelo. Os resultados obtidos sugerem que o RL-CD constitui uma abordagem eficiente para lidar com uma subclasse de ambientes não-estacionários, especificamente aquela formada por ambientes cuja dinâmica é corretamente representada por um conjunto finito de Modelos de Markov estacionários. Por fim, apresentamos a análise teórica de um dos parâmetros mais importantes do RL-CD, possibilitada pela aproximação empírica de distribuições de probabilidades via métodos de Monte Carlo. Essa análise permite que os valores ideais de tal parâmetro sejam calculados, tornando assim seu ajuste independente da aplicação específica sendo estudada.

Palavras-chave: Não-estacionariedade, aprendizado por reforço, múltiplos modelos.

Reinforcement Learning in Non-Stationary Environments

ABSTRACT

In this work we introduce RL-CD (*Reinforcement Learning with Context Detection*), a novel method for solving reinforcement learning (RL) problems in non-stationary environments. In face of non-stationary scenarios, standard RL methods need to continually readapt themselves to the changing dynamics of the environment. This causes a performance drop during the readjustment phase and implies the need for relearning policies even for dynamics which have already been experienced. RL-CD overcomes these problems by implementing a mechanism for creating, updating and selecting one among several partial models of the environment. The partial models are incrementally built according to the system's capability of making predictions regarding a given sequence of observations. First, we present the motivations and the theorical basis needed to develop the conceptual framework of RL-CD. Afterwards, we propose, formalize and show the efficiency of RL-CD both in a simple non-stationary environment and in a noisy scenarios. We show that RL-CD performs better than two standard reinforcement learning algorithms and that it has advantages over methods specifically designed to cope with non-stationarity. Finally, we present the theoretical examination of one of RL-CD's most important parameters, made possible by means of the analysis of probability distributions obtained via Monte Carlo methods. This analysis makes it possible for us to calculate the optimum values for this parameter, so that its adjustment can be performed independently of the scenario being studied.

Keywords: Non-stationarity, reinforcement learning, multi-model learning.

1 INTRODUÇÃO

O termo *aprendizado*, em inteligência artificial (IA), é dotado de inúmeros significados. No contexto das técnicas não-supervisionadas, por exemplo, o aprendizado se refere à descoberta de protótipos que caracterizam agrupamentos de dados com semelhanças estruturais; já no contexto de técnicas supervisionadas, o aprendizado consiste na descoberta de parâmetros capazes de generalizar exemplos em tarefas de classificação. A terceira grande metodologia de aprendizado proposta pela IA se chama Aprendizado por Reforço (RL, do inglês *Reinforcement Learning*). Nas técnicas de RL, ao contrário das anteriores, o aprendizado ocupa-se primariamente em determinar *maneiras de agir*, tendo como base a interação repetida com um ambiente.

No âmbito do aprendizado por reforço, vários são os problemas que impedem o bom funcionamento dos algoritmos de treinamento, tais como o crescimento exponencial do número de estados e a dificuldade de aprendizado de políticas hierárquicas. Nesta dissertação, iremos abordar especificamente o problema de **como aprender a agir em ambientes não-estacionários**.

A expressão *ambiente não-estacionário* diz respeito a todo ambiente cujas regras de funcionamento são modificadas ao longo do tempo. Podemos imaginar, como exemplos, salas cujas portas estão ora trancadas e ora destrancadas, presas que fogem de maneiras imprevisíveis de seus predadores, ou então ambientes em que as características de atrito do chão são modificadas, por razões desconhecidas, ao longo do tempo.

De forma geral, é razoável supor que o ambiente em que um agente se encontra pode ser descrito por algum tipo de modelo de previsão, ou seja, por um modelo matemático que descreve probabilisticamente o resultado de uma ação quando tomada em determinado estado. Se, por alguma razão não diretamente perceptível ao agente, as probabilidades estimadas por tais modelos forem modificadas ao longo do tempo, dizemos se tratar de um ambiente não-estacionário. As causas da não-estacionariedade em geral estão relacionadas ao conhecimento imperfeito do agente; sempre que este não possuir sensores adequados para a percepção de todos atributos relevantes do ambiente, por exemplo, ou for incapaz de observar diretamente a variação de algum importante atributo de estado, acabará por perceber seu ambiente como sendo não-estacionário. Outras causas possíveis para a não-estacionariedade estão relacionadas com uso de modelos aproximados do ambiente – uma vez que estes estão constantemente sendo atualizados e aprimorados, e também com o aprendizado em ambientes com múltiplos agentes, nos quais cada aprendiz não necessariamente modela de forma completa todos os demais agentes. Finalmente, a não-estacionariedade pode ocorrer caso o projetista decida alterar a tarefa a ser resolvida pelo agente, mesmo que o agente em si continue inserido no mesmo ambiente. Nesse caso, a menos que o aprendiz receba uma sinalização explícita de que uma nova tarefa deve ser resolvida, sua percepção das recompensas recebidas em cada situação irá

se alterar com o tempo.

Infelizmente, as técnicas clássicas de aprendizado por reforço não são bem adaptadas a este tipo de ambiente. Em geral ocorre que as políticas de ação calculadas precisam ser reaprendidas a cada vez que a dinâmica do mundo muda. Além disso, mesmo quando as regras do ambiente retornam a algum patamar já experimentado, ainda assim o agente precisa reaprender políticas que haviam sido anteriormente descobertas. Também é importante notar que a grande maioria das provas de convergência para os algoritmos clássicos de RL assume ambientes estacionários (KAELBLING; LITTMAN; MOORE, 1996).

O objetivo desta dissertação, em linhas gerais, é propor um método capaz de sanar alguns dos problemas enfrentados durante o aprendizado em ambientes não-estacionários. O foco deste trabalho é tratar uma subclasse de ambientes não-estacionários, formada especificamente por ambientes cuja dinâmica pode ser eficientemente descrita através de um conjunto finito de regimes estacionários. A cada momento, apenas um destes regimes estacionários de operação está ativo. Entretanto, os regimes se alternam temporalmente de uma maneira imprevisível ao agente, o que faz com que não seja possível determinar *a priori* qual é a dinâmica atual do ambiente. Assim, caberá ao agente aprendiz modelar cada um dos regimes, detectar quando ocorrem as trocas na dinâmica do ambiente, e decidir quando novos modelos precisam ser criados a fim de representar dinâmicas ainda não experimentadas.

O método a ser proposto é fortemente baseado no cálculo da *qualidade de previsão* para cada um dos modelos construídos. Tal medida de qualidade reflete o ajuste total necessário para tornar cada modelo coerente com as experimentações reais do agente. A validação do método será efetuada por meio de comparações empíricas tanto com estratégias clássicas de aprendizado por reforço quanto com algoritmos especialmente projetados para lidar com cenários não-estacionários.

A contribuição desta dissertação consiste no desenvolvimento de um algoritmo, chamado de RL-CD (*Reinforcement Learning with Context Detection*), o qual permite a utilização efetiva de técnicas convencionais de RL em ambientes não-estacionários. De fato, o RL-CD foi projetado de forma a ser independente do algoritmo de aprendizado específico, atuando externamente a este na forma de um guia para a gerência e seleção de múltiplos modelos de previsão e políticas parciais.

Além da proposta do RL-CD, esta dissertação também contribui com a análise teórica do parâmetro mais importante do algoritmo, o qual especifica a qualidade mínima aceitável para cada modelo. Sempre que a qualidade do melhor modelo disponível for ainda pior que o valor de qualidade mínima, um novo modelo de previsão será criado. Note que o ajuste deste parâmetro é bastante difícil, uma vez que valores muito baixos podem impedir que o agente detecte mudanças legítimas na dinâmica do ambiente, enquanto que valores muito altos podem fazer com que modelos desnecessários sejam criados. O estudo teórico deste parâmetro será desenvolvido através da análise das distribuições de probabilidade da qualidade de modelos em diferentes situações (ex: durante trocas de dinâmica em ambientes não-estacionários, durante operação em regimes estacionários, etc). A partir desses resultados, poderemos não somente analisar as características do RL-CD sob o prisma do teste estatístico de hipóteses, como também poderemos encontrar o valor ideal de qualidade mínima aceitável. Este valor corresponde àquele que minimiza simultaneamente as probabilidades de falsos positivos (falsas detecções de troca de dinâmica do ambiente) e falsos negativos (não detecção de trocas legítimas).

O texto dessa dissertação está organizado da seguinte maneira. No capítulo 2, apresen-

tamos uma revisão abrangente sobre a área de aprendizado por reforço, incluindo as três principais abordagens disponíveis: métodos de programação dinâmica, de Monte Carlo e de Diferenças Temporais; ainda no capítulo 2, apresentamos e discutimos algumas técnicas para lidar com ambientes não-estacionários. A seguir, no capítulo 3, formalizamos a proposta do RL-CD e apresentamos comparações empíricas com outros algoritmos. No capítulo 4 discutimos, do ponto de vista do teste estatístico de hipóteses, algumas das propriedades do RL-CD no que tange à detecção de contextos, e também encontramos o valor ótimo para o parâmetro principal do método. Por fim, no capítulo 5 apresentamos as conclusões da dissertação e indicamos possíveis trabalhos futuros.

2 APRENDIZADO POR REFORÇO

Estudos pioneiros sobre aprendizado animal, datados do início do século passado (THORNDIKE, 1911), demonstram a existência da relação entre a tendência de um comportamento se repetir e os níveis de satisfação obtidos a partir de tal comportamento. A teoria, chamada de condicionamento operante, postula que todo e qualquer ato que produza satisfação cria associações, na mente do animal, de forma que a probabilidade de repetição de tal ato torna-se maior. Ao estudar o comportamento de gatos em labirintos, Thorndike (THORNDIKE, 1911) observou que o tempo necessário para alcançar a saída pela primeira vez era bastante elevado. Entretanto, conforme as experiências do gato se acumulavam, todos comportamentos não-efetivos tendiam a se tornar menos freqüentes. A partir desta observação, foi proposta a chamada Lei de Efeito, que sugere que comportamentos bem-sucedidos, isto é, comportamentos que levam o animal a conseqüências satisfatórias (*recompensas*), tendem a se tornar mais freqüentes. Da mesma forma, comportamentos que levam a situações não-satisfatórias ou desagradáveis (*punições*) tendem a ser extintos.

Embora atualmente se conheça várias limitações para esta teoria, os experimentos de condicionamento animal se tornaram clássicos e são repetidos até hoje. De forma geral, o uso de recompensas e punições como forma de guiar o aprendizado influenciou fortemente a área de pesquisa da inteligência artificial ocupada do aprendizado de comportamentos sem professores, isto é, sem a indicação de quais são os comportamentos corretos a cada momento. Esta área é hoje conhecida hoje como Aprendizado por Reforço, e supõe a existência não de um professor, mas sim de um *crítico*, o qual *avalia* as ações do aprendiz sem no entanto indicar a resposta correta. Essa avaliação se dá na forma de sinais de reforço ou de punição com base no tipo de comportamento (ação) efetuado pelo aprendiz em cada situação. A conseqüência do aprendizado, nesse contexto, é que o aprendiz tende a escolher ações que maximizam os reforços esperados para o futuro.

Embora o aprendiz consiga observar os sinais de recompensa recebidos a cada instante, a tarefa de aprender por reforço não é simples. Este tipo de aprendizado implica a determinação de um mapeamento de situações para comportamentos de forma a maximizar o total de recompensas futuras tanto quanto possível. As dificuldades para esse tipo de aprendizado se devem principalmente a três razões: *1)* o aprendiz não recebe instruções explícitas de como atingir seus objetivos, e portanto precisa determinar, por *tentativa-e-erro*, quais ações são mais vantajosas; *2)* o ambiente pode ser altamente estocástico e imprevisível, e, no caso geral, não se pode assumir que o agente possui quaisquer modelos de previsão; e *3)* as ações tomadas não necessariamente geram recompensas no exato instante em que são tomadas. Em alguns casos, uma ação instantaneamente desvantajosa pode ser essencial para levar o aprendiz a áreas do espaço de estados capazes de fornecer bons sinais de reforço. Dito de outra forma, problemas gerais de RL podem apresen-

tar *recompensas atrasadas*[1]. Esse complicante faz com que o agente precise ser capaz de determinar o quanto cada uma de suas ações passadas contribuiu para a obtenção de determinada recompensa acumulada. Na Inteligência Artificial, essa dificuldade é conhecida como o *problema da atribuição de crédito*, e de certa forma todas abordagens baseadas em reforço constituem tentativas de resolver este problema.

O restante desta capítulo será organizado da seguinte forma. Na seção 2.1 iremos apresentar uma breve motivação e contextualização do aprendizado por reforço dentro do contexto mais amplo da IA. A seguir, nas seções 2.2 e 2.3, iremos formalizar os elementos básicos definidores dos problemas gerais de aprendizado por reforço e dos Processos de Decisão de Markov. Na seção 2.4 abordaremos as técnicas RL baseadas em programação dinâmica; na seção 2.5, as técnicas baseadas em métodos numéricos Monte Carlo; e por fim, na seção 2.6, abordaremos o RL no âmbito de métodos de diferenças temporais. O uso de traços de elegibilidade e de modelos de previsão como meios de acelerar o aprendizado é discutido nas seções 2.7 e 2.8. Por fim, algumas técnicas desenvolvidas para lidar com o problema da não-estacionariedade são discutidas na seção 2.9.

2.1 Breve contextualização

Antes de iniciarmos a descrição formal dos algoritmos de RL, é interessante contextualizar tal área de pesquisa. Historicamente, o aprendizado por reforço sempre esteve ligada à área de controle ótimo (BELLMAN, 1957) e aos primeiros dias da cibernética, e teve como inspiração trabalhos da estatística, engenharia, neurociências, biologia (aprendizado animal) e psicologia. Esta última fonte de inspiração rendeu várias críticas, uma vez que muitos argumentam que o aprendizado por reforço constitui apenas uma repetição da reconhecidamente limitada teoria psicológica do *behaviorismo*. Deve-se reconhecer que aprendizado por reforço possui raízes nas pesquisas de comportamento animal, e que os conceitos de *estados* e *ações* do RL são essencialmente correlatos aos estímulos e respostas do *behaviorismo*. Entretanto, enquanto que o *behaviorismo* peca por focar excessivamente (e apenas) nos comportamentos, recusando-se a considerar o que acontece internamente ao aprendiz, o RL muitas vezes se preocupa justamente em guiar o aprendizado através da construção de complexos *modelos internos causais* do ambiente. É possível perceber, por exemplo, que linhas de pesquisa recentes, como as que estudam RL no contexto de sistemas com observações imperfeitas do estado, buscam modelagens aperfeiçoadas do ambiente por meio da análise da estrutura intrínseca do espaço de estados. Este tipo de modelagem interna do mundo nunca seria admitido em uma teoria behaviorista. Em outras palavras, o problema geralmente assumido pelo RL – determinar o melhor curso de ação – acaba, muitas vezes, exigindo representações complexas e eficientes do mundo.

Conforme mencionado anteriormente, os métodos de RL também possuem intersecções com a engenharia, estatística e aprendizado animal. As soluções da engenharia para o problema de controle em geral assumem a existência de modelos completos do ambiente, e baseiam-se em técnicas de programação dinâmica. As abordagens puramente estatísticas, por outro lado, baseiam-se em métodos numéricos, como os de Monte Carlo. Por fim, os métodos relacionados com o aprendizado animal baseiam-se em correções de diferença temporal entre estimativas. Todas essas abordagens possuem pontos em comum; em muitos casos, a área de aprendizado por reforço propõe justamente maneiras de implementar pontes entre essas metodologias. Um exemplo disso, que será melhor

[1] do inglês *delayed rewards*.

discutido nas próximas seções, diz respeito ao método TD(λ), que essencialmente unifica e apresenta sob o mesmo véu as aproximações Monte Carlo e as correções de diferença temporal de um passo.

2.2 Elementos do aprendizado por reforço

Como veremos a seguir, o que se convencionou chamar de "Aprendizado por Reforço" é na verdade uma coleção de métodos de controle e de estimação para uma família específica de funções. O primeiro passo para compreendermos o RL consiste em entendermos e discutirmos seus componentes formadores básicos. Todos os métodos de aprendizado por reforço possuem, de uma forma ou de outra:

- *uma política* π, responsável por fazer o mapeamento de situações para ações;
- *uma função de recompensa* $R(s, a)$, responsável por especificar sinais numéricos de reforço a serem recebidos em resposta ao comportamento a no estado s; a função de recompensa implementa a entidade que critica individualmente, e instantaneamente, as ações tomadas pelo agente;
- *uma função de valor* $V(s)$ ou $Q(s, a)$, responsável por representar, respectivamente, a expectativa do agente quanto à recompensa futura partindo do estado s, ou então à recompensa futura partindo de s e escolhendo a ação a. Note que esta formulação diz respeito especificamente a sistemas com múltiplos estados. No caso de aprendizado em sistemas sem estados (ou com um único estado), como os apresentados na seção 2.2.3, a função de valor pode ser do tipo $Q(a)$, para toda ação a.;
- opcionalmente, *um modelo do ambiente*.

As *funções de recompensa* R essencialmente definem o problema a ser resolvido: diferentes sinais de reforço implicam diferentes tarefas a serem cumpridas. Pode-se entender um sinal de reforço como um indicativo de quão intrinsecamente desejável é cada estado (ou ação, no caso de sistemas sem estado) do ambiente. É importante ressaltar que algumas funções de recompensa são definidas de acordo com o estado atual e a última ação do agente; outras, a partir da ação do agente e do tipo de transição observada no sistema. Nesse último caso, a função teria como a assinatura $R(s, a, s')$, onde s' corresponde ao estado atingido após executar a em s. Em última análise, o objetivo do agente é maximizar a soma futura esperada destes sinais de reforço. Fazendo uma análise do problema de RL sob o prisma do aprendizado animal, é natural compararmos tais sinais de reforço com as sensações de prazer e dor.

Intimamente relacionada à função de recompensa, está a *função de valor*. Essa função mensura o quanto cada estado (ou par de estado-ação) é bom a longo prazo. Em outras palavras, as funções de valor expressam a expectativa do agente para a recompensa total acumulada que pode ser recebida a partir de um estado, seguindo-se determinada política. Essa função é necessária ao se lidar com problemas de recompensas atrasadas, pois o recebimento de recompensas imediatas máximas nem sempre corresponde à melhor estratégia. É comum, por exemplo, que sinais de reforço baixos não sejam atrativos instantaneamente, mas que ainda assim sejam desejáveis por levarem, com alta probabilidade, a regiões promissoras do espaço de estados. Em resumo, um agente de RL utiliza a função de valor para *determinar uma política de ação satisfatória a longo prazo*. A

estimativa desta função implica um constante embate entre recompensas imediatas e benefícios futuros.

Os métodos de RL também podem assumir a existência de *modelos do ambiente*. Esses modelos podem ser fornecidos diretamente ao agente ou então incrementalmente estimados a partir de experiências. Caso estimados, os modelos são construídos após a observação de amostras de trajetórias no espaço de estados; com base nessas amostras, o agente constrói uma representação interna tanto das probabilidades de transição entre estados quanto das recompensas imediatas recebidas em cada situação. É importante ressaltar que modelos do ambiente não são estritamente necessários para o aprendizado por reforço. Existem tanto métodos capazes de estimar as funções de valor a partir de interação direta com o ambiente, sem modelos (métodos chamados livre-de-modelos), quanto métodos que executam tal estimação a partir de combinações entre conhecimentos empíricos e "simulações mentais" (métodos ditos baseados em modelo). Atualmente, o espectro de abordagens para RL varia desde processos que operam somente por tentativa-e-erro, até métodos de planejamento puro, incluindo nesse ínterim vários graus de abordagens mistas. Exemplos de várias destas abordagens serão discutidos nas subseções seguintes.

2.2.1 Modelo padrão de aprendizado por reforço

No modelo padrão de aprendizado por reforço, o agente está conectado ao seu ambiente através de percepções e ações. A cada passo de interação, o agente recebe como entrada uma indicação do estado atual do ambiente, e, com base neste estado, escolhe e efetua uma ação. A ação modifica probabilisticamente o estado atual do ambiente. A transição para um novo estado gera um valor escalar de reforço, ou recompensa, o qual é imediatamente comunicado ao agente. O comportamento do agente, a cada momento, é dado por uma política de ação, e deve ser tal que as ações escolhidas maximizem a soma esperada de recompensas ao longo do tempo. Uma representação esquemática deste ciclo é apresentada na figura 2.1. Esse ciclo de observação do estado, decisão de ação, observação de recompensa, e transição para um novo estado, se repete até o final de um episódio de experimentação, ou até que o agente encontre um estado terminal, a partir do qual não pode mais sair nem receber reforços adicionais.

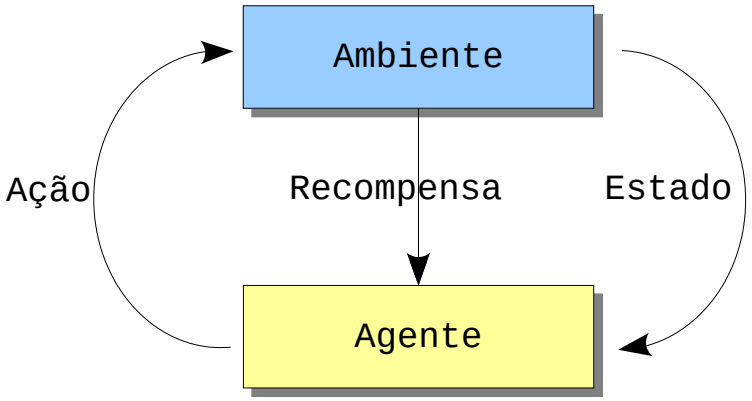

Figura 2.1: Arquitetura tradicional de um sistema de aprendizado por reforço

Formalmente, dizemos que o agente e o ambiente interagem em uma série de passos de tempo discretos, $t = 0, 1, 2, 3, \ldots$. A cada passo t, o agente observa uma representação

de seu estado atual, $s_t \in S$, sendo S é o conjunto de todos estados. Com base nisso, escolhe uma ação, $a_t \in A$, onde $A(s_t)$ é o conjunto de ações disponíveis no estado s_t. Em muitas situações, as ações disponíveis são as mesmas independentemente do estado. No passo seguinte, o agente recebe, com base no estado anterior e na ação escolhida, um sinal de reforço, r_{t+1}, e é transportado para o novo estado, s_{t+1}. A cada passo, a escolha da ação se dá através de uma política π_t tal que $\pi_t(s, a)$ informa a probabilidade de se escolher a ação a no tempo t, dado que o estado naquele tempo é s.

Para fins de análise teórica, em geral se assume que o conjunto de estados pode ser enumerado e é finito. Como veremos, isso facilita a construção de métodos tabulares de aprendizado, e também permite a construção de várias provas de convergência. Para o caso de sistemas com espaços de estado contínuos, é possível optar pelo uso de aproximadores de função e de métodos de discretização do espaço de estados. Para alguns tipos de aproximadores lineares, é possível provar a convergência para políticas que maximizam a recompensa futura esperada. Entretanto, sabe-se que o uso de aproximadores não-lineares, como redes neurais, pode causar divergência das funções de valor mesmo em cenários bastante simples. O leitor interessado em uma discussão preliminar sobre o uso de generalização em RL pode consultar (SUTTON; BARTO, 1998). Para alguns resultados interessantes em sistemas de espaço e tempo contínuos, veja (DOYA et al., 2002).

Em geral, tarefas de RL também assumem que o ambiente no qual o agente se encontra é não-determinístico, o que significa que tanto as transições entre estados, dada uma ação, quanto as recompensas recebidas, dada uma ação e um estado, obedecem a *distribuições de probabilidade*. Isso implica que observações individuais de transições são apenas *amostras* da distribuição real, mas que, com suficiente amostragem, os parâmetros destas distribuições podem ser corretamente estimados.

Adicionalmente, em geral assume-se que os parâmetros destas distribuições de probabilidade, embora desconhecidos, não variam com o tempo. Nesses casos, dizemos que o ambiente é *estacionário*. **No caso de ambientes não-estacionários, as distribuições probabilísticas que ditam a dinâmica do ambiente e das recompensas podem mudar**, muitas vezes em função de variáveis ocultas às quais o agente não tem acesso.

2.2.2 Modelos de comportamento ótimo

Conforme mencionamos anteriormente, o objetivo de um agente RL é maximizar alguma medida de recompensas recebidas a longo prazo. Existem diversas formulações matemáticas para essas medidas. Algumas consistem simplesmente em somatórios de todas as recompensas recebidas durante um determinado período de tempo; outras, úteis especialmente quando tal somatório pode ser infinito, incluem fatores que privilegiam recompensas imediatas em detrimento de recompensas futuras.

O primeiro modelo de comportamento ótimo a ser apresentado chama-se modelo de *horizonte finito*, e é o mais simples de ser entendido. A cada passo, o objetivo do agente é maximizar a recompensa esperada para as próximas h iterações de experiência no mundo. Note que falamos em *recompensas esperadas*, pois, no caso geral, o ambiente não é determinístico, e portanto, mesmo para uma política fixa, não podemos determinar exatamente quais recompensas serão recebidas a cada tempo. A recompensa futura esperada, nesse caso, é:

$$E(\sum_{t=0}^{h} r_t)$$

onde r_t é a recompensa recebida no tempo t e E é o operador de expectativa. Essa equação pode ser interpretada e utilizada de duas maneiras. Na primeira delas, o agente está ciente de que o seu tempo de vida terminará após h passos, e portanto precisa agir de maneira a maximizar seu ganho nesse período de tempo. A maneira convencional de se fazer isso é agir de acordo com uma política não-estacionária, isto é, uma política que depende do tempo. No tempo 0, o agente age de acordo com a política h-ótima, ou seja, uma política que maximiza o reforço sabendo que ainda existem h passos à frente; no tempo 1, o agente age de acordo com uma política $(h-1)$-ótima, e assim por diante. A segunda maneira de interpretar a equação para o caso de horizonte finito é imaginar que o tempo de vida do agente se estende indefinidamente, mas que, a cada momento, ele só se preocupa com os h próximos passos. Neste caso, chamado de horizonte retrocedente, o agente deve agir sempre de acordo com uma política h-ótima.

A formulação de recompensas futuras por horizonte finito pode ser útil em muitos cenários, mas falha em casos em que não é fácil determinar uma janela de tempo especialmente significativa. Nessas situações, um modelo de horizonte infinito é mais interessante. Entretanto, dado que a soma de recompensas para problemas de horizonte infinito pode ser infinita, costuma-se descontar geometricamente recompensas futuras de acordo com um parâmetro γ. Sob essa ótica, a formulação da recompensa futura é dada por:

$$E(\sum_{t=0}^{\infty} \gamma^t r_t)$$

Podemos interpretar o fator γ, ou *taxa de desconto*, como um fator de juros, como a probabilidade de viver por mais um passo, ou simplesmente como um truque matemático para manter o somatório limitado. Note que a taxa de desconto faz com que um reforço recebido k passos no futuro valha apenas γ^{k-1} vezes o que valeria se fosse recebido imediatamente. Conforme γ tende a um, o agente se torna mais "visionário", estando disposto a aguardar mais tempo por reforços atrasados, ao invés de se preocupar com retornos imediatos.

Ainda outro critério de otimalidade pode ser dado pelo modelo de *recompensa média*. Esse modelo é tal que as ações do agente devem otimizar a média de recompensas a longo prazo:

$$\lim_{h \to \infty} E(\frac{1}{h} \sum_{t=0}^{h} r_t)$$

Um dos problemas dessa formulação, e que impede que seja adotada em muitos problemas, é que ela pondera de forma igual todos as recompensas. Portanto, se torna impossível para o agente distinguir, analisando apenas a soma esperada de reforços, políticas que geram grandes recompensas apenas no início da vida do agente, daquelas que não o fazem. Por essa razão, ocorre que o agente não conseguiria diferenciar políticas capazes de gerar recompensas iniciais muito atraentes, uma vez que tais recompensas seriam absorvidas pelos reforços recebidos a longo prazo.

Uma vez que as formulações apresentadas anteriormente são bastante distintas, é evidente que as políticas que as maximizam também podem ser distintas. Por essa razão, a escolha correta do modelo de otimalidade para determinado problema é crucial. Modelos de horizonte finito são interessantes quando se conhece o tempo de vida do agente; entretanto, quando não se assume um horizonte retrocedente, precisa-se lidar com políticas não-estacionárias, dependentes do tempo restante até o final do episódio. Modelos de ho-

rizonte infinito, por outro lado, facilitam a análise matemática das equações, mas exigem o ajuste do parâmetro γ, muitas vezes feito de maneira subjetiva.

2.2.3 Aprendizado por Reforço em sistemas com um estado

Considere, inicialmente, a tarefa de escolher dentre n ações disponíveis, em um sistema onde há apenas um estado. Um exemplo clássico deste cenário se chama *k-armed bandit*, e corresponde à seguinte situação: um agente se encontra frente a uma máquina de caça-níqueis com k alavancas. A cada momento, ele pode escolher uma dentre as k opções, e o deve fazer de modo a maximizar seus ganhos. O agente possui apenas h tentativas. Nesse caso, assumimos que a alavanca i pode pagar ou não, ou seja, emitir recompensas de 1 ou 0, respectivamente. As recompensas são fornecidas de acordo com algum parâmetro p_i probabilidade, o qual é desconhecido. Dado este cenário, qual a melhor estratégia a ser seguida?

Dado que o agente não conhece a probabilidade de ganhar, para cada alavanca, uma estratégia aleatória poderia parecer interessante. Entretanto, o agente sempre poderia, através da execução de alguns testes com cada alavanca, *estimar* as probabilidades de vencer. Digamos que o agente puxou a primeira alavanca 3 vezes e foi sorteado duas vezes. Será que valeria a pena investir algumas de suas tentativas para descobrir o comportamento das outras alavancas? Talvez exista alguma alavanca que pague em 100% das vezes; talvez todas as outras alavancas sejam piores que a já tentada. Esse dilema é conhecido na IA como *dilema do aproveitamento-exploração*, e pode ser enunciado da seguinte forma: "até que ponto um agente deve utilizar apenas seu conhecimento atual, e garantir uma certa quantidade de reforços, ao invés de explorar outras ações que por ventura possam vir a se mostrar mais vantajosas?" Este problema não possui uma resposta fechada, mas, ao que tudo indica, a melhor política de exploração depende basicamente do tempo de vida do agente e da incerteza associada às expectativas de reforço futuro.

Dizemos que o agente aproveita seu conhecimento atual sempre que escolhe uma ação gulosa (do inglês, *greedy*), ou seja, sempre que sua ação for tal que, de acordo com seu conhecimento, trará os melhores benefícios. Dado que o conhecimento do agente é limitado, e que ele pode ainda não ter experimentado outras ações mais rentáveis, o agente pode, eventualmente, abrir mão da escolha gulosa e decidir explorar o ambiente através de escolhas que lhe pareçam subótimas. Uma maneira heurística de forçar o agente a explorar é fazê-lo otimista em face à incerteza, isto é, configurá-lo de forma que ações não tentadas sempre pareçam fornecer excelentes recompensas. Dessa forma, mesmo depois de já ter descoberto ações boas para algumas situações, o agente continuará explorando todas as opções desconhecidas, na maioria das vezes apenas para se decepcionar com seus resultados.

2.2.3.1 Métodos de valor-de-ação

Imagine que, para o problema do *k-armed bandit*, exista um mecanismo de escolha que iterativamente estime a probabilidade de uma alavanca a vencer. Supondo que, até o tempo t, a alavanca a tenha sido puxada k vezes, podemos dizer que

$$Q_t(a) = \frac{r_1 + r_2 + \cdots + r_k}{k}$$

onde $Q(a)$ é a expectativa de recompensa ao se escolher a. Pela Lei dos Grandes Números, sabemos que, quando $k \to \infty$, $Q_t(a)$ converge para $Q^*(a)$, ou seja, para a probabilidade correta de a fornecer a recompensa.

Perceba que não é preciso armazenar todas as recompensas recebidas no passado, a fim de poder atualizar Q no tempo $t+1$. Dada a $(k+1)$-ésima recompensa, r_{k+1}, a média de todos os $k+1$ reforços passados pode ser computada incrementalmente por:

$$\begin{aligned} Q_{k+1} &= \frac{1}{k+1}\sum_{i=1}^{k+1} r_i \\ &= \frac{1}{k+1}\left(r_{k+1} + \sum_{i=1}^{k} r_i\right) \\ &= \frac{1}{k+1}(r_{k+1} + kQ_k) \\ &= \frac{1}{k+1}(r_{k+1} + kQ_k + Q_k - Q_k) \\ &= \frac{1}{k+1}(r_{k+1} + (k+1)Q_k - Q_k) \\ &= Q_k + \frac{1}{k+1}\left(r_{k+1} - Q_k\right) \end{aligned}$$

Esse tipo de atualização, na forma

$$Estimativa_{(t+1)} \leftarrow Estimativa_{(t)} + PassoAtualizacao\Big(Alvo\ -\ Estimativa_{(t)}\Big) \tag{2.1}$$

é comum em problemas de aprendizado por reforço. No caso da atualização da média móvel de Q, o passo de atualização é variável e dado por $\frac{1}{k+1}$. Em geral, poderíamos assumir que esse passo é um valor qualquer, fixo ou variável. O valor $Alvo - Estimativa_{(t)}$ é chamado de *erro*, pois reflete o quanto o alvo difere do valor estimado até então.

Para os valores Q estimados como acima, a ação gulosa é dada por $a^* = \arg\max_a Q_t(a)$. A fim de enfrentar o dilema do aproveitamento-exploração, o agente poderá escolher explorar ações que, dado o seu conhecimento atual, lhe pareçam inferiores. O objetivo, conforme já mencionado, é certificar-se de que tais ações realmente não valem a pena, ou, ao contrário, descobrir que elas são ótimas, mas que apenas ainda não haviam sido tentadas ou haviam sido tentadas poucas vezes. Uma forma de efetuar este tipo de exploração é escolher, com probabilidade ϵ, uma ação subótima; a essa política exploratória chamamos ϵ-gulosa. Embora essa estratégia garanta convergências assintóticas, o uso de uma taxa de exploração fixa faz com que, mesmo depois que o agente já tenha determinado os parâmetros corretos, continue tomando ações subótimas. Como o agente não tem como saber quando já explorou o suficiente, para então escolher sempre a ação gulosa, a técnica tradicional para resolver este impasse é decair ϵ com o tempo. De fato, muitas das provas de convergência para algoritmos de RL presumem que esse decaimento ocorre.

Um dos problemas de fazer a seleção ϵ-gulosa de forma completamente aleatória, isto é, de escolher uniformemente uma ação qualquer em uma fração ϵ das vezes, é que acaba-se por dar pesos iguais tanto para ações que aparentam serem muito ruins, quanto para aquelas que aparentam serem apenas um pouco piores do que a gulosa. Uma maneira de resolver isso é através de seleção softmax, que ocorre de acordo com uma distribuição de Boltzmann:

$$P(a) = \frac{e^{\frac{Q_t(a)}{\tau}}}{\sum_{i=1}^{n} e^{\frac{Q_t(i)}{\tau}}}$$

onde n é o número de ações, τ é um parâmetro de temperatura, e $P(a)$ é a probabilidade de a ação a ser escolhida. Quanto mais alta a temperatura, mais as ações se tornam equiprováveis. Temperaturas mais baixas tendem a ressaltar as diferenças na seleção de ações. Quando $\tau \to 0$, a seleção softmax equivale à gulosa.

2.2.3.2 Aprendizado de autômatos

Outra possibilidade para resolver o problema dos k-*armed bandits* é sugerida por uma área de controle adaptativo conhecida como Aprendizado de Autômatos[2] (NARENDRA; THATHACHAR, 1989). Um dos algoritmos de Aprendizado de Autômatos é chamado de L_{R-P}, ou "*Linear Reward-Penalty*". Para apresentá-lo, imagine que o agente tem que resolver o problema dos 2-*armed bandits*, isto é, tem que aprender a escolher dentre duas alavancas em uma máquina de caça-níqueis. Apenas duas recompensas são possíveis – sucesso ou fracasso – e o agente pode inferir, caso a última ação tomada não tenha sido bem sucedida, que a outra ação deveria ser a correta. Para este problema, o L_{R-P} mantém, em estruturas de dados separadas, probabilidades $\pi_t(i)$ de seleção cada ação i. Se no passo t a ação inferida correta é i, então atualiza-se sua probabilidade em direção a 1 por um fator α:

$$\pi_{t+1}(i) = \pi_t(i) + \alpha\Big(1 - \pi_t(i)\Big)$$

Complementarmente, a probabilidade da outra ação é ajustada em direção a zero, de forma que a soma das probabilidades permaneça um. Este tipo de ajuste gradual é essencial em cenários estocásticos, em que os resultados correspondem apenas a amostras, e não a valores definitivos.

Outro algoritmo proposto pela área de Aprendizado de Autômatos é o L_{R-I}, que significa "*Linear, Reward-Inaction*". Este algoritmo é semelhante ao L_{R-P}, exceto pelo fato de que as probabilidades são atualizadas apenas em jogadas onde ocorre sucesso. No caso da ação i ter falhado no tempo t, a probabilidade π_t de todas as ações permanece inalterada. Esse algoritmo converge, com probabilidade um, para um vetor de probabilidades com exatamente um elemento em 1 e todos outros em 0, mas infelizmente a ação correspondente nem sempre é a correta. A probabilidade de que o L_{R-I} convirja para a ação correta pode ser arbitrariamente reduzida diminuindo-se o valor de α (NARENDRA; THATHACHAR, 1989). Note que, por implicar probabilidades zero de seleção de ações, este método se mostra bastante inadequado para ambientes não-estacionários, uma vez que seria incapaz de se recuperar frente a alterações na dinâmica do ambiente.

2.2.3.3 Métodos de perseguição[3]

Os métodos de perseguição são uma união entre as abordagens apresentadas até agora. Os métodos de perseguição mantêm tanto estimativas Q para o valor de cada ação, como também preferências de escolha de cada ação, na forma de probabilidades π de escolha. Nesse tipo de método, as preferências estão constantemente "perseguindo" as ações

[2] do inglês *Learning Automata*.
[3] do inglês *Pursuit methods*.

gulosas em relação às estimativas atuais de valor. Após cada jogada, as probabilidades de escolha são atualizadas para fazer com que a ação gulosa se torne mais provável. Considere o caso em que, após a $(t+1)$-ésima jogada, a ação gulosa seja dada por $a^*_{t+1} = \arg\max_a Q_{t+1}(a)$. Então, a probabilidade de escolher a^*_{t+1} no tempo $t+1$ deve ser incrementada em direção a um:

$$\pi_{t+1}(a^*_{t+1}) = \pi_t(a^*_{t+1}) + \beta\Big(1 - \pi_t(a^*_{t+1})\Big)$$

e as probabilidades das demais ações devem ser decrementadas em direção a zero:

$$\pi_{t+1}(a) = \pi_t(a) + \beta\Big(0 - \pi_t(a)\Big) \qquad \forall a \neq a^*_{t+1}$$

Depois disso, os valores Q são atualizados conforme discutido nas subseções anteriores, por exemplo, em direção à média móvel das últimas recompensas). Pode-se iniciar os valores π de forma equiprovável, isto é, com $\frac{1}{n}$, caso haja n ações disponíveis, e os valores Q em zero.

2.3 Processos de Decisão de Markov

Até agora, consideramos apenas as tarefas de aprendizado em que o sistema encontrava-se sempre no mesmo estado, e nas quais o agente apenas precisava aprender a escolher dentre $|A|$ ações. No caso geral, entretanto, o agente pode ter que distinguir, adicionalmente, a melhor ação em cada um dos $|S|$ estados. Um forma particularmente útil de modelagem para este tipo de sistema é o formalismo dos Processos de Decisão de Markov (ou MDP, do inglês *Markov Decision Processes*).

Um MDP consiste em um processo estocástico caracterizado por um conjunto de estados, ações e matrizes de probabilidade de transição entre os estados. A transição entre os estados ocorre de maneira discreta no tempo e depende de uma ação. Além disso, a cada estado (ou, mais comumente, par estado-ação) é associada uma função de recompensa, a qual define o ganho instantâneo que o agente recebe.

Os Processos de Decisão de Markov em geral assumem que a *Hipótese de Markov* é válida. Esta hipótese diz que um sistema é *markoviano* caso a distribuição de probabilidades condicionais para os estados futuros dependa apenas do estado atual. Isto é equivalente a dizer que os estados subseqüentes do sistema são condicionalmente independentes dos estados pelos quais o processo já passou. Dito de outra forma, a Hipótese de Markov equivale a dizer que o sinal de estado é suficiente para resumir, de forma compacta, as informações passadas relevantes. Em um jogo de xadrez, por exemplo, o estado poderia ser representado diretamente pelas posições das peças no tabuleiro. Em sistemas markovianos, a melhor política a ser seguida a partir de determinado estado é independente da trajetória específica que levou o sistema até aquele estado.

Caso o sistema seja markoviano, então, para todo estado e toda ação, podemos dizer que

$$P(s_{t+1} = s',\, r_{t+1} = r \,|\, s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0) = P(s_{t+1},\, r_{t+1} \,|\, s_t, a_t)$$

onde $P(a\,|b)$ é a notação para probabilidade condicional de a dado o evento b. Em sistemas markovianos, um modelo de predição de um passo é suficiente para determinar a probabilidade de cada próximo estado e da próxima recompensa, sabendo-se apenas o último estado e ação.

Formalmente, dizemos que um *modelo de Markov* consiste em:

- um conjunto discreto de estados, S;
- um conjunto discreto de ações, A;
- uma função de transição de estados $T : S \times A \rightarrow \Pi(S)$, onde $\Pi(S)$ é uma distribuição de probabilidade sobre S; escrevemos $T(s, a, s')$ como a probabilidade de ocorrer a transição de s para s', dado que a ação a foi tomada.
- uma função de recompensa $R : S \times A \rightarrow \Re$.

A resolução de um processo de Markov, no âmbito de RL, consiste em encontrar uma seqüência de ações que garanta o maior ganho esperado para o sistema. Denotamos π^* como a política ótima para o MDP, ou seja, o mapeamento de estados para ações que gera o maior ganho futuro esperado.

2.3.1 Funções de valor

Praticamente todos os algoritmos de RL funcionam baseados em estimavas para *funções de valor*. Conforme mencionado anteriormente, as funções de valor estimam o quão bom é para o agente estar em determinado estado (ou o quão bom é executar determinada ação em um dado estado). No contexto de aprendizado por reforço, "quão bom" corresponde a uma medida numérica de expectativa de recompensas futuras.

Seguindo os princípios de otimalidade delineados na subseção 2.2.2, podemos definir funções de valor para uma dada política π. Informalmente, $V^\pi(s)$ corresponde ao valor esperado e acumulado de recompensas a serem recebidas a partir do estado s, no tempo t, caso o agente siga executando a política π. Formalmente,

$$V^\pi(s) = E_\pi\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s \right\} \tag{2.2}$$

e

$$Q^\pi(s, a) = E_\pi\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s,\ a_t = a \right\} \tag{2.3}$$

onde E_π corresponde ao valor esperado a ser recebido caso o agente siga a política π. Note que essas funções podem ser estimadas através de experimentações do agente; dado um número suficiente de tomadas de ações em estados, seguidas de observações do estado subseqüente e da recompensa resultante, os valores corretos para V^π (ou Q^π) podem ser estimados por métodos Monte Carlo, os quais, conforme discutido na seção 2.5, envolvem médias de amostras aleatórias de resultados de experiências.

Uma propriedade importante das funções apresentadas em (2.2) e (2.3) é que elas obedecem a uma relação recursiva conhecida como *equação de Bellman*. A equação de Bellman relaciona o valor da função no estado s com o valor da função nos possíveis estados subseqüentes, dado o modelo do ambiente (isto é, dadas as probabilidades de transição T e a distribuição de recompensas R):

$$
\begin{aligned}
V^{\pi}(s) &= E_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^{k} r_{t+k+1} \,\middle|\, s_t = s \right\} \\
&= \sum_{a} \pi(s,a) \sum_{s'} T(s,a,s') \left\{ R(s,a) + \gamma E_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^{k} r_{t+k+2} \right\} \right\} \\
&= \sum_{a} \pi(s,a) \sum_{s'} T(s,a,s') \left\{ R(s,a) + \gamma V^{\pi}(s') \right\}
\end{aligned}
$$

A Equação de Bellman, apresentada acima, consiste na formalização matemática da seguinte afirmação: *a utilidade de um estado equivale à recompensa imediata correspondente a este estado, somada à utilidade descontada esperada dos próximos estados, supondo-se que o agente continue seguindo a mesma política.*

Na terminologia de Aprendizado por Reforço, chamamos de *backup* a atualização do valor de um estado (ou par estado-ação) com base nos valores dos estados seguintes. Dito de outra forma, no momento em que obtemos uma melhor estimativa $V(s')$ podemos repassar essa informação para todos antecessores de s'. Como exemplo, imagine que o agente acabou de descobrir que um determinado estado s' fornece uma recompensa muito alta, τ. Nessa situação, um *backup* equivaleria a atualizar o valor dos estados antecessores a s' de forma a refletir o fato de que, com probabilidade proporcional à chance de cada um desses estados atingir s', eles também podem podem obter τ.

2.3.2 Funções de valor ótimas

Da mesma forma que definimos, acima, as funções de valor para uma dada política π, também podemos definir as *funções de valor ótimas*. Essas funções são tais que, para todos os estados, maximizam a recompensa futura esperada. O uso dessas funções é importante pois elas definem uma ordem parcial sobre políticas: $\pi' \geq \pi$ caso a recompensa esperada de π' for maior ou igual, em todos estados, à recompensa fornecida por π. Equivalentemente, $\pi' \geq \pi$ se e somente se $V^{\pi'}(s) \geq V^{\pi}(s)$ para todo $s \in S$. A função de valor ótima, V^*, é definida como:

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

para todo $s \in S$. Seguindo a mesma idéia, também podemos definir a função de estado-ação ótima:

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

para todo $s \in S$ e toda $a \in A$. Utilizando as Equações de Bellman, podemos reescrever as funções ótimas da seguinte forma:

$$V^*(s) = \max_{a} \left(R(s,a) + \gamma \sum_{s'} T(s,a,s') V^*(s') \right) \tag{2.4}$$

e

$$Q^*(s,a) = R(s,a) + \sum_{s'} T(s,a,s') \gamma \max_{a'} Q^*(s',a') \tag{2.5}$$

Essas equações são conhecidas como Equações de Otimalidade de Bellman, e expressam recursivamente as restrições que toda função de valor, para cada estado, deve obedecer. Note, mais uma vez, que essas equações são definidas supondo-se o conhecimento do modelo do ambiente (funções de transição T, e de recompensas, R). Como veremos mais adiante, o problema geral de aprendizado por reforço também pode ser resolvido em situações nas quais essas funções são desconhecidas.

Dada a função de valor ótima e um modelo do ambiente, a *política ótima* π^* pode ser calculada, simplesmente considerando-se a ação que maximiza a soma da recompensa imediata com a recompensa ótima esperada:

$$\pi^*(s) = \arg\max_a \left(R(s,a) + \gamma \sum_{s'} T(s,a,s') V^*(s') \right) \tag{2.6}$$

De forma semelhante, dada a função $Q^*(s,a)$ para todo s e a, podemos calcular a política ótima π^* através da equação (2.7). Note que essa equação permite o cálculo de π^* sem o conhecimento explícito do modelo de transições T, ou de recompensas, R:

$$\pi^*(s) = \arg\max_a Q(s,a) \tag{2.7}$$

É importante ressaltar que, ao menos teoricamente, poderíamos também utilizar métodos padrão de resolução de sistemas de equações não-lineares para calcular os valores de V^*. Isso é possível pois a Equação de Otimalidade de Bellman (equação (2.6)), na verdade, define um *sistema não-linear* de N equações, para N estados. Entretanto, a resolução desse tipo de sistema se torna impraticável quando o número de estados cresce, e também quando não dispomos de estimativas perfeitas para T e R. Nas próximas seções iremos estudar três abordagens para obtenção de funções de valor e/ou políticas ótimas:

1. Programação dinâmica;
2. Métodos Monte Carlo;
3. Métodos de diferença temporal.

As abordagens de programação dinâmica aproximam V^* assumindo que que há conhecimento perfeito do modelo do ambiente. Métodos Monte Carlo, por sua vez, não exigem um modelo e são conceitualmente simples, mas não podem ser implementados de forma a obter melhorias a todo passo. Por fim, métodos de diferença temporal também não requerem modelos e são totalmente incrementais, mas se mostram mais difíceis de serem analisados do ponto de vista teórico.

2.4 Programação Dinâmica

Programação dinâmica (DP, do inglês *Dynamic Programming*) é uma coleção de algoritmos que podem ser usados para calcular políticas ótimas em problemas de RL, dado um modelo perfeito do ambiente, geralmente na forma de um MDP. Embora a suposição muitas vezes irreal de um modelo perfeito, e também o grande custo de resolução exigido por esses algoritmos, façam da programação dinâmica um mecanismo pouco usado para cenários reais, sua importância teórica é indiscutível.

O termo "programação dinâmica" possui um significado mais amplo aquém daquele normalmente utilizado em Aprendizado por Reforço. Em geral, DP diz respeito à resolução de problemas cuja estrutura é formada por vários subproblemas em comum. Estes

subproblemas são tais que o cálculo da soluções ótimas de subproblemas também leva à solução ótima do problema como um todo. É interessante notar que o termo "programação" confunde o sentido original dado pelo seu criador, Richard Bellman (BELLMAN, 1957). No contexto original, *programação* era usado no sentido de *planejamento*. Portanto, programação dinâmica diz respeito ao planejamento de uma estratégia de ações, dado um modelo do sistema, para o caso em que o cálculo das decisões ótimas em um estado é baseado no cálculo de decisões ótimas em todos outros estados. A formulação DP para o problema de aprendizado por reforço é bastante interessante em termos teóricos, por ser de fácil análise e por representar o melhor cenário possível, ou seja, aquele em que se tem conhecimento perfeito. De uma forma ou de outra, todos outros métodos de RL que serão discutidos tentam aproximar as soluções de DP, mas exigindo menos computação e tomando suposições mais brandas a respeito do conhecimento do ambiente.

A seguir, iremos apresentar os dois algoritmos mais comuns de programação dinâmica: *iteração de política* e *iteração de valor*. Ambos funcionam misturando etapas de avaliação de uma política, isto é, de obtenção de estimativas para V^π, seguidas de uma etapa de atualização da política, de forma a torná-la gulosa em relação aos novos valores de V^π.

2.4.1 Iteração de política

Dado que a solução dos sistemas de equações não-lineares induzidos pelas equações (2.4) ou (2.5) é inviável em termos práticos, os métodos de programação dinâmica tentam aproximar as soluções para as funções de valor através de algoritmos iterativos.

O primeiro passo para o algoritmo de Iteração de Política (IP) é obter uma estimativa para a função V correspondente a uma política π. Inicia-se a primeira aproximação de V de forma arbitrária (exceto pelo fato de que todos estados terminais precisam possuir valor zero). A avaliação de V^π ocorre basicamente transformando-se a Equação de Otimalidade de Bellman (2.4) em uma regra de atualização:

$$V_{k+1}(s) = \sum_a \pi(s,a)\left(R(s,a) + \sum_{s'} T(s,a,s')\gamma V_k(s')\right) \tag{2.8}$$

para todo $s \in S$. Essa equação tem como ponto fixo $V_k = V^\pi$, uma vez que a própria Equação de Otimalidade de Bellman nos assegura de tal igualdade. Conforme $k \to \infty$, V_k tende para a estimativa correta V^π. Iniciando-se com uma estimativa V_0^π, calcula-se o valor de $V_1^\pi(s)$ com base nos valores de $V_0^\pi(k)$, para todo k sucessor de s. Essa atualização segue para V_2^π, V_3^π, ..., V_k^π, até que os valores de $V(s)$ não se alterem mais do que alguma quantidade mínima estipulada, θ (resíduo de Bellman). O algoritmo que implementa o primeiro requisito da Iteração de Política, ou seja, a *avaliação de uma política*, é apresentado em 1.

Note que a atualização do valor de cada estado leva em conta o valor de *todos* os estados sucessores; nesse caso, dizemos que o algoritmo baseia-se em *backups completos*[4]. Algoritmos que aproximam $V^\pi(s)$ com base em amostras do valor de apenas um estado sucessor também são possíveis, principalmente no âmbito de métodos livres-de-modelo, e serão discutidos mais adiante. Note também que o algoritmo, ao calcular o valor $V_{k+1}(s)$, considera que as ações a serem tomadas no estado s são probabilísticas; isto é, a ação a pode ser escolhida com probabilidade $\pi(s,a)$, a ação b com probabilidade $\pi(s,b)$, etc. Entretanto, sabe-se que qualquer Processo de Markov possui como solução ótima ao menos

[4]do inglês *full backups*.

Algoritmo 1 Avaliação iterativa de política

Seja π a política a ser avaliada
1: Inicialize $V_0(s) \leftarrow 0$ para todos estados;
2: **Repita**
3: $\quad \Delta \leftarrow 0$
4: $\quad$ **para todo** $s \in S$ **faça**
5: $\quad\quad v \leftarrow V_k(s)$
6: $\quad\quad V_{k+1}(s) \leftarrow \sum_a \pi(s,a)\,(R(s,a) + \sum_{s'} T(s,a,s')\gamma V_k(s'))$
7: $\quad\quad \Delta \leftarrow \max(\Delta, |v - V_{k+1}(s)|)$
8: $\quad$ **fim para**
9: **até que** $\Delta < \theta$
10: Retorne $V \approx V^\pi$

uma política determinística. Por essa razão, também é possível atualizar o valor $V_{k+1}(s)$ considerando-se apenas a ação $\pi(s)$ sugerida pela política (determinística) atual:

$$V_{k+1}(s) \leftarrow R(s,\pi(s)) + \gamma \sum_{s'} T(s,\pi(s),s')V_k(s')$$

Dada a correção iterativa para a estimativa V^π, o segundo passo para o algoritmo de Iteração de Política é corrigir a política. Isso é feito basicamente testando-se, para todo estado s, se a ação atualmente sugerida pela política não poderia ser substituída por uma melhor, dada a nova estimativa mais precisa de V^π. Essa ciclo de atualização da função de valor, seguido da correção da política, define o algoritmo de iteração de política. A ação gulosa da política corrigida, π', é dada por:

$$\begin{aligned} \pi'(s) &= \arg\max_a Q^\pi(s,a) \\ &= \arg\max_a R(s,a) + \sum_{s'} T(s,a,s')\gamma V^\pi(s') \end{aligned}$$

O algoritmo completo para a iteração de política consiste em uma série de atualizações de V e de π na forma:

$$\pi_0 \xrightarrow{A} V^{\pi_0} \xrightarrow{M} \pi_1 \xrightarrow{A} V^{\pi_1} \xrightarrow{M} \ldots \pi^* \xrightarrow{A} V^*$$

onde A equivale ao passo de avaliação da política, e M equivale ao passo de melhoria (correção da política para se tornar gulosa em relação a V^π). O processo completo é apresentado no algoritmo 2. Note que as linhas 2 até 10 equivalem ao passo A, enquanto que as linhas 11 até 20 equivalem ao passo M. Sabe-se que esse algoritmo converge para a política ótima em tempo pseudopolinomial, no pior caso (LITTMAN; DEAN; KAELBLING, 1995).

2.4.2 Iteração de valor

Um dos problemas da IP é que cada iteração exige a *avaliação completa de uma política*, o que por si só já é processo iterativo que exige múltiplas passadas por todos os estados. Quando o número de estados cresce, cada iteração deste algoritmo pode demorar muito. Entretanto, empiricamente é possível perceber que, em alguns casos, antes mesmo que o valor numérico de V tenha convergido para V^π, a política induzida já é ótima. Isso

Algoritmo 2 Iteração de política

1: Para todo $s \in S$, inicialize $V_0(s)$ e $\pi(s)$ arbitrariamente
2: **Repita**
3: $\Delta \leftarrow 0$
4: **para todo** $s \in S$ **faça**
5: $v \leftarrow V_k(s)$
6: $V_{k+1}(s) \leftarrow R(s, \pi(s)) + \sum_{s'} T(s, \pi(s), s')\gamma V_k(s')$
7: $\Delta \leftarrow \max(\Delta, |v - V_{k+1}(s)|)$
8: **fim para**
9: **até que** $\Delta < \theta$
10: política-estável $\leftarrow$ verdadeiro
11: **para todo** $s \in S$ **faça**
12: $b \leftarrow \pi(s)$
13: $\pi(s) \leftarrow \arg\max_a R(s, a) + \sum_{s'} T(s, a, s')\gamma V^{\pi}(s')$
14: **se** $b \neq \pi(s)$ **então**
15: política-estável $\leftarrow$ falso
16: **fim se**
17: **se** não política-estável **então**
18: volte para passo 2
19: **fim se**
20: **fim para**
21: Retorne $\pi \approx \pi^*$

pode ocorrer pois pequenas mudanças em V nem sempre alteram a política, e o algoritmo pode demorar bastante tempo até conseguir reduzir suficientemente o resíduo de Bellman para todos os estados.

Uma possível otimização para o IP, portanto, é truncar o passo de avaliação da política: ao invés de percorrer inúmeras vezes todos os estados, corrigindo seus valores, até que as mudanças sejam arbitrariamente pequenas, é possível fazer apenas *uma* correção para cada estado, com base nos valores estimados no passo anterior. Se quisermos representar essa idéia em uma equação, e ainda unificar no mesmo passo a melhoria da política, obtemos:

$$V_{k+1}(s) = \max_a R(s, a) + \sum_{s'} T(s, a, s')\gamma V_k(s')$$

Prestando atenção a esse resultado, percebemos que ele equivale exatamente a transformar a Equação de Otimalidade de Bellman (eq. (2.4)) em uma regra de atualização. A esse tipo de atualização chamamos de Iteração de Valor. O efeito prático da equação proposta é combinar, a cada passagem pelo conjunto de estados, tanto os passos de avaliação quanto os de melhoria da política. Isso faz com que, ao contrário do que ocorre na IP, políticas razoáveis já estejam disponíveis mesmo após poucos passos de execução do algoritmo. O processo completo é descrito no algoritmo 3.

Sabe-se que o número de iterações para atingir a função de valor é polinomial no número de estados e na magnitude do maior sinal de recompensa, caso o fator de desconto seja mantido constante. Entretanto, a velocidade de convergência pode se tornar consideravelmente menor para taxas de desconto arbitrariamente próximas a um (LITTMAN; DEAN; KAELBLING, 1995). Existem várias possíveis otimizações para esse algoritmo, incluindo a mistura de experiências reais do agente, durante o cálculo de programação

Algoritmo 3 Iteração de valor

Para todo $s \in S$, inicialize $V_0(s)$ arbitrariamente
Repita
  $\Delta \leftarrow 0$
  para todo $s \in S$ **faça**
    $v \leftarrow V_k(s)$
    $V_{k+1}(s) \leftarrow \max_a R(s,a) + \sum_{s'} T(s,a,s')\gamma V_k(s')$
    $\Delta \leftarrow \max(\Delta, |v - V_{k+1}(s)|)$
  fim para
até que $\Delta < \theta$
Retorne uma política determinística π tal que
$\pi(s) = \arg\max_a R(s,a) + \sum_{s'} T(s,a,s')\gamma V(s')$

dinâmica, a fim de determinar quais áreas do espaço de estados merecem ser atualizadas antes. Essa abordagem é interessante visto que a garantia de convergência para o método de iteração de valor não depende da ordem em que os backups são feitos. Em outras palavras, atualizar prioritariamente os estados mais importantes pode não apenas acelerar o tempo total de processamento, como não fere as condições de convergência.

2.5 Métodos Monte Carlo

As abordagens de Programação Dinâmica mencionadas na seção anterior apresentam três principais problemas que as tornam difíceis de serem amplamente aplicadas: *1)* assumem um modelo perfeito do ambiente; *2)* podem exigir uma quantidade muito grande de computação, para casos em que o número de estados é grande; e *3)* são pensadas para planejamento, ou seja, para o caso em que é possível ocorrer aprendizado *offline* baseado em modelos completos. Em casos em que o agente é imperfeito e necessita aprender *enquanto* imerso em um ambiente, a programação dinâmica se mostra inconveniente.

Uma alternativa aos métodos DP são os chamados *Métodos Monte Carlo* (MC). Nestes, não se exige um modelo completo, e sim apenas um conjunto de experiências concretas do agente, na forma de amostras de seqüências de estados, ações e recompensas recebidas. Essas seqüências podem ser obtidas tanto a partir da interação online (real) do agente, ou através de simulação do ambiente de interesse. É importante ressaltar que a possibilidade de se utilizar amostras de experiências obtidas por simulação é muito interessante, pois não exige um modelo completo do ambiente. Um modelo completo não é necessário para obter para obter exemplos de trajetórias de estados-ações-recompensas em um jogo de xadrez, por exemplo, pois basta conhecermos as regras do jogo e as simularmos na forma de uma partida. Não é preciso explicitarmos a matriz de transições completa entre todas as possíveis configurações de tabuleiro. Esse tipo de simulação é muito útil, visto que é comum encontrarmos cenários em que é fácil amostrar exemplos de transição, de acordo com uma determinada distribuição de probabilidade, mas extremamente difícil calcular analiticamente tal distribuição em sua forma explícita.

A abordagem MC baseia-se principalmente no cálculo de médias de recompensas amostradas. Por essa razão, métodos MC podem ser aplicados apenas para tarefas episódicas, ou seja, aquelas tarefas em que existe um "fim" ou uma "pausa" na qual podemos processar o conjunto de valores amostrados até então. Ao fim de cada episódio, os métodos MC atualizam as estimativas de política e da função de valor. Dizemos, portanto, que

são incrementais episódio-a-episódio, e não a cada experiência do agente.

Como no caso de programação dinâmica, dizemos que o aprendizado nas técnicas Monte Carlo se baseia em duas grandes etapas cíclicas: *1)* avaliação da função de valor para uma política; e *2)* correção (melhoria) da política em relação à nova função de valor. Portanto, primeiramente vamos mostrar como efetuar a avaliação de uma política através de algoritmos de Monte Carlo.

2.5.1 Avaliação de políticas

A idéia básica para avaliar uma política sob MC, e portanto estimar a função de valor para cada estado, é coletar recompensas observadas experimentalmente após a visita a determinado estado, e então calcular médias ao final de vários episódios. Este cálculo justifica-se devido à própria definição do *valor de um estado*, ou seja, a recompensa que se espera receber a partir dele. Como exemplo, considere o caso em que as iterações que seguiram a visita ao estado s, até o final do episódio, forneceram uma recompensa total de 300 pontos; na segunda vez em que observamos o seguimento da visita a s, percebemos que as iterações até o final do episódio forneceram 250 pontos de recompensa. Com base nesses dados, podemos obter uma estimativa de $V(s) = \frac{300+250}{2} = 275$.

À técnica de estimar $V^\pi(s)$ através do cálculo da média de valores que seguiram a primeira visita de s, para cada um dos s visitados, dado um conjunto de n episódios, chamamos de *Estimativa MC de Primeiras-Visitas*. O processo é apresentado no algoritmo 4. Uma pequena alteração a essa abordagem se chama *Estimativa MC para Todas-Visitas*, e consiste em calcular médias de recompensas que seguem *cada uma das visitas* a s, e não apenas a primeira. Ambos os métodos convergem para V^π conforme o número de visitas tende ao infinito (embora possuam propriedades teóricas um pouco diferentes), uma vez que cada amostra de retorno, ou de recompensas totais que seguem um estado, é i.i.d. (independente e identicamente distribuída) em relação ao valor verdadeiro de V^π. Por essa razão, a Lei dos Grandes Números nos garante que a média realmente irá convergir para o valor esperado. Note que, ao contrário das técnicas DP, a estimativa de valor para cada estado não depende das estimativas dos estados sucessores; dizemos que os métodos MC não fazem *bootstrap* (estimativas a partir de estimativas). Por essa razão, são bastante atraentes para casos em que precisamos obter a função de valor apenas para alguns estados, mas não nos interessamos pela grande maioria dos demais.

Algoritmo 4 Estimativa Monte Carlos de Primeiras-Visitas para V^π

Seja π: a política a ser avaliada;
Seja $V(s) \leftarrow 0$, para todo $s \in S$;
Seja Retornos(s): uma lista vazia, para todo s, onde iremos armazenar as médias de recompensas que seguem cada primeira-visita.

1: **Repita**
2: Gere um episódio de experiência seguindo π
3: **para todo** s visitado no episódio **faça**
4: $R \leftarrow$ soma de recompensas que seguem primeira visita a s
5: Adicione R ao final de Retornos(s)
6: $V(s) \leftarrow$ média($Retornos(s)$)
7: **fim para**
8: **fim repita**

2.5.2 Estimativa de pares estado-ação

Como dissemos, métodos MC não dispõem de um modelo do ambiente. Nesse caso, o conhecimento apenas de V não é muito útil, pois, para determinar uma política, seria preciso considerar os valores de estados subseqüentes ponderados pela respectiva probabilidade de transição (equação (2.6)). A solução óbvia é estimar, ao invés de valores de estado, valores para cada par estado-ação. Como vimos anteriormente, tendo os valores ótimos de estado-ação $Q(s, a)$, para todo s e todo a, a escolha da ação que maximiza a recompensa é dada diretamente pela equação (2.7). Note que essa equação não supõe o conhecimento explícito do modelo do ambiente (matriz de transições T e função de recompensa R), o que evidencia uma das maiores vantagens em se estimar Q ao invés de V.

O método MC para estimativa de $Q(s, a)$, para cada primeira-visita a s e a, é muito semelhante ao apresentado no algoritmo 4. Um cuidado que devemos ter, no entanto, é que garantir que todos os pares de estado-ação relevantes vão ser visitados em cada episódio. Sem essa visita, se torna impossível obter amostras de retornos, e o método MC falha por falta de informações. Uma das maneiras mais simples de resolver este dilema é supor a existência de *inícios de exploração*[5], o que equivale a garantir que todo par estado-ação deve ter probabilidade não-zero de ser selecionado como início de uma trajetória. Isso garante que existirão infinitas amostras para os valores de $Q^{\pi}(s, a)$, no limite de infinitos episódios. Outra possibilidade é supor apenas políticas estocásticas com probabilidade não-zero de selecionar todas as ações.

2.5.3 Estimativa de política MC

Tendo estimado a função Q^{π} para um dada política π, podemos planejar um algoritmo para melhoria de políticas semelhante ao método de iteração de política: ciclicamente, utilizamos a estimativa de Q para atualizar a política, de forma que ela seja gulosa em relação aos últimos valores estimados. Isso implica uma política corrigida, que, por sua vez, será usada para gerar novos valor Q^{π}, e assim por diante. O ciclo de iteração de política Monte Carlo é delineado abaixo:

$$\pi_0 \xrightarrow{A} Q^{\pi_0} \xrightarrow{M} \pi_1 \xrightarrow{A} Q^{\pi_1} \xrightarrow{M} \ldots \pi^* \xrightarrow{A} Q^*$$

onde A equivale ao passo de avaliação da política, e M equivale ao passo de melhoria (correção da política para se tornar gulosa em relação a $Q\pi$). Note que, ao contrário do DP, onde novas estimativas para V e π eram obtidas a cada iteração do algoritmo, o método MC apenas atualiza Q e π ao final de cada episódio. Por essa razão, ele não é muito eficiente em cenários de aprendizado online. O processo completo de aprendizado de política MC é apresentado no algoritmo 5. Note que as linhas 2 até 7 equivalem ao passo A, enquanto que as linhas 8 até 10 equivalem ao passo M.

Além do método apresentado no algoritmo 5, também podemos projetar abordagens MC baseadas em controle *off-policy*, isto é, estratégias de controle onde a política que se aprende *não é* a mesma política que se segue. Embora isso possa parecer estranho, ou mesmo impossível, é uma técnica bastante utilizada nos métodos de diferença temporal como o Q-Learning (vide subseção 2.6.3). Uma das grandes vantagens de métodos *off-policy* é que eles permitem, por exemplo, que se aprenda diretamente a política ótima, mesmo que o agente siga uma política exploratória, onde diversas ações subótimas são

[5] do inglês *Exploring Starts*.

Algoritmo 5 Método de controle Monte Carlos de Primeiras-Visitas, assumindo Inícios de Exploração

Seja $Q(s, a)$ inicializado arbitrariamente, para todo s e a
Seja $\pi(s)$ uma política inicial, inicializada arbitrariamente para todo s
Seja Retornos(s, a): uma lista vazia, para todo s e a, onde iremos armazenar as médias de recompensas que seguem cada primeira-visita.

Repita
  Gere um episódio de experiência seguindo π, assumindo *Inícios de Exploração*
  para todo (s, a) visitado no episódio **faça**
    $R \leftarrow$ soma de recompensas que seguem primeira visita a s, a
    Adicione R ao final de Retornos(s, a)
    $Q(s, a) \leftarrow$ média$(Retornos(s, a))$
  fim para
  para todo s visitado no episódio **faça**
    $\pi(s) \leftarrow \arg\max_a Q(s, a)$
  fim para
fim repita

utilizadas. Para detalhes de como implementar o aprendizado Monte Carlo *off-policy*, consulte (SUTTON; BARTO, 1998).

2.6 Métodos de Diferença Temporal

Como vimos, os métodos de programação dinâmica exigem um modelo completo do mundo, e fazem atualizações (correções) por *bootstrap*, ou seja, geram estimativas de valor de estado a partir de outras estimativas. Infelizmente, a abordagem DP não funciona em cenários de aprendizado *online*. Métodos Monte Carlo, por outro lado, têm a vantagem de não exigirem modelos completos do ambiente e de podem funcionar *online*. Entretanto, precisam esperar até o final de um episódio para obterem melhorias na política, e não podem, portanto, aproveitar-se de estimativas parciais.

O métodos de *Diferença Temporal* (TD, do inglês *Temporal Difference*), descritos a seguir, atuam como um meio termo entre as abordagens DP e MC: podem tanto aprender com base em experimentação direta, sem modelo, quanto fazer *bootstrap* a fim de acelerar a obtenção de políticas.

2.6.1 Atualização TD

A atualização de valores por diferença temporal tenta resolver uma das limitações das abordagens MC, isto é, a demora em propagar atualizações nos valores. Ao invés de esperar até o fim de cada episódio, coletando recompensas recebidas após a primeira-visita a cada estado, como ocorre nos métodos MC, é possível atualizar V^π após a observação de *cada recompensa individual*.

As regras TD baseiam-se na idéia geral apresentada na equação (2.1). Informalmente, novas estimativas do valor de um estado são formadas corrigindo-se as estimativas antigas em direção a um *valor alvo*, que em geral equivale à soma da amostra de recompensa recém recebida, e do valor descontado para a estimativa do estado seguinte. Aqui, a idéia chave é que a soma $r + \gamma V(s')$ equivale a uma *amostra* do valor real de $V(s)$. Digamos que um agente, no tempo t, escolhe determinada ação e observa uma transição para o

estado s_{t+1}, com recompensa r_{t+1}. A partir disso, ele pode repassar essas informações (efetuar um backup) e corrigir o valor do estado predecessor, s_t. O método mais simples para fazer esse tipo de backup se chama *TD(0)*, e é formulado da seguinte maneira:

$$V(s)_t \leftarrow V(s)_t + \alpha\left(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right) \tag{2.9}$$

Nessa equação, o *alvo*, dado um passo de correção α, é $r_{t+1} + \gamma V(s_{t+1})$, ou seja, é em direção a esse valor que corrigimos a estimativa atual $V(s)_t$. Perceba que essa equação possui tanto características MC, pois contém uma amostra real de recompensa, quanto de DP, pois faz *bootstrap* com o valor estimado do estado seguinte, s_{t+1}. Ao contrário do DP, entretanto, métodos de diferença temporal não fazem backups completos, pois suas estimativas não se baseiam no valor de todos possíveis estados sucessores, e sim apenas em uma amostra de estado sucessor. O fato de poderem fazer correções incrementais, ao contrário dos métodos MC, que precisam esperar até o final de um episódio, é importantíssimo em tarefas com episódios muito longos, onde, mesmo já tendo recebido informações úteis, o agente precisa seguir um longo período de tempo com uma política que já poderia ter sido melhorada de forma incremental. Além disso, muitas tarefas não podem ser divididas naturalmente em episódios, e para essas o uso de Monte Carlo se torna inviável.

Caso atualizemos o valor de V^π de acordo com a equação (2.9), dada uma política e um ciclo de experimentações do agente (observações de estado-recompensa, tomada de decisão, e assim por diante), garante-se que há convergência com probabilidade 1 para o valor correto de V, desde que o passo de correção α decaia com o tempo[6].

2.6.2 SARSA: controle *on-policy*

O TD(0), apresentado na equação (2.9), permite que aproximemos o valor de V a partir de uma seqüência de experimentações. Entretanto, a fim de obter uma política, precisamos de algum método de *controle* e não apenas de estimativas da função de valor. O primeiro método de controle TD que iremos apresentar baseia-se em aprendizado *on-policy*, ou seja, aprende exatamente a mesma política que segue. Esse método se chama SARSA e foi proposto por (SUTTON, 1984). Dado que nas abordagens TD não temos acesso a um modelo do ambiente, iremos, como no caso do controle MC, estimar a função Q, e não V.

A estratégia geral já é conhecida: iremos estimar Q^π para a política π sendo seguida, e, ao mesmo tempo, corrigir a política para ser gulosa em relação a π.

A regra de atualização SARSA é dada por[7]:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right) \tag{2.10}$$

onde a_{t+1} é a ação que será escolhida no estado sucessor, s_{t+1}, dada a mesma política que gerou a ação a_t. Uma vez que a política sendo aprendida é exatamente a política que está sendo utilizada para definir as ações do agente, dizemos que o SARSA é *on-policy*. Em geral, a política aprendida pelo SARSA é ϵ-gulosa, e é seguida a fim de garantir que haja exploração suficiente de pares estado-ação. Para garantir a convergência, exige-se

[6] A condição exata exige que $\sum_{t=1}^{\infty} \alpha_t = \infty$, mas também que $\sum_{t=1}^{\infty} \alpha_t^2 < \infty$.

[7] Note que o nome do método – SARSA – deve-se aos valores envolvidos na equação: $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$.

adicionalmente que a taxa de exploração decaia com o tempo, ou seja, que aos poucos a política ϵ-gulosa se torne puramente gulosa em relação a Q.

2.6.3 Q-Learning: controle *off-policy*

Uma das mais importantes descobertas para a área de RL diz respeito ao método de aprendizado TD conhecido como Q-Learning (WATKINS; DAYAN, 1992). A equação de ajuste TD do Q-Learning é dada simplesmente por:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right) \tag{2.11}$$

Note que essa formulação difere da equação (2.10) quanto ao alvo da atualização TD. Enquanto que no SARSA o valor estado-ação é corrigido em direção ao valor Q que obteríamos se seguíssemos a mesma política, no Q-Learning o alvo é o valor Q que obteríamos caso seguíssemos a política gulosa. Em outras palavras, independentemente da política seguida durante o aprendizado Q-Learning, a política aprendida é a gulosa. Por essa razão, dizemos que o Q-Learning implementa aprendizado *off-policy*. O pseudo-código de um agente Q-Learning é apresentado no algoritmo 6.

Algoritmo 6 Controle por Q-Learning

Seja $Q(s, a)$ inicializado arbitrariamente, para todo s e a

Repita
  Defina um estado inicial s
  Repita
    Escolha a de acordo com uma política derivada de Q (ex: ϵ-gulosa)
    Execute a e observe a recompensa r e o estado sucessor, s'
    $Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$
    $s \leftarrow s'$
  até que s seja um estado terminal
fim repita

O Q-Learning é um dos algoritmos mais utilizados atualmente, tanto pela sua simplicidade quanto pelo fato de que consegue aprender a política gulosa *independentemente* da política sendo seguida pelo agente. Isso significa que o agente poderia, inclusive, seguir uma política aleatória, e mesmo assim a política ótima seria aprendida; de maneira semelhante, o agente poderia usar sua experiência e, através da idéia de correções *off-policy*, usar as observações do ambiente para atualizar várias políticas simultaneamente[8].

Dado que a política que guia o agente garanta que todos os pares de estado-ação vão ser visitados continuamente, é possível provar que o Q-Learning converge com probabilidade 1 para os valores corretos Q^* (TSITSIKLIS, 1994). Note que essa exigência às vezes é impraticável para sistemas com um grande números de estados. Uma vez que os *backups* corrigem apenas o estado antecessor, pode ser necessário um grande número de experiências próximas a determinada trajetória, até que um valor seja propagado suficientemente para trás, até um estado inicial, por exemplo. Esta limitação será parcialmente

[8]Isso é uma técnica comum em sistemas de aprendizado por reforço hierárquico, onde é preciso aprender, simultaneamente e com base nas experiências de apenas um agente, políticas distintas para várias subtarefas.

superada pelos métodos $TD(\lambda)$, os quais aperfeiçoam os métodos de diferença temporal de um passo (como SARSA e Q-Learning) de forma a torná-los mais próximos das abordagens Monte Carlo. O $TD(\lambda)$ será comentado na seção 2.7.

2.6.4 Métodos Ator-Crítico

Lembre-se que a proposta inicial de aprendizado temporal, dada pela equação (2.9), previa apenas a correção de um passo para a função V. Entretanto, ao contrário do SARSA, por exemplo, no qual tanto a política quanto a função de valor são dadas por Q, não é possível definir uma política ótima conhecendo-se *apenas* V. Uma possível solução é obter um modelo do ambiente, e então proceder de acordo com a equação (2.6). Entretanto, no âmbito dos métodos TD, estamos supondo que não há modelos disponíveis. A outra alternativa é implementar algo parecido com os Métodos de Perseguição, apresentados na seção 2.2.3.3.

A idéia básica dos métodos de Ator-Crítico (AC) é manter duas estruturas de memória separadas:

1. uma estrutura para *controle da política*, independente da função de valor;
2. uma estrutura para *estimação da função de valor*, dada a política.

A primeira estrutura de memória é chamada de *ator*, pois é usada para selecionar as ações do agente. A segunda estrutura é chamada de crítico, pois "critica" (avalia) a política do ator. A crítica, nesse caso, consiste na emissão de um erro de diferença temporal. O aprendizado ocorre de forma cíclica: o ator seleciona uma ação conforme sua política; a ação gera um resultado no ambiente, na forma de uma recompensa, e uma transição de estado. Com base nisso, o crítico corrige sua estimativa de V e emite um sinal de erro TD para o ator. Por fim, o ator usa esse sinal de erro para corrigir sua política, e assim por diante (figura 2.2).

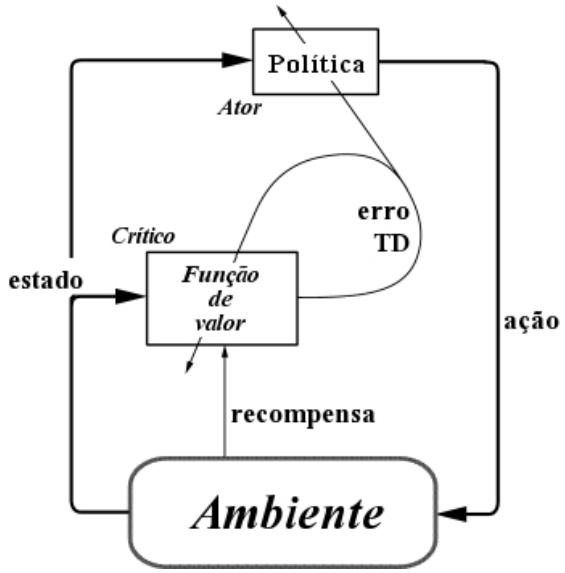

Figura 2.2: Arquitetura tradicional de um sistema Ator-Crítico (adaptada de (SUTTON; BARTO, 1998))

O erro TD emitido pelo crítico pode ser na forma:

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

considerando o caso em que o crítico observa a recompensa r_{t+1} e a transição para o estado s_{t+1}. Após a correção da estimativa de $V(s)$, o erro δ é informado para o ator, que corrige sua política.

Suponha que a seleção de ações do ator é feita de acordo com parâmetros de *preferência* p, como nos Métodos de Perseguição:

$$\pi_t(s,a) = Pr(a_t = a | s_t = s) = \frac{e^{p(s,a)}}{\sum_b e^{p(s,b)}}$$

onde $p(s,a)$ é a preferência por escolher a em s. Nesse caso, o ator poderia adaptar sua política através da atualização do parâmetro p, usando o erro TD gerado pelo crítico:

$$p(s_t, a_t) \leftarrow p(s_t, a_t) + \beta\delta_t$$

Outra formulação possível é aquela que atualiza p de forma inversamente proporcional à probabilidade de selecionar a:

$$p(s_t, a_t) \leftarrow p(s_t, a_t) + \beta\delta_t \left(1 - \pi_t(s_t, a_t) \right)$$

Historicamente, os primeiros métodos de RL foram do tipo Ator-Crítico; com o tempo, acabou-se dando preferência para os métodos que estimam diretamente os valores de estado-ação. Entretanto, uma das grandes vantagens do modelo AC é que é muito fácil escolher a ação a ser tomada. Métodos que não armazenam a política em uma estrutura separada inevitavelmente precisam percorrer e comparar valores de estado (ou estado-ação), a fim de selecionar o mais promissor; no caso de muitas (ou até mesmo infinitas) ações, isso claramente se torna inviável. Por outro lado, pode ser difícil ajustar as taxas de aprendizado do Ator e do Crítico para que os dois convirjam simultaneamente (SUTTON; BARTO, 1998). Além disso, algoritmos *off-policy*, como o Q-Learning, ao contrário do AC, não têm suas condições de convergência afetadas pela estratégia de exploração.

2.7 Traços de elegibilidade

Conforme mencionado anteriormente, o Q-Learning pode demorar bastante até convergir porque, a cada passo, o valor de um estado é transferido (*backed-up*) apenas para o estado imediatamente anterior.

Uma possível solução para este problema é o uso dos chamados traços de elegibilidade. Esses traços são basicamente indicadores de quão suscetível cada estado estará para ser corrigido pelo erro de diferença temporal; quanto mais recentemente o estado tiver sido visitado, maior sua elegibilidade. Na prática, os traços de elegibilidade criam uma marcação dos estados visitados em uma trajetória, de forma que a intensidade da marcação é tanto maior quanto mais recentemente o estado tiver sido visitado. Em termos teóricos, os traços de elegibilidade atuam como marcadores para distribuição de crédito (mensurado na forma de um erro TD), para estados visitados. Numericamente, os traços podem ser vistos como *registros temporais* das ocorrências de um estado, ou par estado-ação.

Os traços de elegibilidade resolvem um dos motivos que levam às baixas velocidades de convergência de algoritmos como o Q-Learning ou SARSA. Mais especificamente, os

traços de elegibilidade solucionam a limitação de atualizações de passo único, ou seja, onde apenas o estado imediatamente anterior tem seu valor corrigido. Imagine, por exemplo, que um "estado objetivo" s_G (estado de alta recompensa) tenha sido atingido após uma trajetória de estados $s_1, s_2, \ldots, s_{n-1}, s_n, s_G$. Depois da visita a s_G, s_n terá seu valor corrigido. Na hipótese do agente seguir exatamente a mesma trajetória por uma segunda vez, será a vez do valor de s_{n-1} ser atualizado, e assim por diante. Com o uso de traços de elegibilidade, por outro lado, todos os estados da trajetória são "marcados" de acordo com algum grau de intensidade (nível de elegibilidade para atualização). A cada passo de atualização, o erro de diferença temporal é usado para corrigir todos estados anteriores, sendo essa correção tanto maior quanto for a elegibilidade do estado em questão. Um exemplo do efeito do uso de traços de elibigidade pode ser visto na figura 2.3.

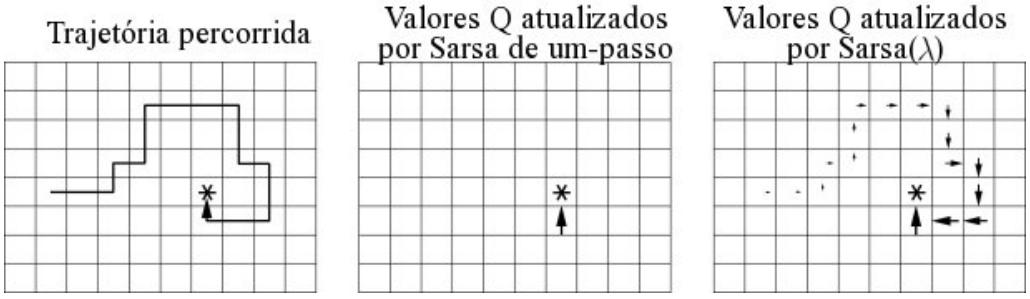

Figura 2.3: Exemplificação do funcionamento dos traços de elegibilidade.

Note que se *não* fizermos os traços diminuirem de intensidade após a última visita do estado, então cada passo de iteração corresponderia a atualizar igualmente todos os estados passados no passado. Se lembrarmos que os algoritmos MC atualizam cada estado visitado em direção à recompensa total até o final do episódio, e que os métodos TD atualizam cada estado em direção apenas à recompensa do próximo estado, fica claro que atualizações TD e MC estão em duas extremidades de um espectro. As abordagens que usam traços de elegibilidade estão no meio esse espectro: nem todos os estados são atualizados a cada tempo, e nem apenas o anterior. Empiricamente, se sabe que métodos que propagam as correções TD para um número intermediário de estados (nem todos, e nem apenas um), possuem desempenho melhor do que as abordagens de extremos (SUTTON; BARTO, 1998).

2.7.1 TD(λ)

A classe de algoritmos TD(λ) diz respeito a todos métodos de diferença temporal que utilizam traços de elegibilidade e que empregam um parâmetro λ, sendo $0 \leq \lambda \leq 1$, a fim de controlar o decaimento dos traços. Dizemos que, a cada tempo t, o traço de elegibilidade de um estado s é $e_t(s)$. Após cada iteração, a elegibilidade de todos estados é decaída de acordo com o parâmetro λ e com a taxa de desconto definida. Além disso, a elegibilidade do estado recém visitado é incrementada de uma unidade.

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{se } s \neq s_t \\ \gamma\lambda e_{t-1}(s) + 1 & \text{se } s = s_t \end{cases} \tag{2.12}$$

O comportamento do traço e é decair lentamente sempre que um estado não for visitado, e aumentar em uma unidade a cada nova visita. A versão padrão para o TD(λ) é

apresentada no algoritmo 7. Perceba que, até agora, estamos apenas preocupados com a estimativa de V, e não com tarefas de controle, já que, tendo apenas a função de valor de estado, precisaríamos ainda de um modelo de transições e de recompensas para calcular a política ótima (equação (2.4)).

Algoritmo 7 Algoritmo TD(λ)

Seja $V(s)$ inicializado arbitrariamente para todo $s \in S$

Repita
  Defina um estado inicial s
  Repita
    Escolha a de acordo com uma política π
    Execute a e observe a recompensa r e o estado sucessor, s'
    $\delta = r + \gamma V(s') - V(s)$
    $e(s) \leftarrow e(s) + 1$
    para todo $s \in S$ **faça**
      $V(s) \leftarrow V(s) + \alpha\delta e(s)$
      $e(s) \leftarrow \gamma\lambda e(s)$
    fim para
    $s \leftarrow s'$
  até que s seja um estado terminal
fim repita

2.7.2 SARSA(λ): controle com traços de elegibilidade

A primeira maneira de adaptar o algoritmo TD(λ) para tarefas de controle é aprender valores de estado-ação, e não valores de estado. O algoritmo que usaremos para fazer essa adaptação se chama SARSA(λ), e é equivalente ao algoritmo tradicional, *on-policy*, SARSA. A diferença essencial para o TD(λ) é que armazenaremos traços de elegibilidade para cada par estado-ação. A atualização será no formato:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t e_t(s,a)$$

onde

$$\delta_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)$$

O traço de elegibilidade é atualizado conforme a equação (2.12). O restante do algoritmo é semelhante ao TD(λ), exceto que as atualizações de $V(s)$ são substituídas por atualizações de $Q(s,a)$.

Um algoritmo semelhante, chamado Q(λ), consiste na adaptação do Q-Learning para o uso de traços de elegibilidade. Entretanto, por ser *off-policy*, precisamos ter o cuidado de zerar os traços de elegibilidade de ações que não correspondem às ótimas. Em outras palavras, dado que o Q-Learning aprende a política gulosa, e não a política sendo seguida, não podemos propagar o erro TD quando acontecem explorações, pois senão estaríamos corrigindo valores estado-ação em direção aos alvos errados. Para uma discussão aprofundada, veja (SUTTON; BARTO, 1998).

2.8 Aprendizado por Reforço com modelos

No âmbito de métodos de programação dinâmica, dizemos haver planejamento, e não aprendizado, pois todas informações do mundo já estão disponíveis para o agente na forma de modelos completos de transições e recompensas. Basta, então, que o agente utilize essas informações para encontrar a estratégia ótima de ações, sem que para isso seja necessária nenhuma atuação real no ambiente. Por essa mesma razão, o uso de modelos pode ser um grande auxiliar para o aprendizado, visto que permite ao agente complementar suas estimativas através de "simulações mentais". Em cenários onde o risco de explorar ações é muito grande, podemos ganhar muito incrementando os dados de experimentação com informações de simulação de modelos.

Até agora discutimos o uso de modelos apenas no contexto de planejamento puro, ou seja, de métodos iterativos onde progressivamente se aproxima o valor de V com base em valores de estados sucessores. Por outro lado, também podemos fazer uso de modelos em técnicas de aprendizado por diferenças temporais; para isso, basta que as ações do agente sejam simuladas no modelo de transições disponível, e que as transições de estado e recompensas recebidas também sejam amostradas a partir destes modelos. A idéia geral é ilustrada no diagrama a seguir:

$$\text{modelo} \rightarrow \text{experiência simulada} \overset{backups}{\rightarrow} \text{estimativa } V(s) \rightarrow \text{modelo } + V = \text{política}$$

Uma situação particular em que essa idéia pode ser aplicada é no uso de amostras aleatórias dos modelos de transição e de recompensas, a fim de obter dados de treinamento do Q-Learning, substituindo, dessa forma, a necessidade de experimentação real. Esse método, que mistura treinamento por diferença temporal e amostragem de modelos, se chama Q-Planning, e é descrito no algoritmo 8.

Algoritmo 8 Q-Planning

1: **Repita**
2: Escolha s e a aleatoriamente de S e A, respectivamente
3: Obtenha uma amostra de próximo estado s' de acordo com a distribuição $T(s, a, \cdot)$
4: Obtenha uma amostra de recompensa r de acordo com a distribuição $R(s, a)$
5: Utilize (s, a, s', r) como dados de treinamento para o Q-Learning:
$Q(s, a) = Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a)$
6: **fim repita**

2.8.1 Dyna-Q: integrando planejamento, ação e aprendizado

Cada vez que um agente efetua uma ação real no ambiente, observa novas informações acerca da dinâmica de transições e de recompensas. Essas informações podem ser usadas de duas formas: *1)* para atualizar/corrigir os modelos T e R; e *2)* para efetuar aprendizado direto, como visto nas seções anteriores. A partir de um modelo, o agente pode tanto efetuar aproximações de V via métodos de programação dinâmica, ou, conforme sugerido pelo algoritmo Q-Planning, utilizar o modelo para atualizar a função de valor através de simulações.

Um algoritmo, chamado Dyna-Q (SUTTON, 1990) integra todos os processos citados acima: planejamento, através de um modelo, ações e experimentações reais, e aprendizado de modelos do ambiente. Todos esses processos ocorrem simultaneamente. O

planejamento, no Dyna-Q, diz respeito ao treinamento utilizando resultados de ações simuladas através de um modelo do ambiente. As ações efetuadas no mundo real, por sua vez, servem tanto para atualizar os modelos T e R do agente quanto para corrigir a função de estado-ação. O processo completo de aprendizado Dyna-Q é apresentado no algoritmo 9. Observe que, para cada correção de Q devido a uma experiência real, N outras correções também são feitas, (em pares (s, a) aleatórios), utilizando experiências simuladas a partir do modelo.

Algoritmo 9 Dyna-Q

Seja $Q(s, a)$ inicializado de forma arbitrária, para todo s e a
Seja $\hat{T}(s, a, \cdot)$, o modelo de transições, inicializado de forma uniforme
Seja $\hat{R}(s, a)$, o modelo de recompensas, inicializado arbitrariamente (ex: com zeros ou com valores otimistas)

Repita
  Escolha a de acordo com uma política π (ex: ϵ-gulosa)
  Execute a e observe a recompensa r e o estado sucessor, s'
  Atualize $\hat{T}$ para considerar a transição $(s, a) \rightarrow s'$
  Atualize $\hat{R}$ para considerar a recompensa $(s, a) \rightarrow r$
  $Q(s, a) \leftarrow \hat{R}(s, a) + \gamma \sum_{s'} \hat{T}(s, a, s') \max_{a'} Q(s', a')$
  para todo k variando de 1 até N **faça**
    $s_k \leftarrow$ estado aleatório
    $a_k \leftarrow$ ação aleatória
    $Q(s_k, a_k) \leftarrow \hat{R}(s_k, a_k) + \gamma \sum_{s'_k} \hat{T}(s_k, a_k, s'_k) \max_{a'_k} Q(s'_k, a'_k)$
  fim para
fim repita

No algoritmo 9, $\hat{R}(s, a)$ corresponde a uma *amostra* da distribuição estimada $\hat{R}$, para a ação a tomada em s. Da mesma forma, $\hat{T}(s, a, s')$ equivale à *probabilidade estimada* pela distribuição $\hat{T}$ para a transição $(s, a) \rightarrow s'$. Perceba também que o algoritmo Dyna-Q utiliza uma regra de atualização dos valores Q que é semelhante à atualização de iteração de valor (DP) para Q. Ao contrário do Q-Learning, essa regra de correção é do tipo *backup completo*, pois considera não apenas uma amostra de transição, e sim todos possíveis estados sucessores de acordo com $\hat{T}$.

As últimas N correções efetuadas pelo Dyna-Q são usadas para atualizar os valores Q por meio de uma "simulação mental"do que ocorreria se um dado par aleatório de estado-ação realmente tivesse sido presenciado pelo agente. Esse tipo de atualização por simulação sempre leva em conta o *modelo corrigido pela experiência real*; por essa razão, dizemos que o Dyna-Q consegue unificar tanto abordagens baseadas em aprendizado experimental quanto técnicas de planejamento com modelo. Embora o Dyna-Q exija aproximadamente N vezes mais esforço computacional do que o Q-Learning, tipicamente consegue atingir a política ótima em tempos menores por um fator de até uma ordem de magnitude. A figura 2.4 apresenta um exemplo de aplicação do Dyna-Q em um cenário de treinamento em labirinto. O estado inicial é marcado por "S", e o estado final (objetivo) por "G". O agente tem como possibilidades de ação se movimentar nas quatro direções cardinais, e todas transições, exceto as que levam a "G", possuem recompensa zero. Atingir o objetivo tem recompensa +1. A figura 2.4 apresenta a comparação de desempenho para o Dyna-Q sem planejamento ($N = 0$) e com planejamento ($N = 50$). As setas indicam a política gulosa aprendida por cada agente durante o segundo episódio

de treinamento. Note que utilizando $N = 50$, a informação obtida ao final do primeiro episódio, isto é, de onde se encontra o objetivo, é rapidamente comunicada a vários outros locais do estaço de estados, enquanto que com $N = 0$ apenas o estado imediatamente anterior ao objetivo é atualizado.

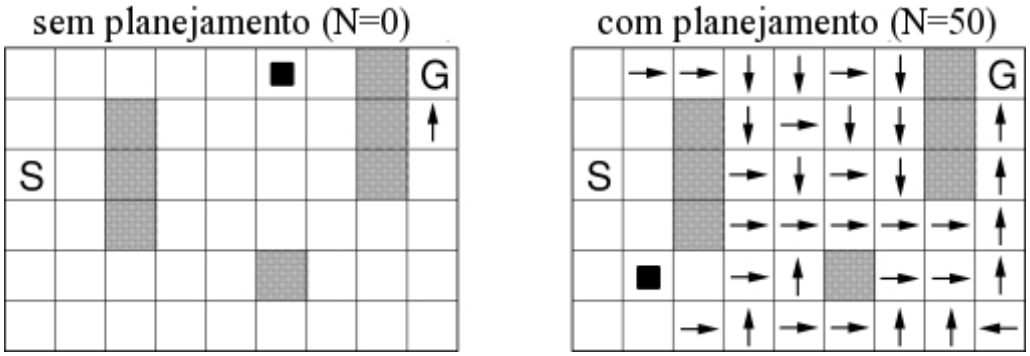

Figura 2.4: Dyna-Q em um cenário de labirinto: aprendizado puramente experimental vs planejamento. Adaptado de (SUTTON; BARTO, 1998).

2.8.2 Prioritized Sweeping: focando o planejamento

Embora o Dyna-Q consiga prover melhores tempos de convergência, por utilizar planejamento baseado em modelo juntamente com o aprendizado por experiência, as N correções que são feitas dizem respeito a pares de estado-ação aleatórios. Claramente, ao descobrir um estado com recompensa muito diferente da esperada, o agente deveria comunicar essa diferença preferencialmente para seus estados vizinhos. Quando a quantidade de estados cresce, diminui de forma drástica a chance de que uma atualização aleatória afete um par estado-ação relevante, isto é, um par estado-ação cujo valor Q deveria ser corrigido de forma significativa.

Uma heurística que pode ser utilizada para acelerar a propagação de correções em valores Q é priorizar a atualização de estados que levam, com grande probabilidade, para o estado recém-atualizado. Mais ainda, podemos fazer com que essa priorização dependa também da magnitude da correção recém efetuada; dessa maneira, ao descobrir valores Q "inesperados", o agente vai corrigir prioritariamente os pares estado-ação que levam com alta probabilidade ao “estado surpreendente”. A fim de implementar essa estratégia, o algoritmo *Prioritized Sweeping* (PS) mantém uma pilha que armazena pares estado-ação cujo valor supostamente será alterado de forma significativa caso receba um backup de seus estados sucessores (MOORE; ATKESON, 1993). A pilha é ordenada de forma a priorizar estados tanto maior for a mudança de valor que sofreriam caso fossem atualizados. Estados no topo da pilha supostamente possuem maior urgência em receberem backups de atualização, uma vez que algum de seus estados sucessores teve seu valor corrigido de forma inesperada. O Prioritized Sweeping é descrito no algoritmo 10.

Perceba que existe uma semelhança entre a idéia do Prioritized Sweeping e dos traços de elegibilidade, na medida em que são utilizadas técnicas para acelerar a propagação de nova estimativas V para outros estados. A diferença essencial é que os traços de elegibilidade servem par atualizar estados que foram visitados recentemente, em uma trajetória específica. O PS, por outro lado, propaga as novidades de V para todos os estados predecessores cujo valor seria bastante corrigido pelo novo conhecimento. Em outras palavras, os traços de elegibilidade estão relacionados com a “profundidade” dos

Algoritmo 10 Prioritized Sweeping

Seja $V(s)$ inicializado de forma arbitrária, para todo $s \in S$

Seja $\hat{T}(s, a, \cdot)$, o modelo de transições, inicializado de forma uniforme

Seja $\hat{R}(s, a)$, o modelo de recompensas, inicializado arbitrariamente (ex: com zeros ou com valores otimistas)

Seja $prioridade(s)$, uma pilha contendo a prioridade de cada estado $s \in S$, inicializada com zeros

Seja N o número de estados de alta prioridade que serão atualizados a cada iteração do algoritmo

1: **Repita**

2: Escolha a de acordo com uma política π (ex: ϵ-gulosa)

3: Execute a e observe a recompensa r e o estado sucessor, s'

4: Atualize $\hat{T}$ para considerar a transição $(s, a) \rightarrow s'$

5: Atualize $\hat{R}$ para considerar a recompensa $(s, a) \rightarrow r$

6: $V_{antigo}(s) = V(s)$

7: Ajuste o valor $V(s)$:

$$V(s) = max_a \left(\hat{R}(s, a) + \gamma \sum_{s'} \hat{T}(s, a, s') V(s') \right)$$

8: $prioridade(s) = 0$

9: Calcule a magnitude da variação do valor do estado: $\delta = |V_{antigo} - V(s)|$

10: Use δ para modificar a prioridade de todos predecessores s_p de s:

$$prioridade(s_p) = \delta \hat{T}(s_p, a_p, s)$$

sempre que a nova prioridade, $prioridade(s_p)$, exceder o valor de prioridade atual; a_p é a ação que maximiza $\hat{T}(s_p, a_p, s)$.

11: **para todo** k variando de 1 até N **faça**

12: Remova o estado j situado no topo da pilha de prioridades;

13: Ajuste o valor $V(j)$:

$$V(j) = max_a \left(\hat{R}(j, a) + \gamma \sum_{s'} \hat{T}(j, a, s') V(s') \right)$$

14: $prioridade(j) = 0$

15: **fim para**

16: **fim repita**

backups em uma trajetória de estados; o PS, por outro lado, está relacionado com o foco para onde os backups serão dirigidos.

2.9 Lidando com a não-estacionariedade

Conforme mencionado no início deste trabalho, cenários não-estacionários obrigam o agente a constantemente reaprender políticas que já haviam sido calculadas. As principais conseqüências disso são: **1)** o tempo necessário para reaprender o modelo e a política faz com que o desempenho do agente caia durante essa fase de reajuste; e **2)** o sistema, ao aprender a nova política, esquece a anterior, e consequentemente faz com que o processo de reaprendizado seja necessário mesmo para dinâmicas que já haviam sido experimentadas. Existem várias maneiras de lidar com estes problemas. Uma delas é simplesmente não se preocupar com o tempo de readaptação do agente, e simplesmente ajustar os parâmetros dos métodos RL para garantir que ao menos as estimativas de valor de estado consigam ser atualizadas constantemente, mesmo face a grandes mudanças. Embora essa solução pareça simples e evidente, observe o que ocorre quando atualizamos a média móvel h, dada uma seqüência de amostras. Para cada nova amostra i:

$$h^{t+1} = h^t + \frac{1}{N+1}(i - h^t)$$

onde N é o número total de amostras observadas. Claramente, conforme N cresce, h é atualizado cada vez menos. Imagine um ambiente não-estacionário em que há alternância entre dois regimes de funcionamento, r_1 e r_2. Após experimentar r_1 por um número grande de amostras, digamos, 1.000.000, estimamos a média móvel de um parâmetro h. Ao ocorrer a troca para o regime r_2, as novas amostras de h fariam com que a média fosse corrigida por um fator de apenas $\frac{1}{1000000}$, ou 0.0001%. Em outras palavras, o sistema nunca conseguiria se recuperar e obter novas estimativas (corretas) para a média de h, após uma troca de regime. Uma solução óbvia para este problema seria limitar (truncar) o valor de N, efetivamente considerando apenas algumas das últimas amostras. A este tipo de solução, ou seja, àquelas que lidam com ambientes não-estacionários apenas modificando e ajustando os parâmetros tradicionais dos algoritmos de RL, chamamos de *abordagens baseadas no ajuste de parâmetros*.

Uma maneira de implementarmos a idéia acima mencionada, ou seja, de considerarmos mais fortemente apenas as últimas amostras de experiência, consiste em mantermos a taxa de atualização de h fixa, ao invés de utilizarmos o fator decrescente $\frac{1}{N+1}$. Suponha que estejamos corrigindo, após N experiências do agente, o valor Q_t em direção à última recompensa observada, r. Suponha também que estejamos utilizando um fator constante α para atualizar Q_t ($0 < \alpha <= 1$). Nesse caso, teremos:

$$\begin{aligned}
Q_t &= Q_{t-1} + \alpha\Big(r_t - Q_{t-1}\Big) \\
&= \alpha r_t + (1-\alpha)Q_{t-1} \\
&= \alpha r_t + (1-\alpha)\alpha r_{t-1} + (1-\alpha)^2 Q_{t-2} \\
&= \alpha r_t + (1-\alpha)\alpha r_{t-1} + (1-\alpha)^2 \alpha r_{t-2} + \cdots + (1-\alpha)^{t-1}\alpha r_1 + (1-\alpha)^t Q_0 \\
&= (1-\alpha)^t Q_0 + \sum_{i=1}^{t} \alpha(1-\alpha)^{t-i} r_i
\end{aligned}$$

Note que a soma dos pesos envolvidos na equação acima é igual a um, o que condiz com o fato de que, ao utilizarmos um valor de α constante, estamos efetivamente atualizando Q através de uma média ponderada. A soma dos pesos envolvidos na equação é dada por S:

$$
\begin{aligned}
S &= (1-\alpha)^k + \sum_{i=1}^{k} \alpha(1-\alpha)^{k-i} \\
&= (1-\alpha)^k + \alpha \sum_{i=1}^{k} (1-\alpha)^{k-i} \\
&= (1-\alpha)^k + \alpha \Big[(1-\alpha)^0 + \sum_{i=1}^{k-1} (1-\alpha)^i \Big] \\
&= (1-\alpha)^k + \alpha * 1 + \alpha \sum_{i=1}^{k-1} (1-\alpha)^i \\
&= \Big[(1-\alpha)^k + \alpha \Big] + \alpha \left[\frac{(1-\alpha)\Big((1-\alpha)^{k-1} - 1 \Big)}{(1-\alpha) - 1} \right] \\
&= \Big[(1-\alpha)^k + \alpha \Big] + \alpha \left[\frac{(1-\alpha)(1-\alpha)^{k-1} - (1-\alpha)}{-\alpha} \right] \\
&= \Big[(1-\alpha)^k + \alpha \Big] - \Big[(1-\alpha)^k - 1 + \alpha \Big] \\
&= \Big[(1-\alpha)^k + \alpha \Big] - \Big[(1-\alpha)^k + \alpha \Big] + 1 \\
&= 1
\end{aligned}
$$

Note também que o peso dado para a recompensa r_i, ou seja, $\alpha(1-\alpha)^{k-i}$, depende de quantos passos atrás r_i foi recebida (no caso, $k-i$ passos). Isso implica que os pesos dados para recompensas passadas decaem exponencialmente de acordo com um expoente $(1-\alpha)$.

Devemos notar também que, ao utilizarmos um passo de correção de Q constante, tal como α, perdemos as propriedades de convergência para o valor correto de Q. Isso ocorre pois as estimativas de tal valor variam (flutuam) indefinidamente, sempre em resposta à última recompensa recebida, e nunca convergem para um valor estático. Entretanto, uma vez que estamos tratanto de ambientes não-estacionários, esse tipo de flutuação é de fato algo *bom*, já que torna a abordagem de correção flexível o suficiente para superar as alterações nos valores de Q causadas pela não-estacionariedade (SUTTON; BARTO, 1998).

2.9.1 Aprendizado por Reforço com múltiplos modelos

Assim como o Prioritized Sweeping, apresentado na subseção 2.8.2, podemos imaginar vários outros métodos capazes de utilizar modelos de transição e de recompensas em tarefas de treinamento por reforço. Entretanto, um salto significativo de qualidade de atuação pode ser obtido se, ao invés de utilizarmos apenas *um* modelo para tentar descrever todas as facetas do ambiente, utilizarmos *vários*. Para tarefas complexas e/ou não-estacionárias, essa é uma solução possível. A abordagem MMRL (*Multiple Model-based*

Reinforcement Learning, ou Aprendizado por Reforço baseado em Múltiplos Modelos), proposta por Doya et al (DOYA et al., 2002), se faz valer justamente dessa idéia.

A proposta do MMRL é organizar uma arquitetura de RL capaz de lidar com tarefas não-estacionárias decompondo a representação de um ambiente complexo em vários módulos preditivos simples. A decomposição é efetuada tendo como critério a capacidade de cada módulo de prever um determinado aspecto do ambiente não-estacionário; idealmente, em um ambiente que pode assumir vários regimes de dinâmica, cada módulo do MMRL acabará por se especializar em um regime de comportamento específico. A abordagem MMRL gira em torno de um valor importante, chamado de *valor de responsabilidade*. A cada módulo preditivo, associamos um valor de responsabilidade, a fim de representar a qualidade daquele módulo em prever a última observação da dinâmica do ambiente. Caso um determinado módulo preditivo tenha sucesso em prever o tipo de transição que acontece em um ambiente, possuirá uma responsabilidade alta; caso contrário, terá uma responsabilidade baixa. Uma maneira alternativa de imaginar este mecanismo é pensar que as previsões feitas por cada um dos módulos, no que diz respeito à dinâmica de transições do ambiente, possuem graus diferentes de *qualidade*. Módulos melhores adaptados ao regime de transição sendo observado supostamente são mais pertinentes para o controle do sistema, e portanto possuirão um valor de responsabilidade mais alto. O MMRL calcula este valor de responsabilidade considerando o erro relativo de previsão de cada módulo. Posteriormente, utiliza-se tal valor para modular o treinamento e a atuação do sistema como um todo.

A arquitetura MMRL é composta por um conjunto *fixo* de módulos, cada qual contendo modelos de previsão do ambiente e também um controlador RL. O treinamento do MMRL consiste em decompor a representação dos modelos de previsão e de atuação, para uma determinada tarefa não-estacionária, em vários modelos simples, cada qual descrevendo determinado um tipo de comportamento do ambiente. Quando utilizados em conjunto, os módulos obtêm poder de previsão, e qualidade de atuação, superiores aos que seriam obtidos caso fossem utilizados algoritmos RL baseados em um único modelo, como o Prioritized Sweeping. Os sinais de responsabilidades irão indicar a qualidade relativa de cada módulo em prever a dinâmica atual do ambiente; dessa forma, o MMRL não escolherá e utilizará apenas o módulo de maior qualidade, e sim fará uso de uma medida composta pela saída de todos os módulos, sendo a contribuição de cada um ponderada pela sua responsabilidade.

Embora o MMRL também possua uma formulação para ambientes de estado e tempo contínuos, apresentaremos aqui apenas a formulação para o caso de estado e tempo discretos, por ser esta a base teórica de todos métodos apresentados até agora.

Considere que a arquitetura MMRL aplicada a determinada tarefa é composta por M módulos. Cada módulo possuirá um valor de responsabilidade, λ_i, que irá refletir o quão bem, relativamente aos outros módulos, i consegue prever o ambiente. Considere que o agente tenha vivenciado uma transição do estado s para o estado s', através da ação a. A equação básica para o sinal de responsabilidade pode ser obtida através da regra de Bayes:

$$\lambda_i = P(i \mid s, s') = \frac{P(i \mid s)P(s' \mid s, i)}{\sum_{j=1}^{M} P(j \mid s)P(s' \mid s, j)}$$

onde $P(i \mid s)$ é a probabilidade *a priori* do módulo i ser o melhor previsor para a tarefa, caso i esteja no estado s, $P(i \mid s, s')$ é a probabilidade de i ser o melhor previsor dado que foi observada uma transição de s para s', e $P(s' \mid s, i)$ é a probabilidade de s' ter sido observado após s, dado que o modelo de previsão correto é o do módulo i. A seguir,

iremos reescrever a equação acima para torná-la coerente com a notação utilizada no restante deste trabalho. Iremos chamar, seguindo a notação de Doya (DOYA et al., 2002), os *priors* $P(i)$ por $\hat{\lambda}_i$. Desta forma, obtemos:

$$\lambda_i = \frac{T_i(s,a,s')\hat{\lambda}_i}{\sum_{j=1}^{M} T_j(s,a,s')\hat{\lambda}_j}$$

onde T_i corresponde ao modelo de transições do módulo i. Se tomarmos *priors* uniformes, isto é, iguais a $\frac{1}{M}$, a equação da responsabilidade se resume a uma medida de proporção entre a probabilidade prevista por um modelo e as probabilidades previstas por todos os modelos. Dito de outra forma, a responsabilidade de um módulo pode ser interpretada como a *qualidade relativa instantânea* daquele módulo, considerando seu poder de previsão da última transição observada, e sua probabilidade *a priori* de ser o módulo previsor correto.

Para o caso de *priors* uniformes, a responsabilidade corresponderá a um valor de qualidade instantânea míope, ou seja, que medirá a qualidade relativa de cada módulo considerando *apenas a última observação*, e não um histórico de observações. Caso queiramos fazer com que a responsabilidade reflita também a qualidade do modelo em tempos anteriores, podemos interpretar a responsabilidade do tempo $t-1$ como o *prior* no tempo t, exceto por um fator de decaimento $0 < \alpha < 1$, o qual irá controlar a "taxa de esquecimento":

$$\lambda_i^{t+1} = \frac{T_i(s,a,s')\hat{\lambda}_i^{t+1}}{\sum_{j=1}^{M} T_j(s,a,s')\hat{\lambda}_j^{t+1}}$$

onde o *prior* da probabilidade de cada módulo, $\hat{\lambda}$, é definido por:

$$\hat{\lambda}_k^{t+1} = (\lambda_k^t)^{\alpha}$$

Idealmente, o MMRL utiliza o sinal de responsabilidade para quatro fins:

1. Ponderar a predição do estado feita por cada módulo, a fim de obter uma predição conjunta. A predição feita por cada módulo terá um peso tão grande quanto for a responsabilidade (ou qualidade preditiva) daquele módulo.

2. Ponderar o aprendizado do modelo de transições e de recompensas em cada módulo. Quanto maior for a qualidade preditiva de um módulo, mais queremos aproximar seus modelos da realidade sendo observada. Módulos com responsabilidade baixa provavelmente não são especialistas no tipo de ambiente observado nas últimas transições, e por isso devem ser ajustados com menor intensidade. O efeito final disso é que cada módulo tenderá a se especializar em um tipo de dinâmica do ambiente não-estacionário.

3. Ponderar as ações propostas por cada módulo, para um dado estado s. Em outras palavras, a probabilidade de seleção da ação a no estado s será dada por uma combinação das probabilidades desta seleção serem feitas pelos controladores de cada um dos módulos.

4. Ponderar o aprendizado dos controladores RL de cada módulo. Caso os controladores aprendam por diferença temporal, será calculado um erro TD considerando as previsões V de todo sistema, e então esse erro será usado no aprendizado de cada

módulo de maneira proporcional à responsabilidade daquele módulo. Módulos com qualidade alta serão corrigidos por uma alta porcentagem do erro TD, justamente por provavelmente terem sido os mais "responsáveis" pelas últimas ações do agente.

O primeiro item enumerado, ou seja, a previsão de transição de estado composta pelas previsões de todos os módulos, pode ser descrito pela seguinte equação:

$$T(s,a,s') = \sum_{i=1}^{M} \lambda_i T_i(s,a,s')$$

onde $T(s,a,s')$ é a probabilidade da transição $s \rightarrow s'$, sob ação a, considerando os modelos de transição de todos os módulos.

O segundo caso de utilização do sinal de responsabilidade corresponde à atualização dos modelos de transições. Para cada modelo i, dada a observação de $s \rightarrow s'$ sob a, seguida da recompensa r, a correção do modelo de transições T_i é feita da seguinte forma:

$$T_i(s,a,\kappa) \leftarrow T_i(s,a,\kappa) + \Delta T_i(\kappa), \quad \forall \kappa \in S \tag{2.13}$$

onde

$$\Delta T_i(\kappa) = \begin{cases} \lambda_i\Big(1 - T_i(s,a,\kappa)\Big) & \kappa = s' \\ & \forall \kappa \in S \\ \lambda_i\Big(0 - T_i(s,a,\kappa)\Big) & \kappa \neq s' \end{cases}$$

De forma semelhante, o modelo de recompensas, R_i, do módulo i, é corrigido da seguinte forma:

$$R_i(s,a) \leftarrow R_i(s,a) + \Delta R_i \tag{2.14}$$

onde

$$\Delta R_i = \lambda_i\Big(r - R_i(s,a)\Big)$$

A terceira utilidade do sinal de responsabilidade corresponde à obtenção de uma probabilidade combinada $\pi(s,a)$ de seleção de uma ação a, dado um estado s. Isso é expresso pela equação:

$$\pi(s,a) = \sum_{i=1}^{M} \pi_i(s,a)$$

onde $\pi_i(s,a)$ é a probabilidade do controlador do módulo i executar a quando em s.

A maneira usual de determinar a ação ótima em um controlador RL é escolher a ação que maximiza a soma esperada da recompensa imediata com as recomensas futuras, partindo do estado s:

$$a^*(s) = \arg\max_a E\Big[R(s,a) + \gamma V(s')\Big] \tag{2.15}$$

onde $R(s,a)$ é a previsão combinada da recompensa, $R(s,a) = \sum_{i=1}^{M} \lambda_i R_i(s,a)$. Na equação (2.15), $V(s')$ corresponde ao valor combinado esperado para os estados sucessores de s. O termo da expectativa na equação (2.15) pode então ser expandido para:

$$
\begin{aligned}
E\Big[R(s,a)+\gamma V(s')\Big] &= Q(s,a) \\
&= \sum_{i=1}^{M} \lambda_i \Big(R_i(s,a)+\gamma \sum_{s'} T_i(s,a,s')V_i(s')\Big)
\end{aligned}
$$

e portanto a ação gulosa será dada por:

$$
a^*(s) = \arg\max_a \left(\sum_{i=1}^{M} \lambda_i \Big(R_i(s,a)+\gamma \sum_{s'} T_i(s,a,s')V_i(s')\Big)\right) \tag{2.16}
$$

Por fim, o sinal de responsabilidade também é utilizado para ponderar a distribuição do erro TD entre os módulos. Considere a estimativa composta $V(s)$ como sendo a ponderação das funções de valor $V_i(s)$ de cada módulo:

$$
V(s) = \sum_{i}^{M} \lambda_i V_i(s)
$$

Assim, a função de valor do módulo i, V_i, pode ser corrigida por uma proporção de δ, o erro TD combinado:

$$
V_i(s) \leftarrow V_i(s) + \lambda_i \delta \qquad \forall i \in M
$$

onde δ corresponde ao erro TD relativo aos valores combinados de estado:

$$
\delta = r + \gamma V(s') - V(s)
$$

Uma representação completa da abordagem MMRL, para o caso de estado e tempo discretos, é apresentada no algoritmo 11.

Algoritmo 11 *Multiple Model-based Reinforcement Learning*

Seja M o número total e fixo de modelos. Cada modelo, M_i, é formado pelas seguintes partes componentes:

Seja $V_i(s)$, inicializada de forma arbitrária, para todo $s \in S$ e $i \in M$

Seja $T_i(s, a, \cdot)$, o modelo de transições de M_i, inicializado de forma uniforme para todo $i \in M$

Seja $R_i(s, a)$, o modelo de recompensas de M_i, inicializado arbitrariamente (ex: com zeros ou com valores otimistas) para todo $i \in M$

Seja λ_i, o sinal de responsabilidade do modelo i, inicializado com $\frac{1}{M}$ para todo $i \in M$

Repita

  Defina um estado inicial s

  Repita

    $V(s) = \sum_i^M \lambda_i V_i(s)$

    Escolha a ação gulosa $a^*(s)$ de acordo com a equação (2.16), ou utilize algum outro critério para determinar probabilidades $P(s, a)$ de executar a em s, tal como a seleção softmax:

$$P(s, a) = \frac{e^{\beta Q(s,a)}}{\sum_{a'} e^{\beta Q(s,a')}}$$

    tal que $Q(s, a)$ é como na equação (2.16)

    Execute a ação a escolhida e observe a recompensa r e o estado sucessor s'

    $V(s') = \sum_i^M \lambda_i V_i(s')$

    $\delta = r + \gamma V(s') - V(s)$

    $V_i(s) \leftarrow V_i(s) + \lambda_i \delta \qquad \forall i \in M$

    Corrija os modelos de transição T_i, como na equação (2.13)

    Corrija os modelos de recompensa R_i, como na equação (2.14)

    Considere que o agente atualmente se encontra no tempo discreto t. As responsabilidades λ_i^t de cada modelo i são então atualizadas para:

$$\lambda_i^{t+1} = \frac{T_i(s, a, s')\hat{\lambda}_i^{t+1}}{\sum_{j=1}^M T_j(s, a, s')\hat{\lambda}_j^{t+1}} \qquad \forall i \in M$$

    sendo $\hat{\lambda}_k^{t+1} = \frac{1}{M}$, no caso de *priors* uniformes, ou $\hat{\lambda}_k^{t+1} = (\lambda_k^t)^\alpha$, no caso de *priors* para continuidade temporal dos modelos, para todo $k \in M$.

  até que s seja um estado terminal

fim repita

3 APRENDIZADO POR REFORÇO COM DETECÇÃO DE CONTEXTOS

Com base nas anteriormente citadas dificuldades de se aprender de ambientes não-estacionários, iremos agora propor uma solução baseada no uso de múltiplos modelos. O método em questão, chamado de RL-CD (Reinforcement Learning with Context Detection), é baseado em um mecanismo capaz de aprender, atualizar e selecionar um dentre vários modelos parciais do ambiente[1]. A suposição essencial deste método, que será discutida em detalhes mais adiante, é a de que os ambientes não-estacionários de interesse podem ser descritos por um conjunto finito de regimes estacionários de funcionamento, que se alternam devido a uma causa não-observável. Em outras palavras, o ambiente não-estacionário se encontrará, a cada momento, em um regime de operação específico. Além disso, postulamos que a alteração de regime não poderá ser percebida diretamente pelo agente, sendo essa incapacidade devida, por exemplo, a sensores defeituosos ou a eventos não-locais, portanto fora do alcance dos sensores. A cada um destes regimes estacionários, denominamos **contexto**. O método RL-CD foi apresentado pela primeira vez em (SILVA et al., 2006).

Evidentemente, a alteração do regime de operação do ambiente implica alterações nas matrizes de probabilidades de transição de estados e de recompensas. Como conseqüência, a política ótima também será afetada. Embora estejamos postulando que o agente não possui a capacidade de perceber diretamente a causa da alteração do regime, presumimos que é possível estimar tal alteração e então selecionar um dentre os vários modelos parciais, utilizando para isso os *erros de previsão* do agente.

De forma geral, o que se poderia esperar de um agente inteligente é que, durante sua interação com o ambiente, ele fosse capaz de aprender *modelos de predição*. Conforme mencionado anteriormente, isso não é um fator obrigatório, uma vez que alguns dos resultados mais importantes para a a área de aprendizado por reforço (RL) são justamente os métodos de aprendizado sem modelo. Entretanto, é inegável que o cálculo da política ótima pode ser acelerado enormemente ao se fazer uso de misturas de aprendizado experimental com técnicas de planejamento. Além disso, também existem evidências a favor da hipótese de que o próprio ser humano aprende tanto por diferença temporal quanto por métodos mistos de planejamento com modelo (DOYA, 2002).

Tendo essas colocações em mente, é razoável supor que um agente racional, ao se valer do uso de modelos, deva ser capaz de perceber quando tais modelos não mais corresponderem à realidade observada. O problema passa a ser, então, como determinar *quando* um modelo deixa de ser correto. Essa decisão não é simples, uma vez que a

[1]Neste capítulo, assim como nos anteriores, entendemos um *modelo do ambiente* como sendo formado pela matriz de probabilidades de transição de estados, T, e pela função de recompensas imediatas, R.

qualidade do modelo, no caso geral, não é do tipo perfeita-ou-imperfeita. Mais razoável é supor que existam vários graus de adaptação, uma vez, por exemplo, que um dado modelo pode ser ruim em prever alguns tipos de situações, mas muito bom para todo o restante.

Entendendo que o ajuste de um modelo à realidade é uma medida que admite vários níveis, a formulação do RL-CD será baseada na suposição de que o agente deve ser capaz de estimar a qualidade de cada modelo a partir de um valor numérico que represente o grau de sucesso do modelo em suas previsões, dado um histórico temporal de experiências. O agente, ao perceber que seu modelo de previsão não é eficiente, pode escolher outro mais adequado ou ainda, na ausência deste, criar um novo.

Note que a abordagem MMRL, apresentada no capítulo passado, também é baseada no uso de múltiplos modelos. Entretanto, existem diferenças essenciais entre o MMRL e o RL-CD, e elas serão apresentadas mais adiante. Por ora é suficiente mencionar que, ao contrário do MMRL, nossa proposta busca a construção e atualização *incremental* de modelos. Dito de outra forma, não supomos que o agente deva conhecer *a priori* todos os regimes de funcionamento do ambiente. Será apenas através da experimentação que o agente perceberá a necessidade de criar novos modelos para as situações diversas não explicadas satisfatoriamente através de seu conhecimento atual.

O presente capítulo está organizado da seguinte forma. Na seção 3.1, explicitamos e discutimos as suposições necessárias para que o método proposto seja válido. Na seção 3.2.1, formalizamos a maneira como modelos parciais do ambiente são atualizados conforme o agente acumula novas experiências. Após, na seção 3.2.2, formalizamos o mecanismo de cálculo de qualidade dos modelos parciais, utilizado para detectar trocas de contexto em um ambiente não-estacionário. A partir disso, apresentamos resultados empíricos que demonstram a aplicabilidade do RL-CD, na seção 3.3 e, por fim, discutimos os parâmetros do algoritmo e possíveis melhorias nas seções 3.4 e 3.5.

Antes de iniciarmos a apresentação do método propriamente dito, enumeramos, na tabela 3.1, os símbolos e a nomenclatura utilizados nas seções seguintes. Tais símbolos, assim como seu significado, são resumidos de forma a facilitar posteriores consultas.

Tabela 3.1: Resumo de símbolos utilizados no capítulo 3

Símbolo	Significado
S	Conjunto de estados de um MDP.
A	Conjunto de ações de um MDP.
$R(s,a)$	Estimativa da recompensa imediata após executar a em s.
$R_m(s,a)$	Estimativa de recompensa imediata utilizada pelo modelo m.
$T(s,a,\cdot)$	Estimativa das probabilidades de transição para próximo estado após executar a em s.
$T_m(s,a,\cdot)$	Estimativa de probabilidades de transição utilizadas pelo modelo m.
$\langle s,a,s',r\rangle$	Tupla de experiências que indica que o agente estava em s, executou a e transicionou para s' com recompensa r.
$V^*(s)$	Valor ótimo de um estado (máximo esperado de recompensas futuras partindo do estado s sob a política ótima).
π^*	Política ótima de um MDP.
π^*_m	Política ótima referente ao modelo parcial m.
$\Delta T_m(\kappa)$	Ajuste necessário na probabilidade de transição $T(s,a,\kappa)$, no modelo m, para torná-la coerente com a última observação. Note que s corresponde ao estado atual e a à última ação executada.
ΔR_m	Ajuste necessário na função de recompensas $R_m(s,a)$ para torná-la coerente com a última observação. Note que s corresponde ao estado atual e a à última ação executada.
$N_m(s,a)$	Contabilizador do número de vezes, no modelo m, em que a ação a foi executada em s.
M	Valor de memória. Utilizado para definir o comportamento da taxa de ajuste dos modelos T_m e R_m, e também relacionada com a confiança c_m.
$c_m(s,a)$	Confiança do modelo m nas estimativas de qualidade de previsão calculadas após a visitação do estado s e execução da ação a.
e_m	Qualidade instantânea de previsão. É uma combinação convexa de e^T_m e e^R_m.
e^T_m	Qualidade instantânea de previsão de transições.
e^R_m	Qualidade instantânea de previsão de recompensas.
Ω	Parâmetro que regula a combinação convexa que forma e_m, dando pesos diferentes para a capacidade de prever transições e de prever recompensas.
E_m	Traço temporal de qualidade de previsão do modelo m. Reflete o comportamento da qualidade de previsão instantânea e_m nas últimas experiências.
ρ	Taxa de ajuste de E_m.
E_{min}	Qualidade mínima aceitável. Se o modelo atual m apresentar qualidade $E_m < E_{min}$, um novo modelo será criado.

3.1 Suposições

São duas as suposições principais que norteiam o desenvolvimento do RL-CD:

1. os cenários não-estacionários de interesse podem ser descritos por um conjunto finito de regimes estacionários, os quais serão representados por modelos parciais de previsão;

2. a qualidade relativa de previsão é uma boa métrica para guiar as decisões de criação e seleção de modelos.

A primeira suposição traz consigo algumas restrições importantes quanto aos ambientes nos quais o RL-CD poderá operar. Especificamente, restringimos a classe de ambientes não-estacionários de interesse àqueles em que ocorre a alternância entre vários contextos estacionários distintos. Assumimos, ademais, que os contextos podem ser representados eficientemente por modelos parciais do ambiente, sendo a designação "parcial" devida ao fato de que cada modelo é capaz de explicar a dinâmica observada em apenas uma porção do tempo. Central ao funcionamento do RL-CD também está a suposição de que a observação de transições e recompensas é suficiente para o cálculo de um valor numérico de qualidade do modelo, valor este associado à eficácia de previsão de um modelo para determinado contexto. Idealmente, gostaríamos que cada modelo parcial fosse capaz de assumir para si a tarefa de modelar exatamente um dos contextos existentes.

Este tipo de representação baseada em múltiplos modelos condiz com a abordagem convencionalmente utilizada por pesquisadores dos chamados *Hidden-Mode MDPs* (HM-MDPs), ou Processos de Decisão de Markov com Modo Oculto. Seguindo a terminologia utilizada neste trabalho, um *modo* corresponde a uma variável oculta de estado que determina o MDP específico associado a determinado contexto. Em outras palavras, um modo pode ser relacionado a qualquer deficiência de percepção que o agente possua e que faça com que a dinâmica do mundo lhe pareça não-estacionária. Nos ambientes do tipo HM-MDP, as seguintes propriedades são verdadeiras:

1. mudanças no ambiente são confinadas a um pequeno número de contextos[2], os quais são MDPs com matrizes de probabilidades de transição e recompensas distintas;

2. o contexto atual não pode ser observado diretamente, mas pode ser estimado de acordo com os tipos de transições e recompensas observadas;

3. trocas de contexto são relativamente infreqüentes;

4. as trocas entre contextos são independentes das ações do agente.

A primeira suposição é razoável no sentido em que os ambientes de interesse são apenas aqueles nos quais o contexto é determinado por uma variável de estado não-observável. Caso o número de contextos fosse próximo do número de estados, então as variáveis não-observáveis ocorreriam em grande número, o que caracterizaria a situação como um problema de MDP Parcialmente Observável (POMDP) geral (KAELBLING; LITTMAN; MOORE, 1996). POMDPs representam a formalização utilizada para tratar MDPs em que um número arbitrário de variáveis ocultas pode ocorrer, ou seja, em

[2]Número de contextos $\ll |S|$.

que um número qualquer de observações do agente não necessariamente corresponde aos estados reais do ambiente. Os métodos existentes para lidar com POMDPs apresentam dificuldades em lidar até mesmo com cenário pequenos, com poucas centenas de estados (CASSANDRA; LITTMAN; ZHANG, 1997).

O fato de assumirmos as suposições enumeradas anteriormentes faz com que o tipo de ambiente não-estacionário de interesse constitua, na verdade, um subcaso tratável dentre os POMDPs gerais, o qual é denominado HM-MDP (CHOI; YEUNG; ZHANG, 2000). Embora os HM-MDPs sempre possam ser transformados em POMPDs através do aumento do espaço de estados, POMPDs são muito mais complexos justamente por não assumirem as restrições enumeradas acima. Em resumo, defendemos que a primeira suposição é razoável pois apenas exige que os aspectos não-observáveis do ambiente não sejam arbitrariamente complexos.

A segunda e terceira suposições, por sua vez, estão relacionadas ao fato de assumirmos que o contexto atual *apenas* poderá será estimado através das medidas de qualidade de previsão do agente. Por essa razão, se torna necessário que a alternância entre contextos seja menos freqüente que o tempo de amostragem necessário para obter estimativas confiáveis. Por exemplo, se a variável não-observável que determina a troca de contexto corresponder ao tipo de chão em que um robô se encontra, tal robô nunca conseguirá detectar que saiu do concreto para a grama caso não experimente determinado tempo mínimo de contato com a grama, no qual irá ser perceber que suas expectativas (probabilidades de transição e recompensas) não foram atendidas.

Por fim, a última suposição garante que a alternância entre contextos não poderá ser *prevista* pelo agente, mas apenas detectada após já ter ocorrido. Esta suposição não é essencial aos HM-MDPs, mas facilita seu tratamento na medida em que não obriga o agente a estimar probabilidades de transição *entre modelos parciais*.

No restante deste trabalho, iremos utilizar a mesma notação matemática sugerida no capítulo anterior. Em resumo, dizemos que um Processo de Decisão de Markov consiste em um conjunto discreto de estados do ambiente, S, um conjunto discreto de ações, A, uma função de recompensa $R : S \times A \rightarrow \Re$, e uma função de probabilidades de transição de estados, $T : S \times A \rightarrow \Pi(S)$, tal que $\Pi(S)$ é uma distribuição de probabilidades sobre S. Denotamos por $T(s, a, s')$ a probabilidade de fazer a transição de s para s' tomando a ação a.

Uma tupla de experiência do tipo $\langle s, a, s', r \rangle$ denota o fato de que o agente se encontrava em s, executou a e transicionou para s', recebendo, neste processo, uma recompensa instantânea r. Dizemos que o valor ótimo de um estado é $V^*(s)$ e corresponde ao total esperado de recompensas futuras, considerando um critério de otimalidade de reforços descontados. A política que maximiza simultaneamente a função de valor para todos os estados é denotada por π^*.

3.2 Aprendizado por Detecção de Contextos

Antes de detalhar as equações e o algoritmo do RL-CD propriamente dito, apresentaremos um breve resumo de como ocorre a criação de modelos parciais e a detecção de trocas de contexto. Conforme mencionado anteriormente, assumimos que o sistema de aprendizado contém vários modelos parciais de predição, cada um especializado em um dos regimes de funcionamento do ambiente. A *qualidade* de um modelo será represen-

tada por um valor inversamente proporcional ao erro[3] de predição do modelo. Quanto mais ajustadas forem as previsões em relação às observações reais, maior será o valor de qualidade. A cada momento, apenas o modelo com a maior qualidade será utilizado para determinar as ações ótimas do agente. Uma troca de contexto será detectada sempre que o modelo atual for substituído por outro mais eficiente. Caso a qualidade do melhor modelo de predição ainda seja menor que um limiar mínimo estipulado, então um novo modelo será criado, e este deverá aprender tanto a nova dinâmica do ambiente quanto a respectiva política ótima.

3.2.1 Aprendendo Modelos Parciais

Cada um dos m modelos parciais que compõem o RL-CD irá conter uma função de probabilidade de transição (T_m) e de recompensas imediatas (R_m). O cálculo da política ótima π^*_m induzida por estas funções poderá ser executado por abordagens tradicionais de RL baseadas em modelos, como o Dyna ou Prioritized Sweeping.

A cada passo de interação com o ambiente, associamos uma tupla de experiência denotada por $\langle s, a, s', r\rangle$. Seja m o modelo atualmente em uso, e considere $N_m(s,a)$ um contabilizador do número de vezes, no modelo m, em que a ação a foi executada no estado s. O primeiro passo para o aprendizado deste modelo parcial é atualizar T_m em direção à estimativa de máxima verossimilhança em relação a s', através das seguintes equações:

$$\Delta T_m(\kappa) = \begin{cases} \dfrac{1 - T_m(s,a,\kappa)}{N_m(s,a)+1} & \kappa = s' \\ & \forall \kappa \in S \\ \dfrac{0 - T_m(s,a,\kappa)}{N_m(s,a)+1} & \kappa \neq s' \end{cases}$$

e

$$T_m(s,a,\kappa) = T_m(s,a,\kappa) + \Delta T_m(\kappa), \quad \forall \kappa \in S \tag{3.1}$$

De maneira semelhante, dado que a última recompensa instantânea recebida foi r, o modelo de recompensas R_m é ajustado em direção à média móvel de recompensas passadas através de:

$$\Delta R_m = \frac{r - R_m(s,a)}{N_m(s,a)+1}$$

e

$$R_m(s,a) = R_m(s,a) + \Delta R_m \tag{3.2}$$

Note que ao invés de incrementarmos o contador $N_m(s,a)$ a cada nova visitação do par estado-ação (s,a), computamos seu novo valor utilizando uma *memória finita* (truncada) das últimas M experiências:

$$N_m(s,a) = min\Big(N_m(s,a)+1,\ M\Big) \tag{3.3}$$

Ao truncar N, garantimos que o coeficiente de ajuste de T_m não irá cair para zero conforme o número de experiências cresce. Inicialmente, as estimativas de probabilidades

[3]No texto que segue, subentende-se que níveis altos de erro implicam níveis baixos de qualidade, e vice-versa. Portanto, embora o RL-CD calcule *qualidades*, o termo erro também será usado quando conveniente.

de transição serão ajustadas mais rapidamente, mas a taxa de ajuste irá se estabilizar após M experiências. Caso N não seja limitado como na equação (3.3), então a equação (3.1) implementará exatamente a estimativa de máxima verrossimilhança para o modelo de transições, e a equação (3.2) implementará exatamente a média móvel de recompensas passadas.

3.2.2 Detectando trocas de contexto

A fim de detectar trocas de contexto, o RL-CD precisa ser capaz de avaliar o quão bem o modelo parcial ativo está prevendo a dinâmica do ambiente. Para isso, um *sinal de qualidade* é calculado para cada modelo parcial. A qualidade de um modelo consiste em um valor numérico que reflete o grau de ajuste necessário para que aquele modelo se torne coerente com a última experiência do agente: quanto menor a necessidade de ajustes, maior será a qualidade. O sinal de qualidade também é proporcional a um *valor de confiança*, o qual reflete o número de vezes que um agente tentou uma ação em determinado estado. O valor de confiança é utilizado para evitar que o agente tire conclusões precipitadas sobre a qualidade de um modelo sem antes tê-lo experimentado suficientemente.

Dado um modelo m e uma tupla de experiência $\langle s, a, s', r\rangle$, calculamos a confiança $c_m(s, a)$ e a qualidade instantânea e_m da seguinte maneira:

$$c_m(s,a) = \frac{N_m(s,a)}{M} \tag{3.4}$$

$$e_m = c_m(s,a)\left(\Omega\ \ e_m^R + (1-\Omega)\ \ e_m^T\right) \tag{3.5}$$

onde e_m consiste em uma combinação convexa de dois termos: a qualidade instantânea de previsão de recompensas, e_m^R, e a qualidade instantânea de previsão das transições, e_m^T. Tais valores de qualidade são interpolados linearmente por Ω, parâmetro este que especifica a importância relativa do conhecimento das recompensas e das transição para a estimativa da qualidade do modelo. As equações para a qualidade de previsão de recompensas e de transições são as seguintes, respectivamente:

$$e_m^R = 1 - 2\ \ (Z_R(\Delta R_m)^2) \tag{3.6}$$

$$e_m^T = 1 - 2\ \ (Z_T \sum_{\kappa \in \mathcal{S}} \Delta T_m(\kappa)^2) \tag{3.7}$$

Acima, Z_R e Z_T são fatores de normalização dados por:

$$Z_R = (\frac{N(s,a)+1}{R_{max} - R_{min}})^2$$

e

$$Z_T = \frac{1}{2}(N(s,a)+1)^2$$

Ademais, note que R_{max} e R_{min} representam os valores máximos e mínimos para as recompensas. Note também que as equações (3.7) e (3.6) reescalam valores nas faixas $[0, 2]$ e $[0, 1]$, respectivamente, para o intervalo $[+1, -1]$. Neste intervalo, interpretamos $+1$ como a melhor qualidade de previsão possível, e -1 como a pior.

Note que chamamos e_m de *qualidade instantânea* devido ao fato de que a equação (3.5) leva em consideração apenas a *qualidade de previsão da última experiência do agente*. O valor da qualidade é modulado pela confiança c_m, a qual, conforme a equação (3.4), representa a razão entre o número de vezes que determinado par estado-ação foi visitado e a memória M. Em outras palavras, se o agente nunca houver visitado determinada situação (par estado-ação), não poderá confiar na qualidade instantânea medida logo após tal visita. Por outro lado, se o agente já houver vivenciado o referido par estado-ação M vezes, então sua confiança valerá 1. Esse valor é interpretado pelo agente RL-CD como um indicativo de que suas estimativas de $T(s, a, \cdot)$ e $R(s, a)$ são confiáveis, e que, portanto, sua última medida de qualidade instantânea também o é.

É interessante justificar a necessidade de utilizarmos tal valor de confiança no RL-CD. Informalmente, podemos esperar que, no início das experiências de um agente, suas previsões não sejam confiáveis, uma vez que tal agente ainda não experimentou o mundo e portanto provavelmente ainda não obteve bons modelos. Seguindo esta linha de raciocínio, quaisquer medidas de qualidade que o agente utilize, e que envolvam o poder de previsão de seus modelos, obrigatoriamente deverão levar em conta seu grau de incerteza. Em outras palavras, agentes inexperientes não poderão afirmar categoricamente se o ambiente está ou não se comportando da "maneira usual". Por outro lado, agentes que forem surpreendidos (i.e, que medirem valores baixos de qualidade) após visitarem pares estado-ação já bastante conhecidos, poderão afirmar com razoável certeza que tal quebra de expectativa reflete reais mudanças na dinâmica do ambiente.

Tendo em mente os fatos acima citados, podemos argumentar que valores baixos de M devem ser usados em ambientes pouco estocásticos, isto é, em ambientes nos quais o número de possíveis próximos estados é pequeno[4]. A explicação para isso é que quanto mais próximo o ambiente estiver do determinismo, menos experiências o agente precisará para ganhar confiança completa em suas estimativas de qualidade.

Note que as medidas fornecidas pelas equações (3.6) e (3.7) referem-se basicamente a uma totalização de todo ajuste necessário para que o respectivo modelo se torne coerente com a última observação do agente. Em outras palavras, caso o modelo precise ser muito ajustado, podemos inferir que tal modelo está bastante incorreto. Por outro lado, quanto menor for a distância do modelo para a realidade, menor a quantidade de ajustes necessários e conseqüentemente maior será sua qualidade.

É importante ressaltar que até agora estivemos ocupados apenas em formular uma medida de qualidade *instantânea*. Entretanto, visto que o sistema é estocástico e inicialmente desconhecido, é mais seguro calcular sua qualidade não apenas a partir de uma única experiência, mas sim a partir de um traço temporal das qualidades observadas nas últimas interações. Outro motivo para o uso de um traço temporal é que, dependendo dos vetores de probabilidades de transição específicos sendo testados, valores relativamente baixos de e_m podem ser calculados devido às características da distribuição de probabilidades da equação (3.7), mesmo que o modelo esteja correto. Uma discussão mais detalhada acerca da distribuição de (3.7) pode ser encontrada no próximo capítulo.

À medida de traço temporal da qualidade, chamada de E_m, reflete uma média ponderada dos últimos valores de qualidade instantânea. Ela é calculada para todos os modelos através da seguinte equação:

$$E_m = E_m + \rho\Big(e_m - E_m\Big) \tag{3.8}$$

[4]Isto é, se para (s, a) atual, $|s_{prox}| \ll |S|$, onde $s_{prox} = \{\kappa \text{ se } T(s, a, \kappa) > 0,\ \forall \kappa \in S\}$.

onde ρ é o coeficiente de ajuste responsável por especificar o quão importantes são as qualidades anteriores à da última experiência. Em geral, em ambientes muito ruidosos ρ deve ser ajustado para um valor baixo. Dessa forma, o traço de qualidade precisará ser consistentemente ajustado para níveis ruins durante várias experiências antes que os erros de previsão sejam considerados relevantes. Em outras palavras, valores baixos de ρ tendem a suavizar o ajuste de E_m.

A qualidade E_m é atualizada após cada experiência do agente para todos modelos, mas apenas o modelo ativo será corrigido de acordo com as equações (3.1) e (3.2). Toda vez que a qualidade de um modelo m se tornar melhor que a qualidade do atual, m_{cur}, o sistema detecta uma troca de contexto e ativa m. O limiar de qualidade mínima aceitável E_{min} é usado para controlar a criação de novos modelos. Sempre que o melhor modelo disponível for ainda pior que E_{min}, um novo modelo será criado. O RL-CD inicia com apenas um modelo parcial e incrementalmente constrói novos conforme necessário[5]. Note que a escolha do valor E_{min} não é simples, e depende do problema específico sendo tratado. Valores altos de qualidade mínima fazem com que o RL-CD detecte trocas de contexto mesmo para erros de previsão pequenos. Isso pode ser desejável em se tratando de cenários determinísticos, por exemplo, nos quais podemos esperar que RL-CD seja bastante exigente em termos de qualidade mínimo aceitável.

Cenários estocásticos são mais complexos de serem tratados. Por um lado, o uso valores altos de qualidade mínima pode fazer com que o RL-CD detecte trocas de contexto quando na verdade elas não ocorreram (falsos positivos). Valores muito baixos podem fazer com que o sistema não seja exigente o suficiente e releve erros de previsão importantes, deixando portanto de detectar trocas legítimas (falsos negativos). No capítulo seguinte iremos analisar formalmente o comportamento da distribuição da medida de qualidade dos modelos, e alguns dos impasses citados acima poderão ser estudados sob a perspectiva estatística de teste de hipóteses.

Note que o RL-CD opera como uma "casca" em torno dos algoritmos de aprendizado da política (ex: Prioritized Sweeping). Após cada atuação do agente no ambiente, os valores das equações (3.5) e (3.8) são calculados para cada modelo. Em outras palavras, o RL-CD estima a qualidade de previsão de cada modelo, ou, equivalentemente, o quão bem as previsões do respectivo modelo são coerentes com o que o agente realmente observou. Quanto mais coerentes forem as previsões, maior será a qualidade do modelo. Após este cálculo, o RL-CD faz com que o agente considere como modelo *atual* aquele com maior valor de qualidade E_m. Caso o melhor modelo disponível ainda possua qualidade pior que a mínima (E_{min}), então um novo modelo será criado. Após escolhido o melhor modelo disponível, o agente atualiza as estimativas daquele modelo para a função de transições e de recompensas de acordo com as equações (3.1) e (3.2), respectivamente. A política calculada pelo modelo ativo é então utilizada para determinar a ação específica que o agente irá tomar naquele instante. O método RL-CD completo é formalizado no algoritmo 12.

[5]Técnicas de descarte de modelos pouco utilizados, ou seja, que repetidamente apresentam qualidades de previsão ruins, não serão tratadas neste trabalho, mas certamente ajudariam a tornar o método mais eficiente em termos de consumo de memória.

Algoritmo 12 *Reinforcement Learning with Context Detection*

Seja $novo\_modelo()$ a rotina que cria e inicializa um novo modelo parcial.
Seja m_{cur} o modelo parcial atualmente ativo.
Seja $\mathcal{M}$ o conjunto de todos modelos disponíveis.

$m_{cur} \leftarrow novo\_modelo()$
$\mathcal{M} \leftarrow \{m_{cur}\}$
$s \leftarrow s_0$, tal que s_0 é algum estado inicial
Repita
    Seja a a ação indicada por $\pi_{m_{cur}}(s)$
    Observe o próximo estado s' e a recompensa r
    para todo $m \in \mathcal{M}$ **faça**
        Atualize E_m de acordo com (3.8)
    fim para
    $m_{cur} \leftarrow \arg\max_m (E_m)$
    se $E_{m_{cur}} < E_{min}$ **então**
        $m_{cur} \leftarrow newmodel()$
        $\mathcal{M} \leftarrow \mathcal{M} \cup \{m_{cur}\}$
    fim se
    Atualize $T_{m_{cur}}$ de acordo com (3.1)
    Atualize $R_{m_{cur}}$ de acordo com (3.2)
    $N_m(s, a) \leftarrow \min(N_m(s, a) + 1, M)$
    Atualize a política ou valores de estados, conforme o algoritmo específico de RL sendo utilizado.
    $s \leftarrow s'$
fim repita

Note que a rotina $novo\_modelo()$, usada no algoritmo 12, é responsável por criar um novo modelo parcial. Essa rotina cria o modelo parcial m e atribui $E_m = 0$, $R_m(s, a) = 0$ e $N_m(s, a) = 0$ para todo $s \in S$ e $a \in A$, e também de inicializa o modelo de transições de forma uniforme:

$$T_m(s, a, \kappa) \leftarrow \frac{1}{|\, S \,|} \quad \forall s \in S, \forall a \in A, \forall \kappa \in S$$

Note também que um novo modelo m recém criado possuirá valores de confiança c_m iguais a zero. Por essa razão, os erros de previsão gerados por m durante suas primeiras experiências, embora grandes devido à imprecisão dos modelos T_m e R_m, serão amenizados pelo fato de que a confiança em tais modelos é bastante baixa. Em outras palavras, é justamente a existência do fator de confiança c_m o que garante que modelos novos, recém criados, possuirão uma qualidade e_m maior do que a qualidade do modelo (ruim) atualmente disponível que causou a necessidade da criação de m.

3.3 Resultados empíricos

Nesta seção iremos apresentar três cenários de validação a fim de mensurar o desempenho do RL-CD em cenários não-estacionários. Nos experimentos que seguem, utilizamos o algoritmo Prioritized Sweeping (veja seção 2.8.2) para calcular a política de cada modelo parcial. Entretanto, qualquer outro método de RL baseado em modelo poderia ter sido usado, uma vez que o mecanismo do RL-CD apenas se encarrega de detectar as trocas de contexto e de decidir quando novos modelos são necessários. Caso se opte pelo uso de um algoritmo de RL sem modelo, como o Q-Learning, estimativas das funções de transição e recompensas instantâneas precisam ser computadas explicitamente.

3.3.1 Perseguição ao alvo

O primeiro cenário de validação consiste em um ambiente não-estacionário definido da seguinte forma:

- O ambiente é um grid discreto toroidal de 15x15 células;
- O agente é um gato cujo objetivo é perseguir e alcançar uma bola em movimento;
- A cada novo episódio de perseguição, a bola é colocada em uma posição aleatória do grid;
- A bola pode assumir um dentre quatro comportamentos, cada um dos quais correspondendo à movimentação numa das quatro direções cardinais;
- Quando o agente alcança a bola, recebe uma recompensa de $+10$. Em todos outros casos, ou seja, enquanto não alcançar a bola, recebe punição de -1.

O fator não-estacionário desta simulação diz respeito ao fato de que as regras de transição da bola mudam com o tempo. Note que neste cenário a não-estacionariedade afeta apenas a função de transicões, uma vez que os momentos em que o gato recebe recompensa permanecem os mesmos (i.e, apenas quando o gato alcança a bola, mas independentemente da forma exata como a bola está se movendo). Lembrando que a equação (3.5) utiliza um parâmetro Ω para especificar a importância relativa das qualidades de previsão de transições e de recompensas, fica claro que, para o presente cenário, podemos ignorar

a qualidade de previsão de recompensas, fazendo com que $\Omega = 0$. Os experimentos descritos nesta subseção fazem uso dos parâmetros apresentados na tabela 3.2, onde quer que eles se apliquem (ex: taxa de desconto é utilizada tanto no PS-M quanto no Q-Learning).

Tabela 3.2: Parâmetros utilizados nos experimentos da seção 3.3.1

Parâmetro	Valor
γ (taxa de desconto)	0.9
α (taxa de aprendizado)	0.1
Número de atualizações extras feitas pelo PS-M	5
M (memória)	5
ρ (taxa de ajuste de E_m)	0.75
E_{min} (qualidade mínima aceitável)	-0.1
Ω (importância relativa das qualidades e_m^T e e_m^R)	0
ϵ (valor de teste de convergência)	10^{-6}

O primeiro experimento foi organizado de forma a testar a eficiência relativa das abordagens RL sem modelo e com modelo, em comparação ao RL-CD, no ambiente não-estacionário definido. Especificamente, comparamos o RL-CD com o Q-Learning (QL) e com o Prioritized Sweeping com Memória Finita (PS-M)[6]. O agente é treinado para cada um dos contextos (tipo de "fuga" da bola) até que o algoritmo convirja para a política ótima (assumimos convergência quando nenhum valor de estado muda mais que $\epsilon = 10^{-6}$ entre iterações sucessivas). Após a convergência, alteramos a dinâmica do ambiente e medimos o quão rápido cada algoritmo consegue se readaptar e convergir para a nova política ótima.

Os resultados para este experimento são apresentados na figura 3.1. Conforme os contextos são alterados, o tempo para recalcular as políticas ótimas em ambas as abordagens clássicas é bastante superior ao tempo de convergência do RL-CD. Embora o PS-M obtenha um desempenho superior ao do Q-Learning, ainda precisa reestimar T a cada troca de contexto. O RL-CD, por outro lado, demora apenas poucas iterações até que a qualidade do modelo atual caia abaixo do limiar e a respectiva mudança de contexto seja detectada. Depois disso, o RL-CD automaticamente escolhe o melhor modelo parcial disponível e o utiliza, ou então cria um novo modelo quando nas primeiras visitações a um contexto.

Este experimento, entretanto, não é completo. Também é importante que mensuremos o tempo médio que o gato demora para capturar a bola *enquanto* a política está sendo aprendida, isto é, antes que a convergência completa em um contexto seja atingida. Para

[6]Conforme mencionado na seção 2.9, o ajuste de modelos únicos em ambientes não-estacionários é difícil quando os parâmetros de tal modelo são estimados na forma de médias móveis, como normalmente acontece nas implementações do PS. Por essa razão, implementamos um truncamento nas contagens de freqüências de visitação de pares estado-ação, a fim de evitar com que o coeficiente de ajuste dos parâmetros caísse para zero conforme o número de experiências crescesse. De certa forma, essa é a mesma abordagem utilizada na equação (3.3). Caso não utilizássemos um limitador para as contagens de visitação, o tempo de convergência do PS em ambientes não-estacionários se tornava exponencial. Chamamos o PS com memória truncada de Prioritized Sweeping com Memória Finita (PS-M).

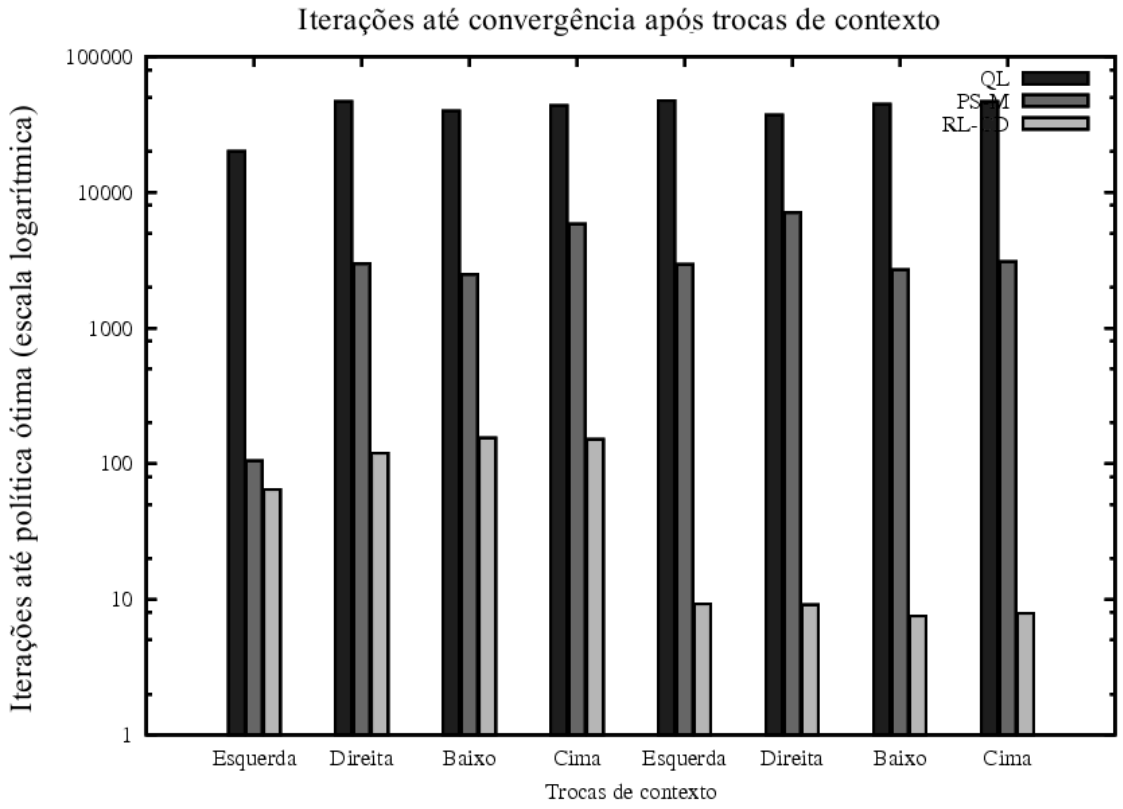

Figura 3.1: Comparação dos tempos de convergência para o Q-Learning, PS-M e RL-CD (Perseguição ao Alvo).

isso, medimos o desempenho de cada um dos algoritmos durante oito trocas de contexto. Para cada contexto, executamos 5 conjuntos de 100 episódios, sendo cada episódio definido como o tempo desde que o gato é colocado em um local aleatório do grid até a bola ser capturada. Os resultados são apresentados na figura 3.2. Note que a colocação de uma linha vertical tracejada, a cada 5 conjuntos de episódios, demarca as trocas de contexto.

Na figura 3.2, listras verticais são usadas para indicar diferentes contextos. Cada vez que o comportamento da bola é alterado, o número de iterações por episódio cresce (picos no gráfico), indicando que os algoritmos demoram certo tempo até reaprenderem uma política razoável. Durante os quatro primeiros contextos, o desempenho do RL-CD é bastante próximo ao do PS-M. Entretanto, assim que os contextos começam a repetir, o RL-CD se mostra capaz de voltar à atuação ótima muito rapidamente, enquanto que os outros algoritmos apresentam tempos de reaprendizado a atuação subótima sempre que há uma mudança de contexto.

Por fim, a figura 3.3 mostra a evolução temporal do traço de qualidade E_m para cada modelo m, neste experimento. Note como o RL-CD cria quatro modelos, cada um correspondendo a uma das dinâmicas do ambiente. Perceba também que durante as primeiras visitações a cada contexto, nem todos os modelos estão disponíveis, o que indica que eles foram criados sob demanda. A linha horizontal em -0.1 indica o limiar de qualidade mínima E_{min}. A cada momento, o sistema seleciona o modelo com a mais alta qualidade ou cria um novo sempre que o melhor estiver abaixo de E_{min} (para uma discussão mais detalhada sobre como determinar um bom valor para E_{min}, vide seção 4.4). Como no experimento anterior, aqui também usamos listras verticais para indicar o contexto atual

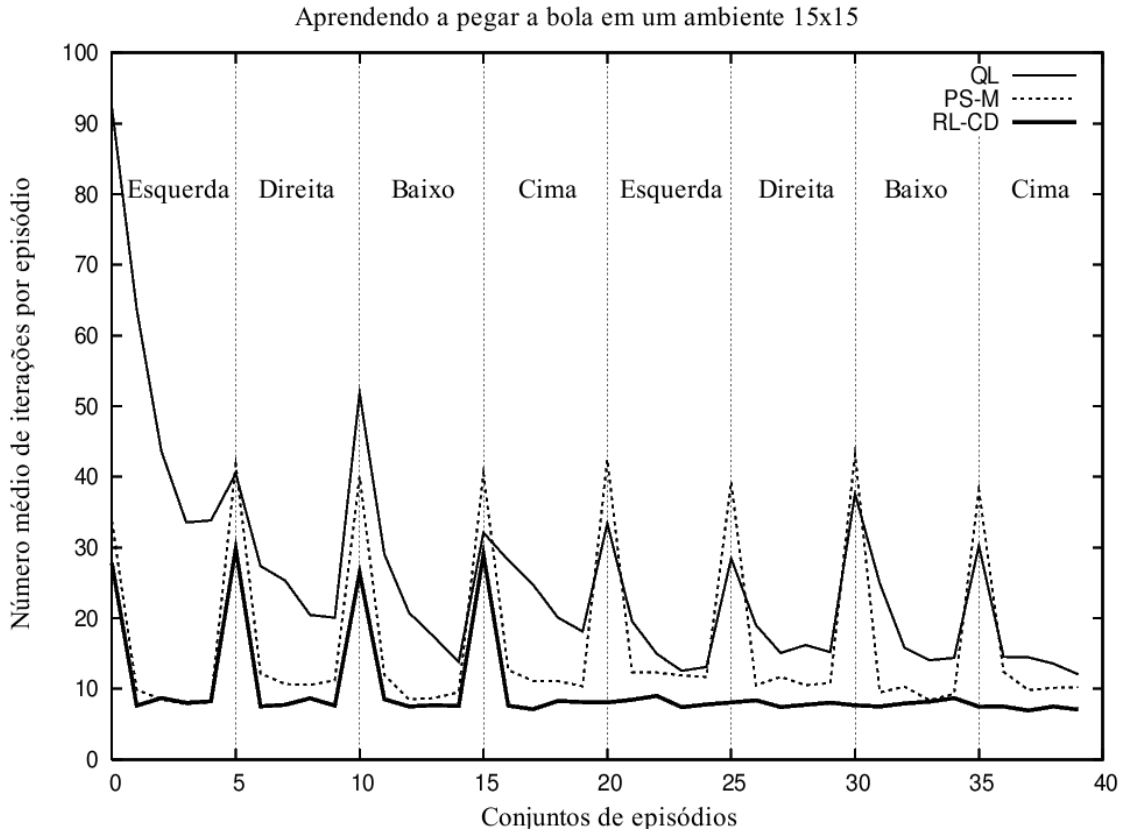

Figura 3.2: Comparação de desempenho entre Q-Learning, PS-M e RL-CD (Perseguição ao Alvo).

do ambiente.

A fim de gerar o gráfico da figura 3.3, alteramos explicitamos a dinâmica do ambiente cada vez que o gato capturava a bola pela vigésima quinta vez. O fato de os contextos durarem cada vez menos conforme o tempo passa indica que a política do gato está se tornando progressivamente melhor, ou seja, que o gato consegue efetuar as 25 capturas cada vez mais rapidamente. Isso permite com que o sistema consiga manter o desempenho em níveis próximos do ótimo, sempre detectando as trocas de contexto e rapidamente escolhendo o modelo mais adaptado para a respectiva situação.

3.3.2 Controle de semáforos

O segundo cenário de validação do RL-CD consiste em uma rede viária 3x3 em forma de grid, com um semáforo colocado em cada intersecção. A figura 3.4 apresenta graficamente a disposição da rede, contendo 9 nodos (cada nodo com um semáforo) e 24 ruas dirigidas conectando-os.

Cada rua dirigida possui capacidade para 50 veículos. Os veículos são inseridos por *injetores* e posteriormente removidos por *sumidouros*, representados na figura 3.4 por losangos e quadrados, respectivamente. O número exato de veículos inseridos pelos injetores é dado por uma distribuição Gaussiana. Se um veículo está para ser inserido mas não há mais espaço disponível na respectiva rua, ele aguarda em uma fila externa até que a inserção seja possível. Os veículos não mudam de direção durante a simulação, e são removidos da rede ao alcançarem um sumidouro.

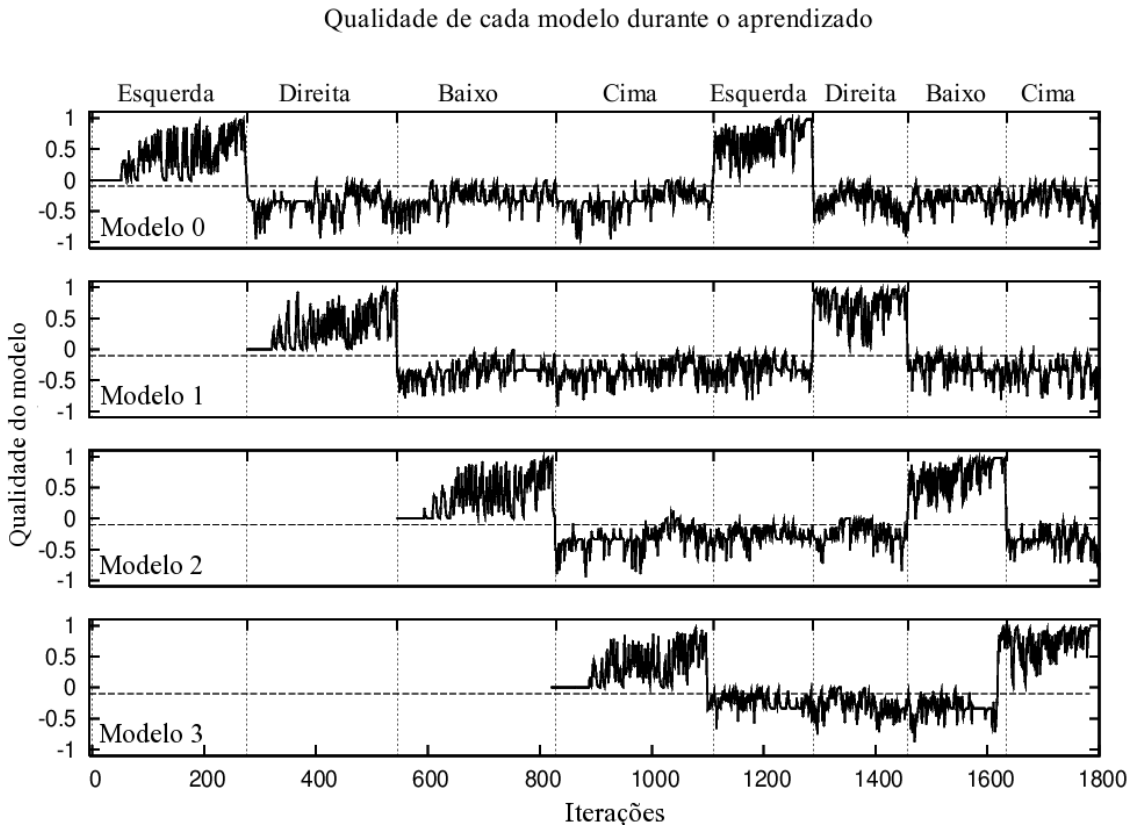

Figura 3.3: Traço temporal de qualidade E_m para todos modelos (Perseguição ao Alvo).

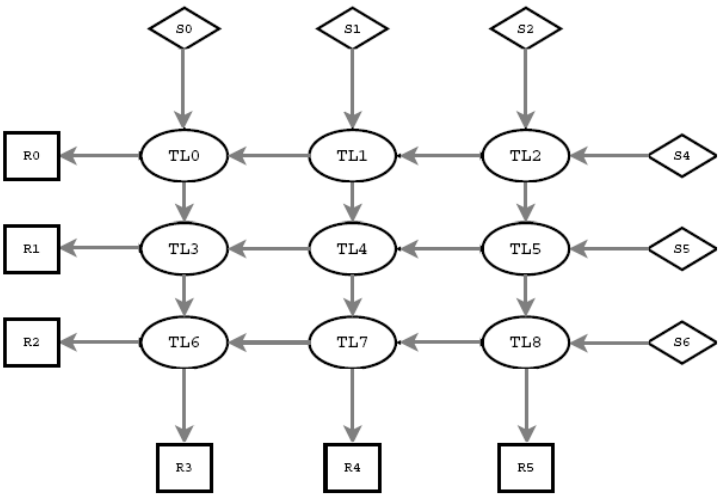

Figura 3.4: Uma rede viária com nove intersecções.

O problema de aprendizado aqui descrito foi modelado de forma que cada semáforo é controlado por um agente, e cada agente toma apenas decisões locais[7]. Tais decisões de

[7]Esse é um bom exemplo em que a aplicação de aprendizado com múltiplos agentes é interessante/necessária, dada a explosão do espaço de estados que ocorreria caso tentássemos resolver este problema de controle de forma centralizada.

controle, como será explicado mais adiante, consistem na seleção de um plano semafórico de forma a priorizar uma direção de trânsito específica (ex: Norte–Sul, Leste–Oeste, etc). O fato das decisões serem *locais* significa que o processo de escolha entre planos leva em consideração apenas a percepção local que o agente tem do estado das ruas que o cercam. Note também que embora tais decisões sejam locais, nós medimos o desempenho do mecanismo de controle sobre a rede como um todo. A métrica exata que é utilizada para desempenho é o número total de veículos parados na rede. Quanto mais veículos parados, maior o engarrafamento, o que indica que os agentes não estão priorizando as direções de tráfego corretas a cada momento.

A fim de definir o conjunto de estados S deste problema, executamos uma discretização nos possíveis tamanhos de filas de veículos. Assim, dizemos que a ocupação de uma rua dirigida pode ser considerada *vazia*, *normal* ou *cheia*, dependendo do número de veículos que contém. O estado do agente será dado pelo conhecimento combinado das ocupações de cada uma das ruas que chegam no semáforo por ele controlado. A recompensa será dada pelo inverso (em termos de sinal) da soma quadrática do tamanho das filas que chegam no respectivo semáforo sob controle.

A fim de definir as possíveis ações de cada agente, nos baseamos no modo de funcionamento de semáforos reais, os quais utilizam um conjunto de *planos semafóricos* a fim de lidar com as diferentes condições de tráfego. Cada plano semafórico especifica um conjunto ordenado de direções de tráfego nas quais o movimento de veículos será permitido (*fases*), ou seja, para as quais o semáforo dará direito de passagem. O tempo total de duração de todas as fases de um plano semafórico equivale ao seu *tempo de ciclo*. Neste problema, consideramos como possíveis ações apenas 3 tipos de plano semafórico. A ação do agente corresponde justamente em selecionar um destes planos a cada passo de simulação[8]. Os planos semafóricos disponíveis ao agente são:

- plano **1**, que dá direito de passagem em proporções iguais para ambas as direções da rede (Norte–Sul e Leste–Oeste);
- plano **2**, que prioriza o trânsito na direção vertical (Norte–Sul);
- e plano **3**, que prioriza o trânsito na direção horizontal (Leste–Oeste).

Todos planos semafóricos possuem tempo de ciclo de 60 segundos, e cada fase dura 42, 30 ou 18 segundos (respectivamente, 70%, 50% e 25% do tempo de ciclo para a direção preferencial do plano). Portanto, o plano semafórico 1 dá direito de passagem por 30 segundos para cada uma das direções; o plano 2 prioriza a direção vertical por 42 segundos e logo após a horizontal, por 18 segundos; por fim, o plano 3 prioriza a direção horizontal por 42 segundos, e a seguir a vertical por 18 segundos.

A fim de modelar a não-estacionariedade do cenário, definimos três padrões de trânsito com diferentes distribuições de inserção de veículos nos nodos injetores. Os padrões (contextos) são:

- *Baixo*: taxa de inserção baixa em todos nodos injetores de veículos, de forma que a rede permanece não-saturada mesmo quando as políticas sendo seguidas não forem ótimas;

[8] Uma vez que a ação do agente corresponde a escolha de um plano, cada passo de simulação equivale à duração do tempo de ciclo do plano.

- *Vertical*: alta taxa de inserção através dos injetores Norte (S0, S1 e S2), mas inserção mediana pelos injetores Leste (S4, S5 e S6);
- *Horizontal*: alta taxa de inserção através dos injetores Leste (S4, S5 e S6), mas inserção mediana pelos injetores Norte (S0, S1 e S2).

As distribuições Gaussianas de inserção de veículos usadas nos contextos *Vertical* e *Horizontal* são tais que a rede satura sempre que as políticas seguidas não forem ótimas[9]. Taxas de inserção altas a partir de ambas as direções não são simuladas pois nenhuma ação ótima seria possível, uma vez que a rede inevitavelmente ficaria saturada em poucos passos, o que implicaria um ambiente estacionário com todas ruas em ocupação máxima.

Perceba que este cenário é bem mais difícil que o de Perseguição ao Alvo, já que ele é probabilístico e possui três diferentes causas para a não-estacionariedade: **1)** mudanças explícitas nas taxas de inserção de veículos; **2)** discretização grosseira do estado; e **3)** ações dos semáforos vizinhos, as quais são difíceis (senão impossíveis) de prever.

Os experimentos descritos nesta subseção fazem uso dos parâmetros apresentados na tabela 3.3, onde quer que eles se apliquem (ex: taxa de desconto é utilizada tanto no PS-M quanto no Q-Learning).

Tabela 3.3: Parâmetros utilizados nos experimentos da seção 3.3.2

Parâmetro	Valor
γ (taxa de desconto)	0.9
α (taxa de aprendizado)	0.1
Número de atualizações extras feitas pelo PS-M	5
M (memória)	20
ρ (taxa de ajuste de E_m)	0.2
E_{min} (qualidade mínima aceitável)	-0.1
Ω (importância relativa das qualidades e_m^T e e_m^R)	0

Note que neste cenário, ao contrário do caso de perseguição ao alvo apresentado na seção 3.3.1, utilizamos um valor mais baixo para a taxa de ajuste ρ e um valor mais alto para a memória. Isso foi feito devido ao fato do ambiente ser bem mais estocástico do que o apresentado em 3.3.1. Um valor de memória mais elevado faz com que a confiança nos sinais de qualidade instantânea somente seja alta quando o agente já tiver experimentado várias vezes um determinado par estado-ação. De forma semelhante, um valor baixo de ρ faz com que o ajuste do traço de qualidade seja mais lento, evitando assim que erros de previsão ocorridos simplesmente pelo fato do ambiente ser probabilístico afetem E_m de maneira severa. Note também que utilizamos $\Omega = 0$, ou seja, que estamos ignorando a qualidade de previsão de recompensas e levando em conta apenas a qualidade de previsão

[9]Em todos os padrões de inserção, a distribuição Gaussiana dos injetores possui desvio 2. A média depende do contexto: para o padrão de inserção *Baixo*, a média de inserção de veículos a partir de todos os injetores é 10; para o padrão de inserção *Vertical*, os injetores S0, S1 e S2 atuam com média 20, enquanto que os injetores S4, S5 e S6 atuam com média 10; por fim, durante o padrão de inserção *Horizontal* os injetores S4, S5 e S6 atuam com média 20, e os injetores S0, S1 e S2 com média 10.

de transições. Isso se deve ao fato de que tanto as recompensas quanto o estado atual, neste cenário, são calculados com base na mesma quantidade numérica (número de veículos em cada rua que chega no semáforo). Dito de outra forma, no ambiente descrito nesta seção a qualidade instantânea e_m^T já contém toda a informação relevante para a detecção de troca de contexto. De fato, em nossos experimentos foi possível constatar que o uso de outros valores para Ω, tais como $\Omega = 0.5$, não afetaram o desempenho do RL-CD.

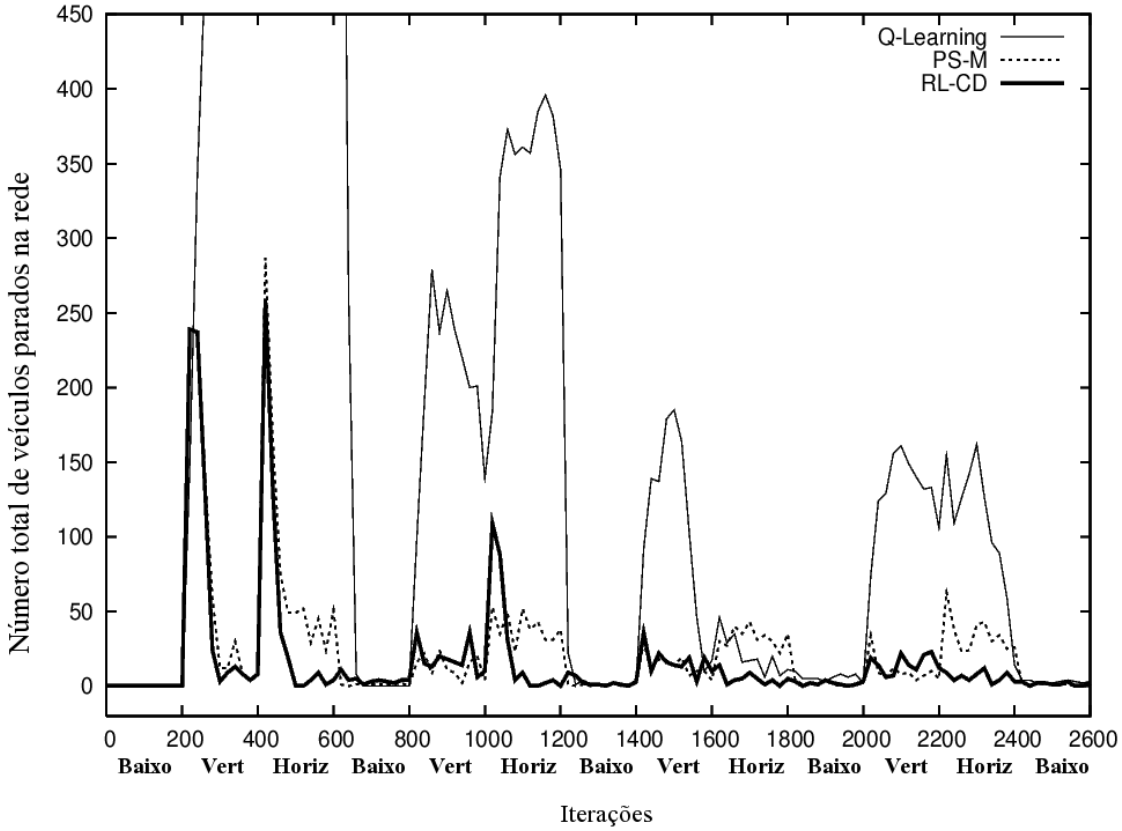

Figura 3.5: Comparação de desempenho entre Q-Learning, PS-M e RL-CD (Contole de Semáforos).

Na figura 3.5, comparamos o desempenho do RL-CD com o de dois métodos clássicos de aprendizado por reforço (Q-Learning e Prioritized Sweeping). A simulação de tráfego para este cenário é tal que os padrões de trânsito (i.e, as taxas de inserção) são modificados a cada 200 iterações, que correspondem à simulação de aproximadamente 3 horas de trânsito real. Nossa medida de desempenho, conforme mencionado anteriormente, consiste no número total de veículos parados em todas as ruas. Essa medida inclui os veículos esperando nas filas externas, ou seja, nas filas que armazenam veículos que não puderem ser inseridos pelos injetores devido à saturação de uma rua. Isso significa que, quanto mais baixo o valor na figura 3.5, melhor a performance, pois menor é o número de veículos parados em engarrafamentos.

A comparação dos métodos é feita na figura 3.5. Em primeiro lugar devemos notar que, embora o Q-Learning não sofra com o problema causado pelo uso de modelos errôneos durante o cálculo da política, ainda assim precisa constantemente reestimar seus valores de estado-ação. O PS-M também precisa reestimar seus valores de estado, além,

é claro, do seu modelo do ambiente. Entretanto, ao contrário do Q-Learning, uma vez ajustado o modelo, as correções de valores V se propagam muito rapidamente, devido ao próprio mecanismo de priorização do PS-M. Portanto, embora o Q-Learning não precise reajustar modelos de dinâmica, tambem não é capaz de contruir representações de atributos capazes de acelerar o cálculo da política. O algoritmo Prioritized Sweeping, por outro lado, tenta construir um *único* modelo do ambiente, e acaba representando um modelo que mistura propriedades de diversos padrões de tráfego. Por essa razão, o melhor que consegue obter é uma política de compromisso entre diversos (e às vezes opostos) padrões de inserção de veículos.

Por fim, o RL-CD possui desempenho superior tanto ao Q-Learning quanto ao PS-M. Em comparação com o Prioritized Sweeping, especificamente, percebemos que o RL-CD mostra vantagens especialmente quando a transição de contexto ocorre entre padrões de tráfego completamente opostos, isto é, entre o contexto *Vertical* e o *Horizontal*. Trocas de contexto envolvendo a dinâmica de *fluxo baixo* em geral não causam mudanças bruscas na matriz de transições, pois vários dos vetores de probabilidades se repetem entre os contextos. Portanto, embora o RL-CD não tenha problemas em detectar as trocas nestes casos, a respectiva variação da dinâmica não é especialmente significativa a ponto de fazer com que o PS-M não se recupere rapidamente. Sempre que a dinâmica muda para o contexto *Horizontal*, entretanto, podemos perceber claramente o ganho de desempenho promovido pelo RL-CD. Por fim, a figura 3.5 também permite concluir que o RL-CD apresenta desempenho superior ao do Q-Learning, não só por utilizar modelos mas também por utilizar os modelos corretos nos momentos apropriados, e com isso evitar a necessidade de contínua readaptação de suas estimativas de valor de estado.

A aplicação do RL-CD em um cenário de controle de tráfego mais realístico, obtido através do auxílio de um simulador microscópico baseado em Autômatos Celulares (SILVA et al., 2006), é descrito em (OLIVEIRA et al., 2006).

3.3.3 Perseguição ao Alvo, Revisitado

Conforme mencionado nas subseções 3.3.1 e 3.3.2, abordagens que lidam com a não-estacionariedade através do reaprendizado contínuo, como o PS-M, requerem tempos de convergência altos devido à necessidade de redescoberta de políticas.

Métodos alternativos baseados em múltiplos modelos foram propostos a fim de amenizar este problema. Especificamente, e devido às similaridades com o RL-CD, ressaltamos novamente o trabalho de Doya *et al* (DOYA et al., 2002) e também de Choi *et al* (CHOI; YEUNG; ZHANG, 2001), os quais se chamam, respectivamente, MMRL e HM-MDP. Os HM-MDPs são usados para modelar ambientes não-estacionários nos quais a não-estacionariedade é causada por um atributo de estado oculto denominado *modo*. O objetivo do método proposto em (CHOI; YEUNG; ZHANG, 2001) é aprender um modelo de transições *entre* os modos, ou seja, entre os MDPs usados para descrever cada regime de funcionamento do ambiente. Entretanto, o trabalho de Choi permite apenas aprendizado *offline* do ambiente não-estacionário, dado um conjunto suficientemente grande de amostras aleatórias de transições e experimentações. Por esta razão, é muito difícil obter uma comparação justa de desempenho entre métodos adaptativos, como o PS-M e o RL-CD, e o método de estimativa de HM-MDPs. Nesta seção, iremos enfatizar testes de desempenho contra o MMRL e contra o Compositional Q-Learning, o qual será brevemente discutido adiante.

O algoritmo MMRL, descrito em 2.9.1, lida com a não-estacionariedade através da decomposição de uma tarefa em múltiplos domínios nos quais o ambiente pode ser previsto

com sucesso. A idéia geral é computar um sinal de responsabilidade para cada modelo parcial de forma que os modelos com predições mais eficazes possuam maior grau de responsabilidade. As principais diferenças do MMRL para o RL-CD são enumeradas abaixo:

1. O MMRL utiliza medidas específicas de continuidade espacial e temporal, enquanto que o RL-CD apenas computa um traço de qualidade mas não se preocupa com medidas de centralidade dos modelos no espaço de estados;

2. O MMRL não decide quais ações tomar baseado apenas no melhor modelo, e sim através da ponderação de ações propostas por cada modelo;

3. Ao contrário do RL-CD, o MMRL assume que o número de modelos, e portanto de regimes de funcionamento do ambiente, é conhecido *a priori*;

4. O MMRL exige menos parâmetros que o RL-CD, principalmente por já partir do conhecimento explícito do número (fixo) de modelos necessários para descrever o ambiente;

5. Ao contrário do RL-CD, o MMRL apenas lida com a não-estacionariedade através de erros de previsão de transições, mas não de recompensas. Portanto, ambientes não-estacionários quanto à tarefa a ser resolvida (i.e, nos quais há alterações na função de recompensa) não são bem tratados pelo MMRL.

Em (DOYA et al., 2002), o MMRL é comparado com uma versão modificada do algoritmo Compositional Q-Learning, ou CQ-L (SINGH, 1992). O cenário proposto é uma tarefa de perseguição similar à apresentada na subseção 3.3.1, exceto pelo fato de que o grid tem dimensões 7x7, e não 15x15. O algoritmo CQ-L consiste basicamente em um Q-Learning composto por múltiplos módulos de aprendizado, tal que a probabilidade de seleção de um módulo específico é dada pela sua capacidade de prever a função de valor de estado. Em outras palavras, o CQ-L mensura *erros de valor de estados*, enquanto que o MMRL e o RL-CD medem erros (ou, equivalentemente, qualidades) através da comparação com *transições e recompensas esperadas*. Em (DOYA et al., 2002), Doya e colegas modificaram o Compositional Q-Learning de forma que ele pudesse funcionar sem que as mudanças no ambiente precisassem ser sinalizadas de forma explícita ao algoritmo, como era exigido pelo método original. Na comparação que efetuaremos, também utilizamos tal versão modificada do CQ-L. Os parâmetros utilizados para o RL-CD são os mesmos apresentados na tabela 3.2. Os parâmetros e detalhes exatos da implementação do MMRL e do CQ-L podem ser encontrados no artigo original de Doya e colegas (DOYA et al., 2002).

Na figura 3.6 é apresentada uma comparação de desempenho entre o RL-CD, MMRL, Q-Learning e CQ-L no mesmo experimento proposto por Doya, isto é, em um cenário 7x7 de perseguição ao alvo. Note que este cenário é ligeiramente diferente do utilizado na subseção 3.3.1. Isto ocorre pois nós agora comparamos o RL-CD sob as mesmas condições e parâmetros propostos por Doya, uma vez que essas são as condições e configurações que geram os melhores resultados para o MMRL. No experimento de Doya, a direção da bola muda após cada episódio, e um episódio dura até que o gato pegue a bola ou que tenham se passado 100 iterações. Entretanto, uma vez que o RL-CD não é capaz de detectar trocas de contexto quando estes são muito breves (ex: duram apenas uma iteração), nós

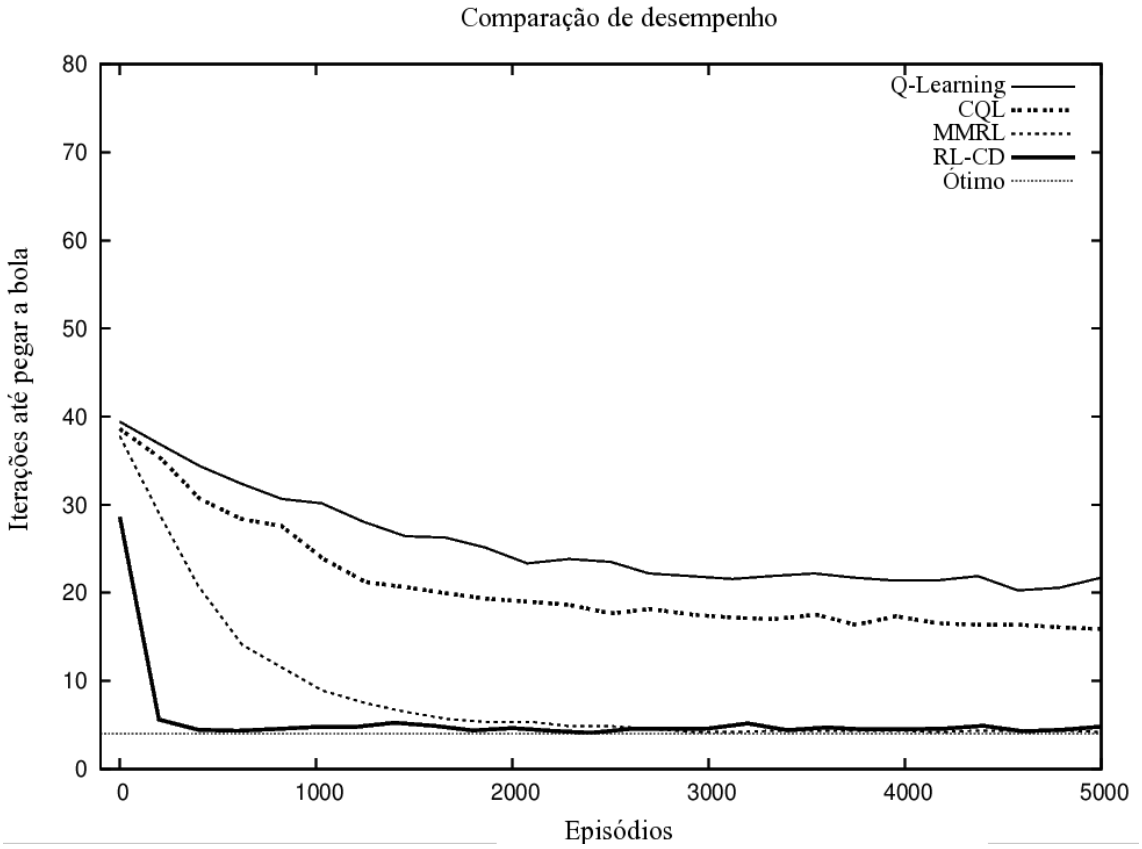

Figura 3.6: Comparação de desempenho entre Q-Learning, Compositional Q-L, MMRL e RL-CD (Perseguição ao Alvo, Revisitado).

medimos os desempenhos e efetuamos as comparações em um cenário onde os contextos são alterados a cada 5 episódios.

Os resultados discutidos a seguir são relativos a figura 3.6. Durante o período inicial de aprendizado, pudemos verificar que todos os modelos do MMRL eram igualmente responsáveis por todo o domínio. Conforme o agente MMRL aprendia, seus modelos se tornavam cada vez mais especializados em domínios restritos dentre todas as possíveis dinâmicas do ambiente. Entretanto, uma vez que o RL-CD utiliza apenas um modelo por vez, sua adaptação ocorre de maneira muito mais rápida. Por outro lado, pelo fato do MMRL não usar uma abordagem "vencedor-leva-tudo", pode ser menos suscetível a ruído. Esse experimento também confirma que a alternância muito rápida entre contextos é problemática para o RL-CD, uma vez que modelos desnecessários são criados, gerando erros de falso positivo. Como conseqüência, não apenas a divisão de responsabilidades entre os modelos é prejudicada, como também o próprio mecanismo de seleção do modelo mais adequado.

3.4 Discussão

Os experimentos acima apresentados confirmam que o RL-CD funciona bastante bem *caso as suposições exigidas sejam obedecidas*. Entretanto, em ambientes nos quais pode ocorrer sobreposição de contextos (i.e, onde há contextos com um número significativo de vetores probabilidades de transição em comum), o RL-CD pode se tornar ineficiente. Isso

ocorre pois o sistema tem de replicar e reaprender modelos referentes a todo o espaço de transições, quando na verdade a não-estacionariedade apenas afeta uma parcela do espaço total.

Uma possível solução para lidar com contextos que não são completamente separáveis, ou seja, que possuem porções importantes de suas matrizes de transição e recompensa em comum, é considerar a *localidade espacial* no cálculo da qualidade. Em outras palavras, poderia ser feito um cálculo de qualidade referente a subregiões específicas do espaço de estados, permitindo com isso que os modelos se especializassem em porções mais delimitadas do ambiente. Entretanto, no caso geral de MDPs pode ser difícil (ou mesmo impossível) definir uma métrica consistente de distância entre estados, e por essa razão a idéia de localidade espacial em espaços de estado discretos é problemática.

Outro ponto importante de análise do RL-CD é a determinação do valor de memória M. Conforme mencionado, a memória serve tanto para truncar o ajuste do modelo de transições e recompensas, de forma a facilitar a adaptação intra-modelo frente à não-estacionariedade, quanto para configurar a forma através da qual a confiança irá variar com o número de experiências. Quanto maior for o valor de memória, menos significativas serão as repetições de uma experiência já conhecida; em outras palavras, quando a memória for grande o agente precisará experimentar várias vezes determinado par estado-ação até ter confiança em suas estimativas de qualidade. Esta característica é útil, por exemplo, em cenários estocásticos onde são necessárias várias amostras de experiência para que o agente possa ter alguma certeza de que os parâmetros do modelo foram estimados razoavelmente.

Por outro lado, em cenários determinísticos ou próximos disso, a memória pode ser pequena, pois bastam poucas experimentações (uma, no caso extremo) para que o agente tenha certeza absoluta da probabilidade de uma transição. O uso de valores altos de memória em cenários determinísticos faz com que a confiança seja mantida baixa durante todas as primeiras experências do agente, enquanto que, na verdade, a confiança nesses casos deveria ser total. Este problema se torna mais crítico ainda caso o cenário, além de determinístico, também apresente trocas de contexto freqüentes. Nessa situação, o agente terá poucas oportunidades para re-experimentar situações dentro de um mesmo contexto, e portanto sua confiança será sempre mantida baixa. Como conseqüência, o agente *nunca* conseguirá obter boas qualidades de previsão instantânea, mesmo que o ambiente seja completamente previsível.

A conclusão que podemos tirar é que o uso de valores de memória excessivamente altos pode tornar o agente rigoroso demais a ponto de não detectar trocas de contexto legítimas. A determinação de um bom valor de memória depende principalmente, portanto, de quão estocástico é o ambiente.

Assim como o valor de memória está ligado a quão estocástico é o ambiente, assim também ocorre com a taxa de ajuste do traço de qualidade ρ. Se o ambiente for completamente determinístico, a taxa de ajuste pode ser 1. Nesse caso, uma qualidade baixa de previsão automaticamente implica uma troca de contexto. Em cenários bastante estocásticos, ρ deve ser usado para suavizar a variação temporal de E_m. Note que a necessidade de se levar em conta o quão estocástico é um cenário se apresenta como um dos pontos fundamentais para as abordagens de múltiplos modelos. Isso não ocorre por acaso. De fato, é bastante difícil determinar, para o caso geral, se uma expectativa não-atendida se deve **1)** ao desconhecimento de uma situação; **2)** à característica altamente probabilística do sistema; ou **3)** a uma legítima troca de contexto. Os parâmetros ρ e M, em conjunto, tentam amenizar estes empecilhos dado os conhecimentos do projetista sobre o sistema

sendo tratado.

3.5 Possíveis melhorias

Inspirados pelo trabalho de Choi (CHOI; YEUNG; ZHANG, 2001), acreditamos que a modelagem de transições entre modelos parciais pode ser um próximo passo importante para o RL-CD. Ao obter este tipo de informação, poderíamos lidar com casos em que a troca de contextos não é independente das ações do agente. Além disso, a troca de contexto poderia ser estimada *previamente*, e não apenas detectada *a posteriori* com base nos erros de previsão.

Outra possível melhoria para o algoritmo RL-CD diz respeito à criação de mecanismos explícitos para lidar com casos em que apenas as intenções do agente são alteradas. Atualmente, caso apenas as funções de recompensa sejam alteradas, mas as transições se mantenham, o RL-CD é capaz de detectar a troca de contextos, mas logo em seguida tem de reestimar T, o que é claramente desnecessário[10]. A solução para este impasse seria criar modelos parciais de R e T separadamente. Caso apenas a tarefa fosse alternada, uma nova função de recompensas seria aprendida, mas a de transições continuaria funcionando corretamente. Isso também permitiria com que várias possíveis tarefas fossem aprendidas sob diferentes dinâmicas, bastando para isso que ocorresse a reestimação das recompensas instantâneas e o cálculo da nova política. Em resumo, essa melhoria faria com que o RL-CD não apenas fosse capaz de detectar trocas gerais na dinâmica do ambiente, como também de reconhecer rapidamente uma dentre diversas tarefas a serem resolvidas, escolhendo em seguida o método de controle mais correto para aquela situação.

Conforme mencionado anteriormente, a escolha do parâmetro E_{min} (qualidade mínima aceitável para os modelos) é complexa, pois valores muito baixos de E_{min} podem fazer com que o agente não detecte trocas de contexto legítimas, enquanto que valores muito altos podem fazer com que o agente crie uma série de modelos parciais desnecessários. No capítulo que segue, iremos apresentar uma análise teórica de parâmetro E_{min} de forma não apenas a estudar o RL-CD sob o prisma do teste estatístico de hipóteses, mas também de descobrir o valor ótimo para a qualidade mínima aceitável.

[10] Isso aconteria mesmo que $\Omega = 1$, pois o modelo parcial ainda seria composto monoliticamente por R e T, indissociáveis.

4 RL-CD BASEADO EM TESTE DE HIPÓTESES

A base central do algoritmo RL-CD são as equações para cálculo da qualidade de modelos. É a partir destes valores que o sistema toma decisões de quando criar novos modelos e de como escolher, a cada momento, o modelo mais apropriado. Uma pergunta importante diz respeito à forma com que os valores de e_m^T são distribuídos, em termos de probabilidades. Dada uma distribuição de probabilidades de próximos estados, $T(s, a, \cdot)$, e uma observação de transição, que tipo de distribuição de valores de qualidade devemos esperar? Se as freqüências de próximos estados observados seguir sempre as probabilidades esperadas, com que freqüência podemos esperar que o valor de qualidade seja positivo? Qual a chance de obtermos uma qualidade negativa mesmo em ambientes estacionários? A análise destas perguntas será o foco de estudo do presente capítulo.

Vamos supor que um jogador possua uma moeda que ele acredita ser justa (não-viciada); suponha, entretanto, que ele lance essa moeda 1000 para o alto e observe 999 caras. Nesse caso, poderá ele afirmar categoricamente que a moeda *não* era, como ele supunha inicialmente, justa? É razoável supor que sim, pois a diferença entre o valor esperado e o valor obtido é muito grande. Suponha, por outro lado, que o jogador tenha lançado 1000 moedas mas observado 450 caras. Poderia o jogador afirmar, com a mesma certeza, que a moeda não era justa? Provavelmente não, uma vez que a chance de uma moeda justa, quando lançada 1000 vezes, resultar em um número de caras próximo a 500, é alta. Em geral, estamos interessados em saber qual a probabilidade de que a variação entre os resultados observados e os esperados se deva tão somente ao acaso; ou, em outras palavras, qual a probabilidade de que a observação de resultados diferentes dos esperados se deva simplesmente ao próprio processo de amostragem. Essa é a questão central da área estatística que lida com *testes de hipóteses*. No restante deste capítulo, iremos estudar maneiras de aproximar os métodos de detecção de contexto das técnicas estatísticas utilizadas em testes de hipóteses.

4.1 Teste de hipóteses

Um dos papéis mais importantes da estatística é o de desenvolver evidências de que algo está ou não se comportando conforme o esperado. Tipicamente, dadas amostras obtidas a partir de uma distribuição com parâmetros desconhecidos, podemos querer testar se elas diferem significativamente do tipo de amostras que obteríamos caso estivéssemos coletando amostras advindas de uma distribuição de probabilidades conhecida. Um teste de hipóteses também pode servir, por exemplo para medirmos o grau de certeza com que podemos afirmar que certo parâmetro (média, variância, etc) se encontra dentro de determinado intervalo numérico. A partir desses testes, é possível "provar", com algum grau de confiança, se determinada hipótese é ou não válida.

Como exemplo inicial, suponha que em um rio existam 3 espécies de peixe: A, B e C. A princípio, esperamos que as espécies ocorram com chances iguais, ou seja, 33% cada. Suponha também que amostremos 100 peixes deste rio, e que as contagens sejam de 40 peixes do tipo A, 32 do tipo B e 28 do tipo C. Evidentemente, esperávamos obter 33 peixes de cada espécie. Qual a chance de que essa variação tenha ocorrido por acaso, e qual a chance de que ela, ao contrário, indique que nossa suposição de equiprobabilidade entre espécies é falha?

Uma maneira de iniciar a resolução deste problema é obter uma métrica de erro. Digamos que o pesquisador decida medir o erro da seguinte forma:

$$Erro = \sum_i \frac{(observado_i - esperado_i)^2}{esperado_i}$$

onde i refere-se a cada uma das categorias envolvidas no teste; no nosso caso atual, a cada espécie de peixe.

A equação apresentada acima é uma forma de erro relativo entre o valor observado e o esperado, exceto que elevamos a diferença ao quadrado para evitar cancelamentos devido ao sinal. Para o exemplo mencionado, obteríamos um erro da seguinte forma:

$$\begin{aligned} Erro_{peixes} &= \frac{(40-33)^2}{33} + \frac{(32-33)^2}{33} + \frac{(28-33)^2}{33} \\ &= 1.48 + 0.030 + 0.75 \\ &= 2.27 \end{aligned}$$

Claramente, se houvessémos observado *exatamente* as freqüências esperadas, então $Erro_{peixes} = 0$. A questão agora passa a ser como determinar o quão significativo é um erro da ordem de 2.27. Uma maneira empírica de chegarmos à essa resposta é tomarmos um grande número de amostras de 100 peixes em um rio controlado, ou seja, de um rio no qual temos certeza de que as distribuições entre espécies são equiprováveis, e organizarmos um histograma relatando a freqüência com que cada erro ocorre. Sem dúvida, em algumas amostras teremos "azar" e acabaremos retirando muito mais peixes de uma espécie do que da outra, mas via de regra esse tipo de erro deve ser raro. Já erros pequenos, que condizem com amostras realmente retiradas de um rio com espécies equiprováveis, devem ser observados bem mais freqüentemente. Se fizéssemos essas medições para um grande número de amostras e desenhássemos o histograma de freqüências de erros, obteríamos algo como a figura 4.1.

A partir deste histograma, podemos perceber que um erro próximo a 2.27 deveria ser observado em aproximadamente 15% das amostras. Em se tratando de teste de hipóteses, normalmente buscamos confirmar ou refutar uma hipótese com um grau de certeza bem menor, da ordem de 5%. Em outras palavras, desejamos garantir que a variação medida entre valores esperados e observados ocorra, devido ao acaso, em apenas 5% dos casos. Ao usar um grau de certeza desta ordem, teríamos então 95% de certeza de que a hipótese de equiprobabilidade de espécies pode ser *rejeitada*.

4.2 Chi Quadrado

O tipo de montagem de histogramas citado anteriormente não é muito prático em situações reais. Felizmente, a estatística conta com um teste de hipóteses clássico chamado

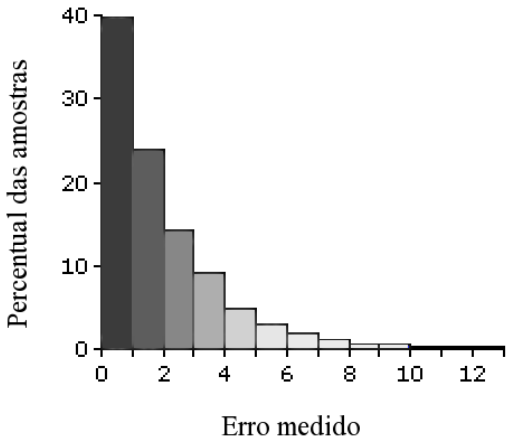

Figura 4.1: Histograma hipótetico após mensuração do erro em relação às proporções esperadas.

teste Chi Quadrado. Este teste, entre outras coisas, pode ser utilizado justamente para testar se a diferença de freqüência observadas em relação às freqüências esperadas se deve ou não ao acaso (KINNEY, 1997). Entretanto, com o Chi Quadrado não mais lidamos com histogramas, e sim com funções contínuas. Especificamente, a função densidade probabilidade da distribuição Chi Quadrado é dada pela seguinte equação:

$$f(x;k) = \frac{(1/2)^{k/2}}{\Gamma(k/2)} x^{k/2-1} e^{-x/2} \tag{4.1}$$

onde k é o número de graus de liberdade envolvidos no problema, x é o valor medido para a variável χ^2 (ver equação (4.2)) e Γ é a função Gamma. Intuitivamente, o número de graus de liberdade corresponde ao número de categorias que esperamos encontrar, menos um. No exemplo dos peixes, temos três categorias (espécies A, B e C). Embora pudéssemos esperar três graus de liberdade, isso não acontece, pois dado o tamanho da amostra e as contagens de peixes em duas das espécies, o número de peixes na terceira fica automaticamente determinado. Portanto, para o problema do peixe, estaríamos em frente a um cenário com 2 graus de liberdade.

A grande utilidade da equação (4.1) é que ela aproxima o tipo de histograma que esperaríamos obter empiricamente, dado um número de categorias (ou, equivalentemente, do número de graus de liberdade), para qualquer tipo de medição. Por essa razão, a equação (4.1) na verdade define uma *família* de curvas, cada qual representando uma função densidade probabilidade. Na figura 4.2, apresentamos a forma como essas curvas se comportam em função do número de graus de liberdade.

Um teste de hipóteses utilizando Chi Quadrado poderia se dar da seguinte forma. Em primeiro lugar, determinamos o número de graus de liberdade k da situação a ser testada. Após, calculamos o valor de erro, também conhecido na literatura como χ^2, de acordo com:

$$\chi^2 = \sum_i \frac{(observado_i - esperado_i)^2}{esperado_i} \tag{4.2}$$

Sabendo a forma com que os valores χ^2 são distribuídos para problemas com k graus de liberdade, a partir da equação (4.1), podemos agora calcular a chance de que esse

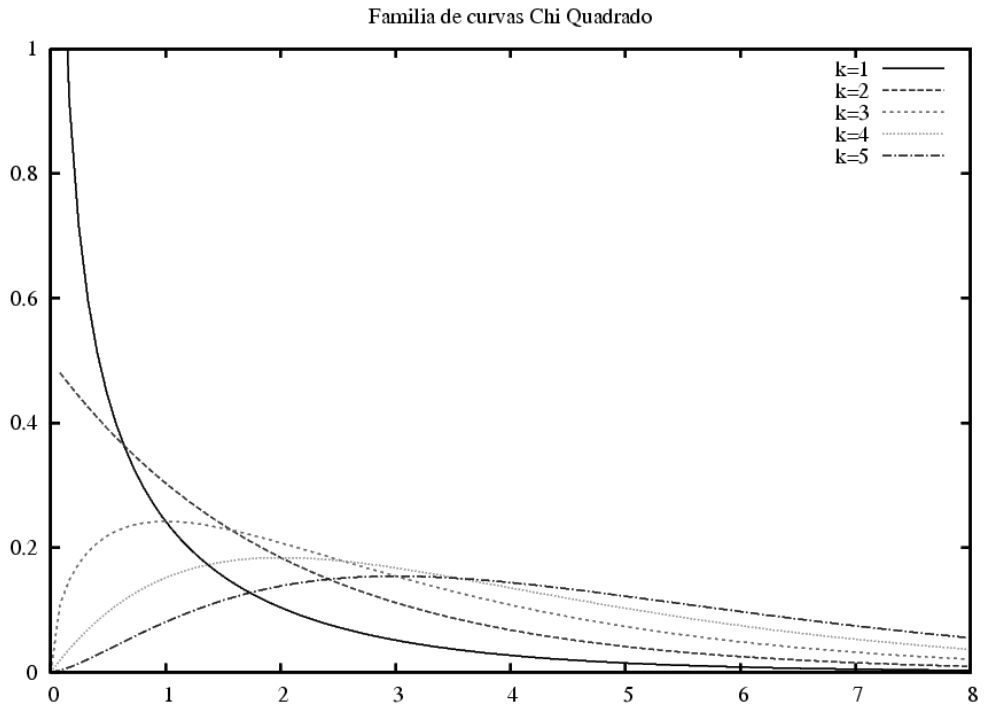

Figura 4.2: Família de curvas Chi Quadrado para vários graus k de liberdade (funções densidade probabilidade χ^2).

valor, ou algum maior, tenha sido obtido somente devido ao acaso. Para tanto, basta calcular a área sob a curva da equação (4.1) após o ponto χ^2. No caso do problema dos peixes, em que temos 2 graus de liberdade, a probabilidade de que obtenhamos um valor $\chi^2 >= 2.27$ é de aproximadamente 32%. Uma vez que existe praticamente uma chance em três de medirmos um valor $\chi^2 >= 2.27$, não podemos afirmar categoricamente que a distribuição de peixes $[A = 40, B = 32, C = 28]$ difere significativamente de $[A = 33, B = 33, C = 33]$. Portanto, o teste de hipóteses Chi Quadrado, para esse problema, não foi capaz de refutar a idéia de que as amostras vêm da mesma distribuição subjacente.

4.3 Detecção de Contexto via testes de hipóteses clássicos

O algoritmo RL-CD, proposto na seção 3.2, faz uso de uma variável chamada $N(s, a)$ a fim de manter contagens de visitação a determinado par estado-ação (s, a). Uma vez que o RL-CD armazena estas contagens de visitação, coloca-se a questão sobre se seria possível utilizar o teste Chi Quadrado para detectar trocas de contexto. Em outras palavras, queremos determinar se seria possível analisar, via Chi Quadrado, a hipótese de que determinadas freqüências de visitação mudaram significativamente entre dois períodos de tempo, a ponto de podermos inferir, com algum grau de certeza, que a matriz de transições também foi alterada. Embora essa idéia pareça interessante a primeira vista, existem alguns motivos pelos quais *não* podemos utilizar o teste Chi Quadrado diretamente na detecção de contextos.

Em primeiro lugar, a curva teórica dada pela equação (4.1) só constitui uma boa apro-

ximação para o caso discreto[1] quando as amostras são de tamanho grande. Em geral, os estatísticos sugerem empiricamente que as amostras devam ter tamanho maior ou igual a 10. Uma vez que no aprendizado por reforço alguns estados podem ser pouco visitados, concluímos que a aproximação dada por (4.1) pode nem sempre ser válida.

Isso se torna ainda mais crítico quando pensamos que, a fim de obedecer a regra das amostras de tamanho 10, o teste de hipóteses só poderia ser efetuado quando garantíssemos que *todos* os possíveis destinos a partir de determinado par estado-ação tivessem sido visitados ao menos 10 vezes. Não somente isso é irreal, como, em alguns casos, dependendo das distribuições de probabilidade, impossível[2].

O segundo motivo pelo qual não podemos utilizar o Chi Quadrado para a detecção de contextos é que ele não é bem adaptado para tarefas *online*. Isso ocorre porque, a fim de aplicar o Chi Quadrado neste tipo de tarefa, teríamos que colher amostras de freqüências de visitações em 2 períodos de tempos distintos, e então compará-las posteriormente. Em outras palavras, o agente precisaria, por exemplo, experimentar o mundo e coletar estatísticas de visitação por 100 iterações, a fim de compor a amostra de freqüências #1. Após este período, o agente coletaria ainda mais estatísticas de visitação, por mais 100 iterações, compondo a amostra #2. O teste de hipóteses seria feito, então, sobre essas duas amostras.

Definir qual o tamanho exato de cada período de amostragem é um problema complicado. Podem ocorrer trocas de contexto *durante* o período em que o agente coleta dados para cada amostra. Nesse caso, a amostra conteria dados mistos, ao invés de contagens advindas de apenas de um regime. Pior ainda, o agente só detectaria a troca de contexto após o tempo total de experimentação necessário para gerar ambas as amostras. A solução para esse impasse seria, obviamente, diminuir o tempo de experimentação para coletagem das amostras. Com isso, minimizaríamos a chance de que trocas de contexto acontecessem durante tal período. Infelizmente, quanto menor for o período de experimentação para cada amostra, tanto menor será a chance de que alterações significativas nas probabilidades possam ser detectadas.

Para exemplificar este segundo problema, suponha que o agente coletou a seguinte amostra #1 de freqüências de visitações, em um sistema de três estados: $[s1 = 10, s2 = 24, s3 = 43]$. Vamos supor agora que o período de experimentação para definição da segunda amostra é de apenas 2 iterações. O segundo vetor de visitações teria, no pior caso, a freqüência de apenas um de seus estados destino incrementada em duas unidades (ex: amostra #2 = $[s1 = 12, s2 = 24, s3 = 43]$). Certamente, a comparação entre essas amostras não geraria um valor χ^2 muito alto, mesmo que a dinâmica do ambiente tivesse de fato sido alterada. De fato, para este exemplo, independentemente do fato de ter ou não havido uma troca de contextos, o valor calculado seria $\chi^2 = 0.4$. Consultando as tabelas do Chi Quadrado, descobrimos que $\chi^2 = 0.4$ nos permite refutar a hipótese de que *houve* alteração nas probabilidades de próximo estado com certeza de apenas 18%.

Por fim, a terceira razão pela qual o Chi Quadrado não pode ser usado diretamente é a mais evidente de todas: a equação (4.2) exige uma divisão pela freqüência esperada. Para o caso de distribuições de probabilidade quaisquer, nas quais podem haver probabilidades

[1] Isto é, para aqueles casos em que as freqüências são divididas em categorias mutuamente exclusivas.

[2] Alguns autores sugerem que, na presença de amostras pequenas, algumas categorias sejam unificadas em uma "meta-categoria", a qual possuiria o tamanho mínimo exigido. Isso equivaleria, no caso do RL-CD, a unificar as contagens de freqüência de visitações a partir de um estado-ação. Entretanto, essa solução acabaria trazendo ainda mais problemas, tais como: Qual o critério para unificar os n estados destinos subamostrados? O que fazer quando nem a unificação de todos os estados existentes é capaz de alcançar o tamanho mínimo de amostras exigido? etc.

zero de transição para certos estados, incorreríamos em uma divisão por zero durante o cálculo de χ^2.

4.4 Estimativa Monte Carlo da distribuição de e^T

Uma vez que o teste Chi Quadrado não pode ser utilizado diretamente para a detecção de contextos, podemos supor que existam outros testes estatísticos que se apliquem. Entretanto, a maioria dos testes que comparam duas amostras falha caso as contagens possam ser pequenas ou de tamanho zero. Portanto, precisamos desenvolver um teste de hipóteses especificamente para o RL-CD.

A solução que iremos adotar consiste em estimar empiricamente a distribuição de probabilidades para os valores de qualidade instantânea[3]. Para isso, utilizaremos simulações Monte Carlo, as quais serão detalhadas nas duas próximas subseções. Serão estimadas duas famílias de curvas: **1)** a família de curvas da distribuição de probabilidades para os valores de qualidade, no caso de *não* ter havido troca de contexto; e **2)** a distribuição de probabilidades para os valores de qualidade, para o caso de *ter havido* uma troca de contexo. Essas famílias de curvas nos permitirão tratar a detecção de contextos como um teste de hipóteses clássico, e também acabarão por indicar o melhor valor teórico para o parâmetro de qualidade mínima aceitável, E_{min}.

4.4.1 Distribuição de probabilidades da qualidade – caso sem troca de contexto

A fim de estimar a distribuição de probabilidades para os valores de qualidade, precisamos obter uma amostra representativa das freqüências com que cada valor de qualidade ocorre, considerando todas as possíveis distribuições de probabilidades para n possíveis próximos estados, dado que o agente se encontre em um par estado-ação qualquer.

A primeira alternativa para isso seria enumerar as possíveis distribuições para n-próximos estados. Obviamente, existem infinitas possibilidades de distribuições discretas de probabilidade para n-próximos estados, uma vez que também existem infinitas combinações de n números reais que somam um. Por essa razão, a fim de enumerar as possíveis distribuições, poderíamos considerar que, ao invés de os valores de probabilidade poderem variar continuamente entre zero e um, eles poderiam assumir apenas um conjunto de valores finitos, incrementados em passos de tamanho ϕ. Por exemplo, se $\phi = 0.25$, estaríamos assumindo que uma probabilidade de próximo estado não mais poderia variar continuamente no intervalo $[0, 1]$, e sim que ocorreria apenas em intervalos fixos de 0.25: poderia valer 0, ou 0.25, ou 0.50, ou 0.75, ou 1.0. Já se $\phi = 0.10$, os possíveis valores de probabilidade poderiam ser 0, 0.1, 0.2, $\ldots$, 0.9 ou 1.0. Note que conforme $\phi \rightarrow 0$, os possíveis valores de probabilidade tendem ao intervalo "correto", isto é, aos valores distribuídos no intervalo real de zero a um. Dito de outra forma, valores altos de ϕ implicam aproximações grosseiras para os possíveis valores que uma probabilidade pode assumir; valores baixos de ϕ implicam aproximações melhores para tais valores. Por esta razão, chamaremos ϕ de *valor de precisão*. Se enumerarmos as possíveis distribuições de próximo estado utilizando alta precisão (valores pequenos de ϕ), estaremos enumerando um grande número de possíveis distribuições, e consequentemente obteremos uma aproximação mais detalhada da real distribuição de valores de qualidade.

[3]Note que se conhecêssemos ao menos *qual* a forma das funções de densidade de probabilidade de e^T, poderíamos usar técnicas de ajuste de funções. Entretanto, como não conhecemos a forma ou a família dessas distribuições, tentaremos estimar diretamente a forma de tais famílias de distribuições usando um algoritmo Monte Carlo.

A fim de clarificar como esta enumeração poderia ocorrer, considere o seguinte exemplo. Independentemente de qual é o par estado-ação atual do agente (assuma um par (s, a) qualquer), suponha que queiramos estimar a distribuição de probabilidades de próximo estado para um sistema com três estados. Suponha que, para isso, desejemos calcular os possíveis valores de qualidade para todas as possíveis distribuições de próximo estado geradas com precisão de $\phi = 0.5$. Nesse caso, as possíveis distribuições de próximo estado, para o sistema de três estados, seriam as seguintes:

1. $[s1 = 0.0, s2 = 0.0, s3 = 1.0]$
2. $[s1 = 0.0, s2 = 0.5, s3 = 0.5]$
3. $[s1 = 0.0, s2 = 1.0, s3 = 0.0]$
4. $[s1 = 0.5, s2 = 0.0, s3 = 0.5]$
5. $[s1 = 0.5, s2 = 0.5, s3 = 0.0]$
6. $[s1 = 1.0, s2 = 0.0, s3 = 0.0]$

Para cada uma destas *seis* distribuições de probabilidade de próximo estado, precisamos calcular, de acordo com a equação (3.7), todos os possíveis valores de qualidade instantânea. Considere como exemplo o cálculo das possíveis qualidades instantâneas obtidas a partir da quarta distribuição enumerada, isto é, de $[s1 = 0.5, s2 = 0.0, s3 = 0.5]$. Nesse caso, seriam três os possíveis valores de qualidade instantânea, dependendo o estado destino observado. Caso o estado destino observado seja $s1$, denotamos a qualidade instantânea calculada por e^T_{s1}:

$$
\begin{aligned}
e^T_{s1} &= 1 - 2\left(\frac{1}{2}(N(s,a)+1)^2\right)\sum_{\kappa \in \mathcal{S}} \Delta T_m(\kappa)^2 \\
&= 1 - 2\left(\frac{1}{2}(N(s,a)+1)^2\right)\left(\left(\frac{1.0-0.5}{N(s,a)+1}\right)^2 + \left(\frac{0.0-0.0}{N(s,a)+1}\right)^2 + \left(\frac{0.0-0.5}{N(s,a)+1}\right)^2\right) \\
&= 1 - \Big((1.0-0.5)^2 + (0.0-0.0)^2 + (0.0-0.5)^2\Big) \\
&= 1 - \Big(0.25 + 0.0 + 0.25\Big) \\
&= 0.5
\end{aligned}
$$

Caso o estado destino observado seja $s2$, a qualidade instantânea e^T_{s2} será:

$$
\begin{aligned}
e^T_{s2} &= 1 - 2\left(\frac{1}{2}(N(s,a)+1)^2\right)\sum_{\kappa\in\mathcal{S}}\Delta T_m(\kappa)^2 \\
&= 1 - 2\left(\frac{1}{2}(N(s,a)+1)^2\right)\left(\left(\frac{0.0-0.5}{N(s,a)+1}\right)^2 + \left(\frac{1.0-0.0}{N(s,a)+1}\right)^2 + \left(\frac{0.0-0.5}{N(s,a)+1}\right)^2\right) \\
&= 1 - \Big((0.0-0.5)^2 + (1.0-0.0)^2 + (0.0-0.5)^2\Big) \\
&= 1 - \Big(0.25 + 1.0 + 0.25\Big) \\
&= 1.5
\end{aligned}
$$

Caso o estado destino observado seja $s3$, a qualidade instantânea e^T_{s3} será:

$$
\begin{aligned}
e^T_{s3} &= 1 - 2\left(\frac{1}{2}(N(s,a)+1)^2\right)\sum_{\kappa\in\mathcal{S}}\Delta T_m(\kappa)^2 \\
&= 1 - 2\left(\frac{1}{2}(N(s,a)+1)^2\right)\left(\left(\frac{0.0-0.5}{N(s,a)+1}\right)^2 + \left(\frac{0.0-0.0}{N(s,a)+1}\right)^2 + \left(\frac{1.0-0.5}{N(s,a)+1}\right)^2\right) \\
&= 1 - \Big((0.0-0.5)^2 + (0.0-0.0)^2 + (1.0-0.5)^2\Big) \\
&= 1 - \Big(0.25 + 0.0 + 0.25\Big) \\
&= 0.5
\end{aligned}
$$

Evidentemente, não devemos esperar que as qualidades e^T_{s1}, e^T_{s2} e e^T_{s3} sejam observadas com freqüências idênticas. Se assumirmos que *não* houve uma troca de contexto, então as probabilidades de transição observadas serão as mesmas probabilidades de transição estimadas pela distribuição em questão (no caso, $[s1 = 0.5, s2 = 0.0, s3 = 0.5]$). Isso implica que cada um dos valores de qualidades e^T_i será observado com freqüência proporcional à probabilidade de observarmos o estado destino i. Tendo isso em mente, podemos esperar que:

1. 50% das transições levem a $s1$;
2. 0% das transições levem a $s2$;
3. 50% das transições levem a $s3$.

Portanto, para cada k visitações do par estado-ação atual do agente, esperaríamos observar, assumindo que não ocorreu troca de contexto:

1. $0.5k$ valores de qualidade e^T_{s1};
2. $0.0k$ valores de qualidade e^T_{s2};
3. $0.5k$ valores de qualidade e^T_{s3}.

Se enumerarmos as possíveis distribuições de probabilidade de próximo estado utilizando alguma precisão ϕ qualquer, podemos anotar as freqüências esperadas de cada valor de qualidade em um histograma. Este histograma seria uma boa aproximação da curva real de distribuição de e^T, para o caso de não ter havido troca de contexto (i.e, para quando as probabilidades observadas são iguais às estimadas). Quanto menor o valor da precisão ϕ, mais distribuições possíveis de probabilidade iremos considerar, e portanto melhor será a aproximação da curva.

O primeiro problema com essa abordagem é a exigência de enumerar todas as possíveis distribuições para n próximos estados. Esse fato é ainda mais crítico uma vez que, para obter boas aproximações, precisaríamos enumerar as distribuições com uma precisão ϕ bem maior que 0.5. Em outras palavras, para um sistema de três estados e $\phi = 0.5$, estaremos calculando os possíveis valores de qualidade para apenas seis dentre todas as infinitas distribuições de próximo estado. A fim de obter um histograma preciso da distribuição dos possíveis valores de qualidade, queremos utilizar valores de ϕ tão pequenos quanto possível, de modo a calcular tais valores para uma grande quantidade de possíveis distribuições de próximo estado. Entretanto, a quantidade destas possíveis distribuições cresce exponencialmente com o número de estados e com a precisão adotada. Portanto, claramente não podemos utilizar a abordagem de enumeração exaustiva de distribuições. A solução adotada neste trabalho será a de *amostrar* uniformemente dentre elas.

O problema correspondente a amostrar uniformemente dentre as possíveis distribuições de probabilidades para n próximos estados é o de amostrar pontos uniformes no subespaço de vetores n-dimensionais de norma-1. Este problema é conhecido como *Seleção de Pontos no Simplex Unitário*[4], e consiste em amostrar uniformemente pontos dentro de um objeto matemático conhecido como *simplex*. Resumidamente, um simplex em $\Re^n$ é invólucro convexo de $(n+1)$ pontos independentes. Para duas dimensões, o simplex unitário consiste no triângulo delimitado pela origem e pelos pontos $(0, 1)$ e $(1, 0)$ no plano cartesiano. A idéia se estende para mais dimensões: em $\Re^3$, o simplex é um tetraedro de base triangular; em $\Re^4$, o simplex é um pentatopo; e assim por diante.

A solução mais simples que encontramos para amostrar uniformemente de um simplex unitário n-dimensional deriva do fato de que tal processo é equivalente a amostrar de uma distribuição de Dirichlet com parâmetros $\alpha = (\alpha_1, \ldots, \alpha_n)$, todos iguais a um. O procedimento exato para isso consiste em amostrar pontos a partir de uma distribuição exponencial e depois normalizá-los, ou seja, dividir cada um pela soma total. Uma maneira simples de amostrar um valor j a partir uma exponencial é gerar um valor i uniforme no intervalo aberto $(0, 1]$ e transformá-lo tal que $j = -ln(i)$. Assim, se precisarmos amostrar uniformemente dentre todas distribuições de probabilidades para n-próximos estados, podemos equivalentemente amostrar um vetor n-dimensional dentre todos os possíveis vetores de norma-1. Uma solução para isso é amostrar n valores de uma exponencial, normalizá-los, e considerar cada valor como uma coordenada do vetor de norma-1. Ao procedimento que efetua este cálculo iremos chamar *amostra_possivel_distribuicao(n)*, onde n é o número de dimensões desejado (no caso do RL-CD, o número de possíveis próximos estados). Este procedimento é descrito no algoritmo 13.

Tendo como base o procedimento eficiente apresentado no algoritmo 13, não mais precisamos enumerar todos as possíveis distribuições de próximo estado. Basta amostrarmos um grande número destas distribuições, através de *amostra_possivel_distribuicao(n)* (algoritmo 13), e então organizarmos um histograma contendo as freqüências esperadas para cada valor de qualidade possível. O método Monte Carlo que implementa essa idéia

[4]do inglês, *unit-Simplex Point Picking*.

Algoritmo 13 Amostra uma distribuição de probabilidades de n-próximos estados uniformememente. Método equivalente à Seleção de n-Pontos no Simplex Unitário.

Seja n o número de próximos estados existentes. O algoritmo irá amostrar uma possível distribuição de probabilidades sobre n-próximos estados dentre todas as possíveis distribuições.

Seja (s, a) o par estado-ação atual. Para fins de amostragem de freqüências esperadas de qualidades, pode ser qualquer.

Seja $\hat{T}(s, a, \cdot)$ a distribuição de probabilidades que será amostrada.

1: $total = 0$
2: **para todo** k variando de 1 até n **faça**
3: $v \leftarrow$ valor uniforme entre $(0, 1]$
4: $prob_k = -ln(v)$
5: $total \leftarrow total + prob_k$
6: **fim para**
7: **para todo** k variando de 1 até n **faça**
8: $\hat{T}(s, a, k) = prob_k / total$
9: **fim para**
10: Retorne a distribuição amostrada $\hat{T}(s, a, \cdot)$.

é capaz de obter boas aproximações para a distribuição de valores de qualidade, sob a hipótese de que não ocorreu uma troca de contexto. O processo exato é apresentado no algoritmo 14.

Algoritmo 14 *Estimação Monte Carlos para distribuição de e_T para casos em que não há troca de contexto*

Seja $|S|$ o número de possíveis próximos estados.

Seja $n\_amostras$ o número de amostras de possíveis distribuições $|S|$-dimensionais que desejamos tomar.

Seja $hist$ o histograma de freqüências esperadas para cada possível valor de qualidade.

1: **para todo** i variando de 1 até $n\_amostras$ **faça**
2: $distribuicao\_esperada = amostra\_possivel\_distribuicao(|S|)$ (alg. 13)
3: **para todo** j variando de 1 até $|S|$ **faça**
4: Considere que as probabilidades estimadas pelo agente são dadas por $distribuicao\_esperada$, para um par estado-ação atual qualquer.
5: Considere $qualidade_j$ o valor de qualidade caso o j-ésimo estado em S seja o estado destino observado.
6: Suponha a ocorrência de K visitas ao par estado-ação atual. Então: $hist[qualidade_j] \leftarrow hist[qualidade_j] + K * distribuicao\_esperada[j]$
7: **fim para**
8: **fim para**

O algoritmo 14 aproxima arbitrariamente bem a a distribuição de valores de qualidade, para o caso de não ter havido troca de contexto, dado que os valores de amostras utilizadas possam ser tornados tão grandes quanto se queira. Note também que a curva específica sendo aproximada depende essencialmente de quantos próximos estados estamos considerando. Analisando a equação (3.7), percebemos que se o sistema for aumentado com κ estados adicionais, todos com probabilidade 0% de ocorrerem, o valor de e^T não será alterado. Portanto, devemos considerar que $|S|$, no algoritmo 14, diz respeito aos

próximos estados *possíveis*, isto é, para os quais há uma probabilidade não-zero.

A família de curvas para a distribuição sobre valores de qualidade, assumindo não ter havido trocas de contexto, é apresentada na figura 4.3. Note que as curvas nesta figura são assimétricas, com a cauda voltada para a qualidade $+1$ (lembre que a equação (3.7) escala as qualidade instantâneas de previsão para o intervalo $[-1, +1]$, onde -1 indica a pior qualidade de previsão possível, e $+1$ a melhor). Isso indica que, caso não tenha havido uma troca de contexto, a probabilidade maior é de que a qualidade se aproxime de $+1$. Valores de qualidade baixos são muito raros, mas possíveis; um exemplo disso são as qualidades geradas por distribuições nas quais um estado destino, embora passível de ser alcançado, o é apenas com probabilidade baixíssima.

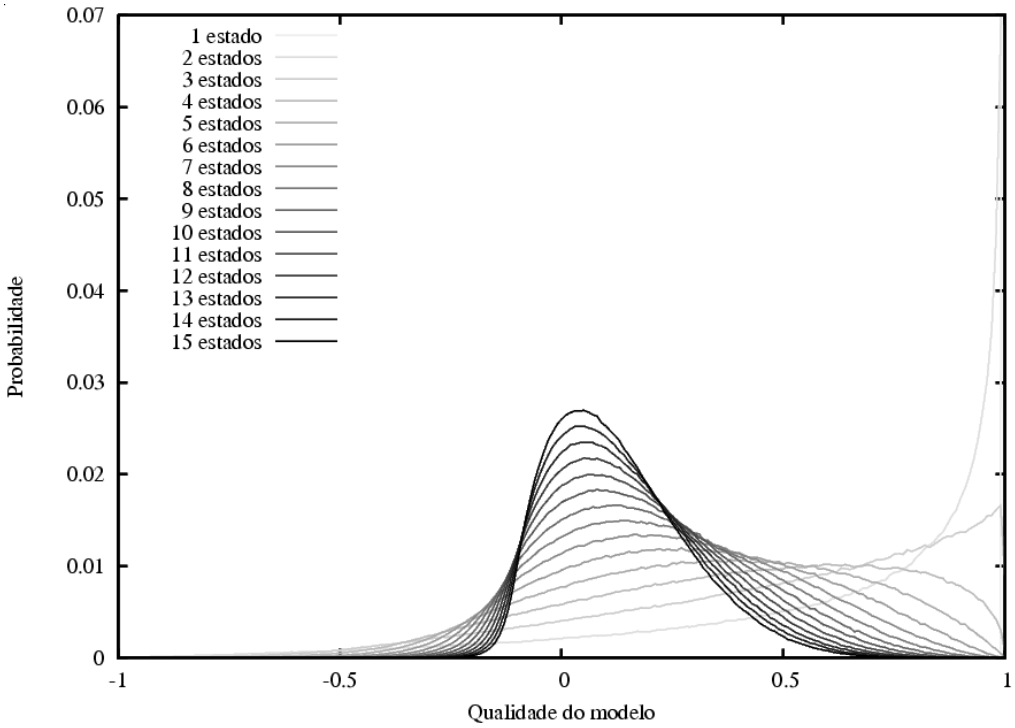

Figura 4.3: Distribuição de probabilidade para valores de qualidade do modelo, caso não haja troca de contexto.

A seguir, na figura 4.4, apresentamos a evolução da distribuição de valores de qualidade conforme o número de estados cresce significativamente. Pode-se perceber que a curva fica cada vez mais estreita, e que aparentemente converge para sua média. Isso sugere que a distribuição está convergindo para uma similar ao Delta de Dirac; no entanto, a descoberta da forma analítica exata da distribuição não é nosso objetivo principal.

O fato de que a curva 4.4 tende a tornar-se um impulso sugere que, caso não haja troca de contextos, a qualidade tenderá a 0 pelo intervalo positivo. Esse resultado faz sentido ao considerarmos que, dentre todas as possíveis distribuições de probabilidade de próximo estado, pouquíssimas concentram suas probabilidades em apenas um (ou poucos) estado. A grande maioria associa pequenas "porções" da probabilidade a todo espaço de próximos estados. Em outras palavras, a maioria das possíveis distribuições de próximo estado, uniformemente amostradas, se assemelha mais à distribuição equiprovável do que à uma distribuição determinística. De fato, se tivermos $|S|$ estados equiprováveis, a qualidade instantânea, independentemente do estado destino visitado, será dada por:

$$
\begin{aligned}
e^T &= 1 - \Big((1 - \frac{1}{|S|})^2 + (|S| - 1)(\frac{-1}{|S|})^2\Big) \\
&= 1 - (1 - \frac{1}{|S|}) \\
&= \frac{1}{|S|}
\end{aligned}
$$

o que confirma nossa suposição de que a distribuição de valores de qualidade tende à distribuição equiprovável, e também de que esta distribuição de fato se aproxima de zero pelo intervalo positivo sempre que $|S| \rightarrow \infty$.

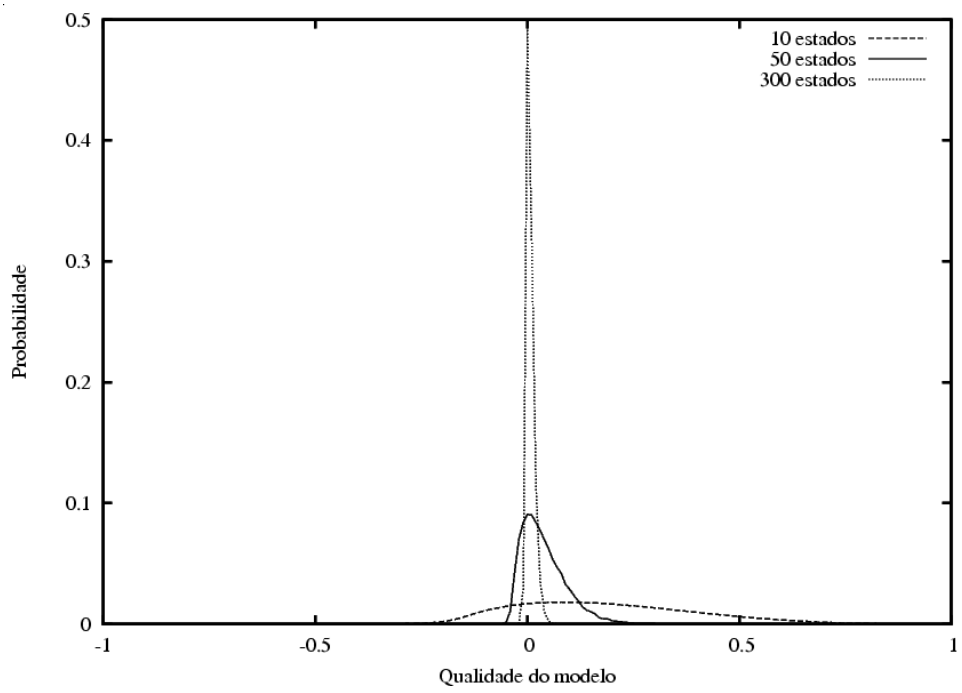

Figura 4.4: Distribuição de probabilidade para valores de qualidade do modelo, caso não haja troca de contexto (limite).

Um resultado importante sugerido pelas figuras 4.3 e 4.4 é que devemos manter o valor de E_{min} (qualidade mínima aceitável de previsão) o mais "à esquerda" possível daquelas curvas. Intuitivamente, queremos que a qualidade mínima aceitável seja *menor* que a maior parte do corpo da distribuição de qualidades obtida na ausência de troca de contextos. Quanto maior for E_{min}, maior será a chance de que uma falsa troca de contexto seja detectada. Isso ocorreria simplesmente pelo fato de que o agente passaria a considerar valores de qualidade típicos de um ambiente estacionário como valores significativamente *baixos* de qualidade.

4.4.2 Distribuição de probabilidades da qualidade – caso com troca de contexto

Nosso próximo passo na análise estatística de e^T é adaptar o algoritmo 14 a fim de aproximar as distribuições de valores de qualidade caso *tenham ocorrido trocas de contexto*. A modificação necessária é relativamente simples. Para cada distribuição uniformemente amostrada através da rotina $amostra\_possivel\_distribuicao(|S|)$, obteremos

outras $n\_subamostras$ amostras de distribuições, as quais representarão possíveis contextos que viriam a seguir a atual. Este novo algoritmo, apresentado em 15, estima, para um grande número de possíveis contextos, quais seriam as freqüências de qualidades observadas caso o contexto viesse a ser substituído. Os resultados da simulação Monte Carlo produzidas pelo algoritmo 15 são apresentadas nas figuras 4.5 e 4.6.

Algoritmo 15 *Estimação Monte Carlo para distribuição de e_T para casos em que há troca de contexto*

Seja $|S|$ o número de possíveis próximos estados.
Seja $n\_amostras$ o número de amostras de possíveis distribuições $|S|$-dimensionais que desejamos tomar.
Seja $n\_subamostras$ o número de amostras de possíveis distribuições $|S|$-dimensionais de próximos contextos.
Seja $hist$ o histograma de freqüências esperadas para cada possível valor de qualidade.

para todo i variando de 1 até $n\_amostras$ **faça**
  $distribuicao\_esperada = amostra\_possivel\_distribuicao(|S|)$
  para todo k variando de 1 até $n\_subamostras$ **faça**
    $distribuicao\_observada = amostra\_possivel\_distribuicao(|S|)$
    para todo j variando de 1 até $|S|$ **faça**
      Considere que as probabilidades estimadas pelo agente são dadas por $distribuicao\_esperada$, para um par estado-ação atual qualquer.
      Considere $qualidade_j$ o valor de qualidade caso o j-ésimo estado em S seja o estado destino observado.
      Suponha a ocorrência de K visitas ao par estado-ação atual. Então teremos, sob o k-ésimo contexto:
      $hist[qualidade_j] \leftarrow hist[qualidade_j] + K * distribuicao\_observada[j]$
    fim para
  fim para
fim para

Note que o comportamento das curvas apresentadas em 4.5 é semelhante ao representado na figura 4.4, ou seja, é semelhante àqueles casos em que não supunhamos ter ocorrido uma troca de contexto. Entretanto, as curvas em 4.5 estão levemente deslocadas em direção à qualidade -1, o que indica que, sob troca de contextos, existe uma maior chance de que os valores de qualidade sejam negativos. A sobreposição destas curvas pode ser vista nas figuras 4.7 e 4.8, que correspondem, respectivamente, às distribuições de probabilidade dos valores de qualidade para sistemas com 4 e 50 estados.

Um resultado importante também sugerido pelas figuras 4.5 e 4.6 é que devemos manter o valor de E_{min} o mais "à direita possível" daquelas curvas. Intuitivamente, queremos que a qualidade mínima aceitável seja *maior* que a maior parte do corpo da distribuição de qualidades obtida na presença de troca de contextos. Quanto menor for E_{min}, maior será a chance de que uma legítima troca de contexto deixe de ser detectada.

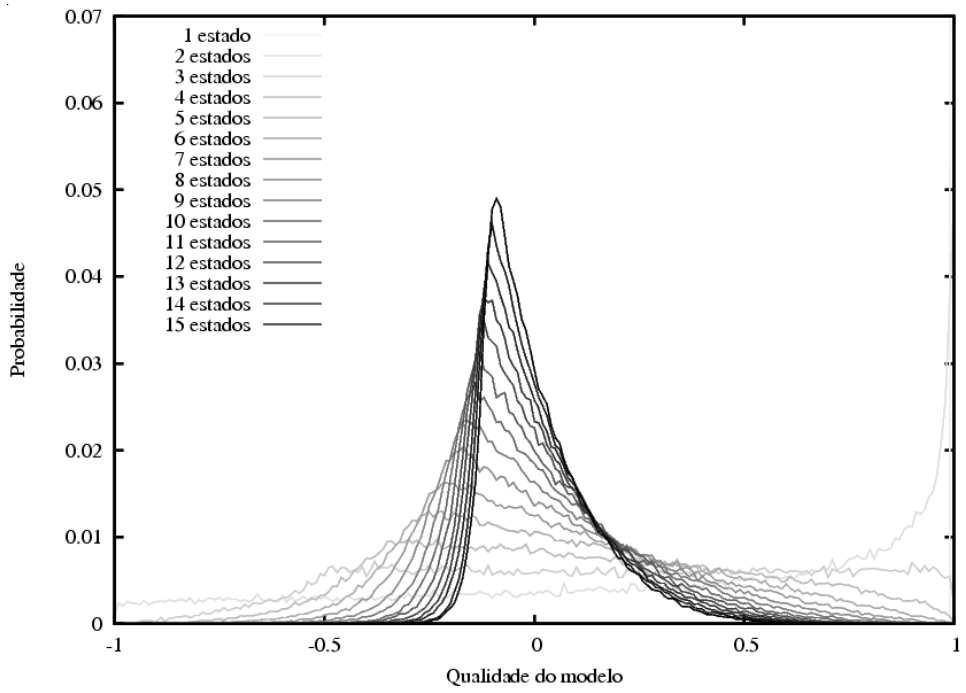

Figura 4.5: Distribuição de probabilidade para valores de qualidade do modelo, caso haja troca de contexto.

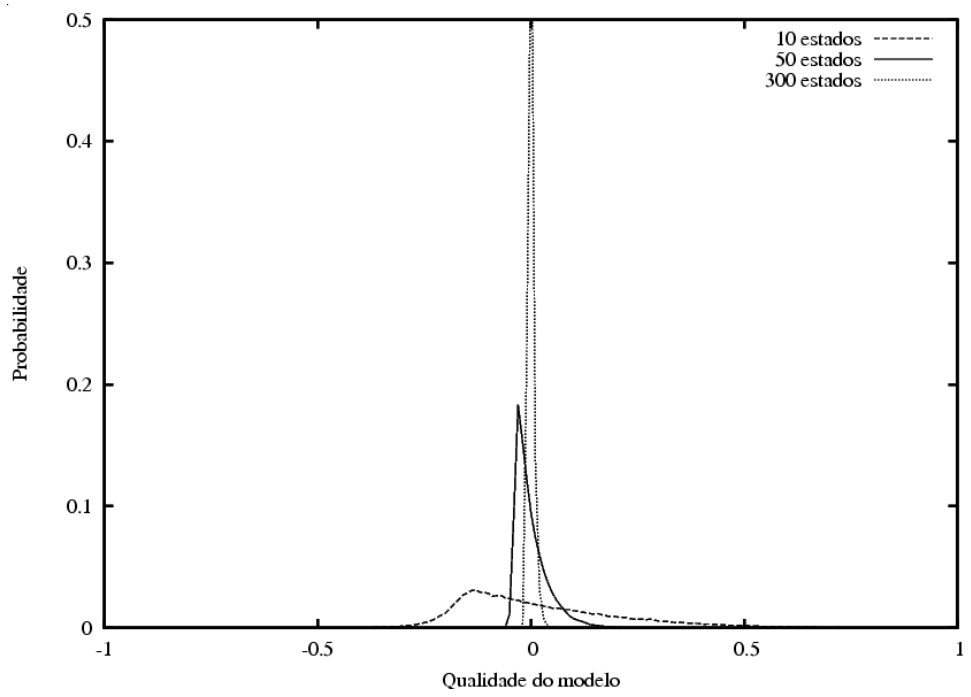

Figura 4.6: Distribuição de probabilidade para valores de qualidade do modelo, caso haja troca de contexto (limite).

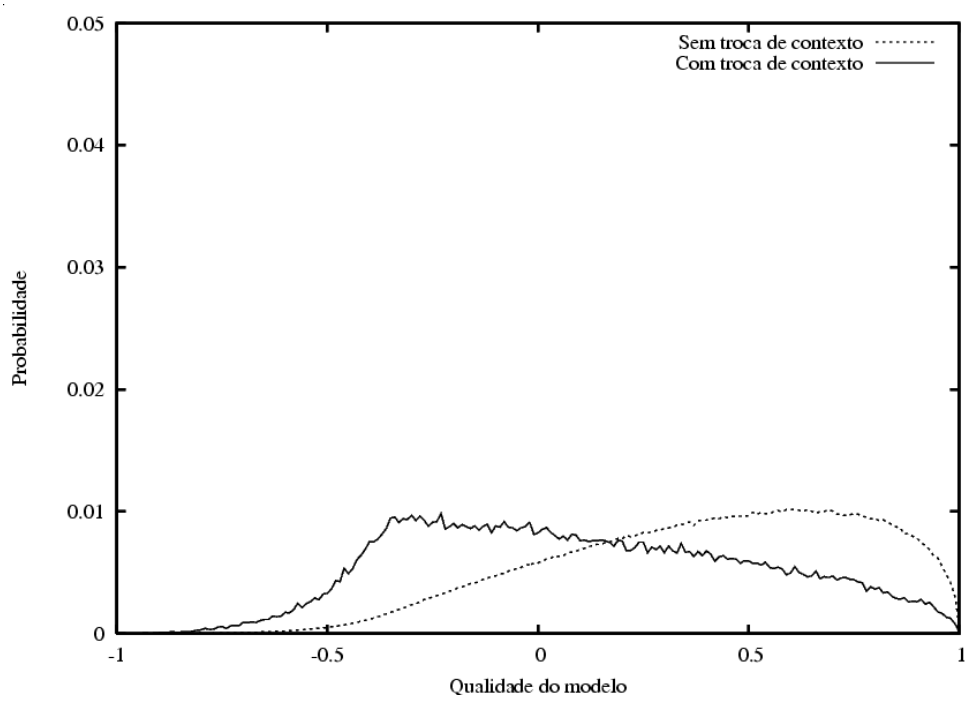

Figura 4.7: Comparação das distribuições de valores de qualidades sob a hipótese de troca de contextos, e sob a hipótese de sistema estacionário (4 estados).

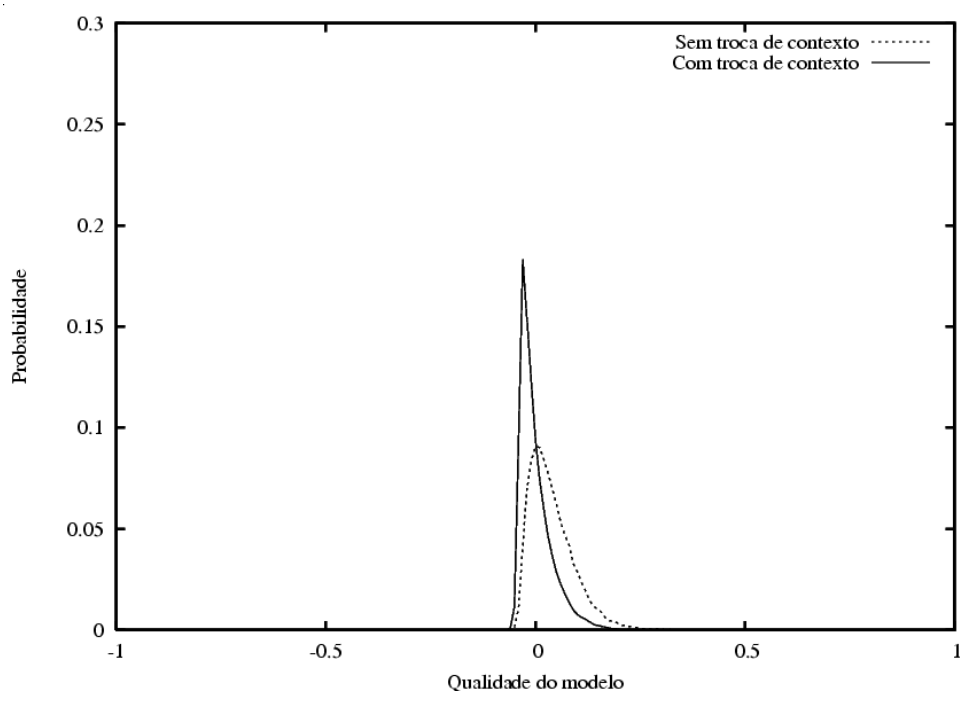

Figura 4.8: Comparação das distribuições de valores de qualidades sob a hipótese de troca de contextos, e sob a hipótese de sistema estacionário (50 estados).

4.4.3 Teste de hipóteses

Com base na família de curvas aproximada pelo algoritmo 14, poderíamos organizar a detecção de troca de contexto da seguinte maneira, utilizando a idéia geral de um teste de hipóteses:

- Seja H_0, a hipótese nula, correspondente à suposição de que não houve trocas de contexto;
- Nesse caso, a distribuição de valores de qualidade, sob H_0 e para um determinado número de estados $|S|$, será dada pelas curvas estimadas pelo algoritmo 14;
- Seja α um nível de significância, tipicamente 5%;
- Com base na distribuição de valores aproximada pelo algoritmo 14, encontre um valor crítico v tal que 95% da curva esteja a sua direita;
- Seja e^T o valor medido de qualidade instantânea;
- Se $e^T <= v$, rejeitamos H_0 com 95% de certeza.

Antes de adotarmos o procedimento acima como o teste de hipóteses correto para o RL-CD, considere alguns dos resultados mencionados anteriormente. Especificamente, considere que idealmente queremos que E_{min} fique o máximo possível *abaixo* da distribuição de qualidade na ausência de troca de contexto, e o máximo possível *acima* da distribuição da qualidade sob trocas de contexto. Se pudermos encontrar um valor E_{min}^{ideal} capaz de maximizar simultaneamente ambos esses critérios, teremos encontrado o valor de qualidade mínima aceitável que minimiza tanto a chance de falsos negativos quanto a chance de falsos positivos.

4.4.4 Valor ideal pra qualidade mínima

Com base no raciocínio recém apresentado, podemos encontrar o valor E_{min} ideal. Seja $f_{troca-contexto}$ a função densidade probabilidade para a variável de qualidade instantânea, obtida sob a hipótese de não ter havido troca de contexto. Essa função é estimada pelo algoritmo 14. Seja $f_{nao-troca-contexto}$ a função densidade probabilidade para a variável de qualidade instantânea, obtida sob a hipótese de ter havido troca de contexto. Essa função é estimada pelo algoritmo 15. A fim de minimizar a chance de falsos negativos e de falsos positivos simultaneamente, queremos encontrar o valor de qualidade E_{min}^{ideal} tal que:

$$E_{min}^{ideal} = \arg \max_{i \in [-1,+1]} \Big(\int_{-1}^{i} f_{troca-contexto}(x)dx + \int_{i}^{+1} f_{nao-troca-contexto}(x)dx \Big) \quad (4.3)$$

Gráficos das funções acumuladas de $f_{troca-contexto}$ e de $f_{nao-troca-contexto}$, assim como da soma de ambas, conforme a equação (4.3), são apresentados nas figuras 4.9, 4.10, 4.11 e 4.12, as quais representam respectivamente sistemas com 4, 10, 50 e 100 estados. Nessas figuras, o valor ideal E_{min}^{ideal} corresponde ao ponto máximo da curva rotulada por *soma*.

Por fim, ressaltamos o fato de que E_{min}^{ideal} parece convergir para um valor constante conforme o número de estados cresce. Especificamente, ao resolvermos a equação (4.3) para curvas geradas a partir de sistemas com vários números de estados, percebemos que

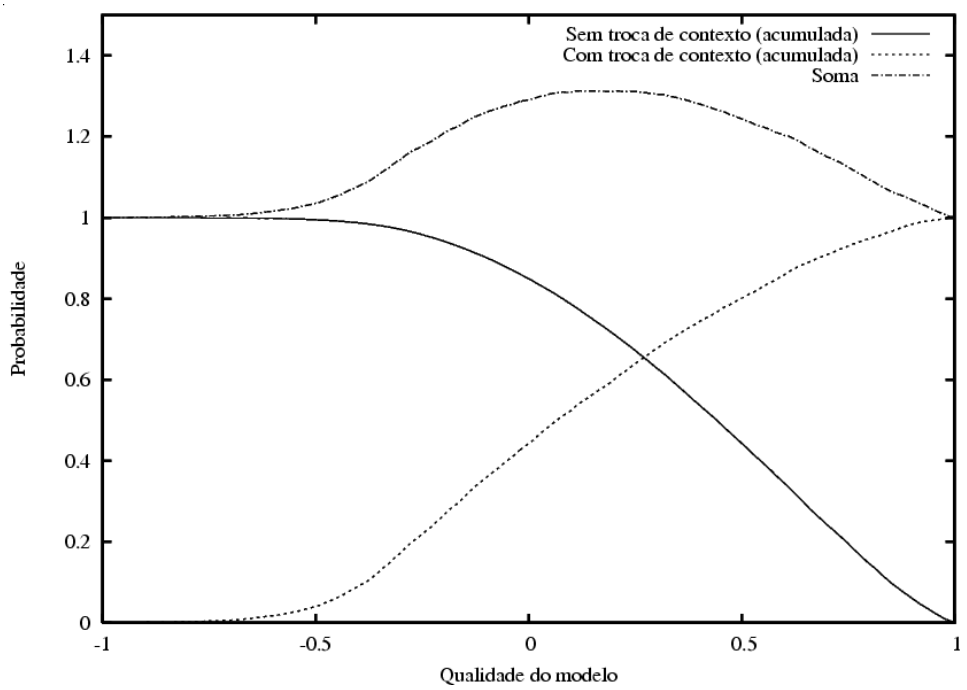

Figura 4.9: Distribuições cumulativas das qualidades – com e sem trocas de contexto (4 estados).

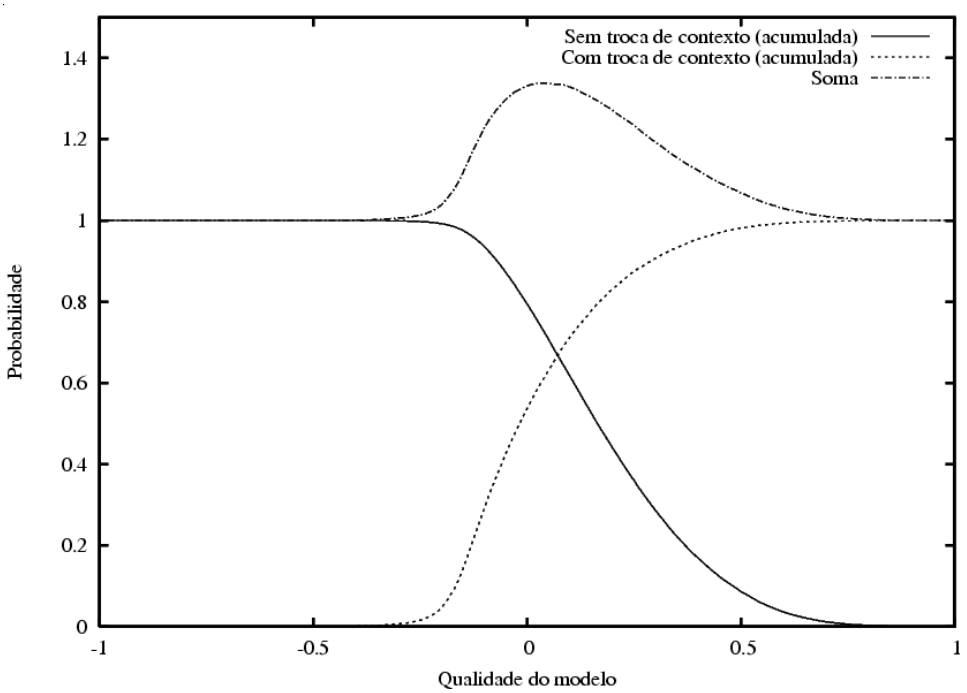

Figura 4.10: Distribuições cumulativas das qualidades – com e sem trocas de contexto (10 estados).

$E_{min}^{ideal} \rightarrow -0.01$ quando $|S| \rightarrow \infty$. A evolução de E_{min}^{ideal} em função do número de estados é apresentada na figura 4.13

Note que, embora -0.01 seja o valor teórico que minimiza simultaneamente as chan-

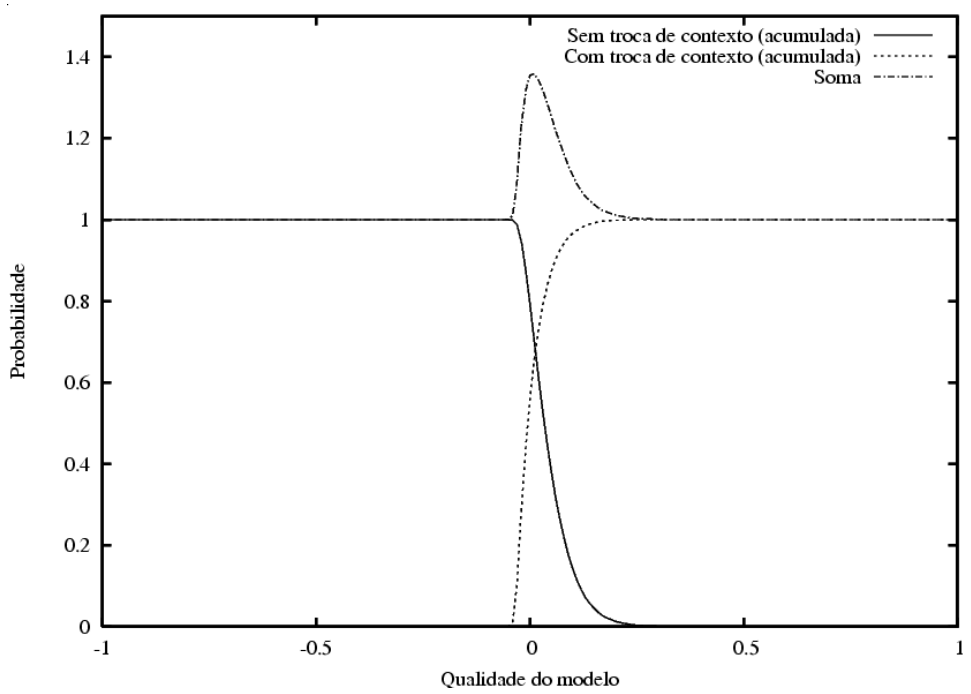

Figura 4.11: Distribuições cumulativas das qualidades – com e sem trocas de contexto (50 estados).

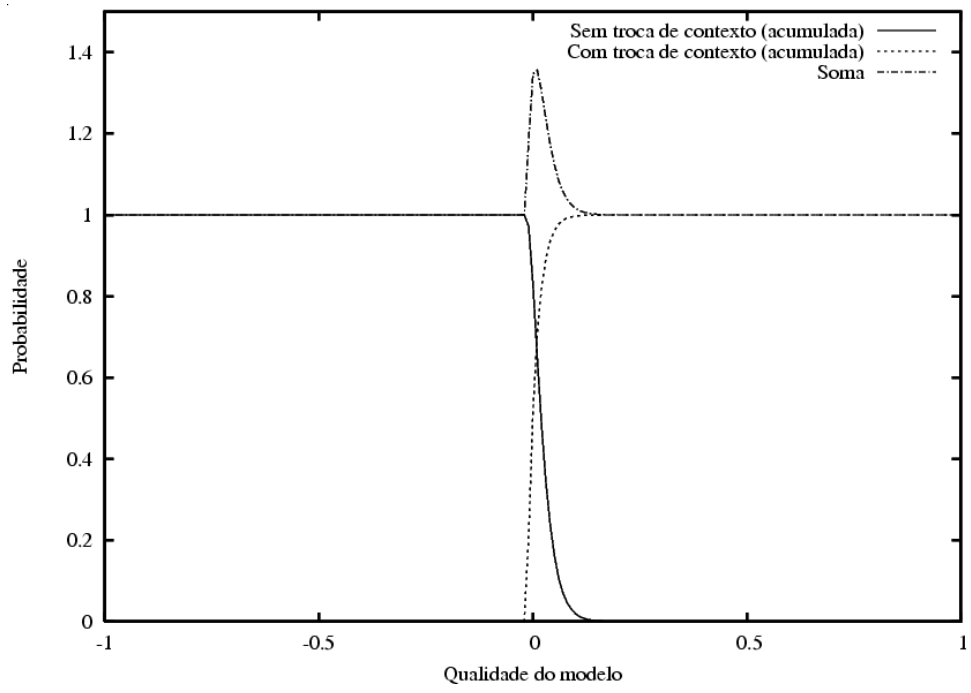

Figura 4.12: Distribuições cumulativas das qualidades – com e sem trocas de contexto (100 estados).

ces de falsos positivos e falsos negativos, em várias situações pode ser aconselhável utilizar valores ainda menores para a qualidade mínima. Isso se explica pelo fato de a exigência de níveis de qualidade bastante baixos antes de reconhecer uma troca de contextos

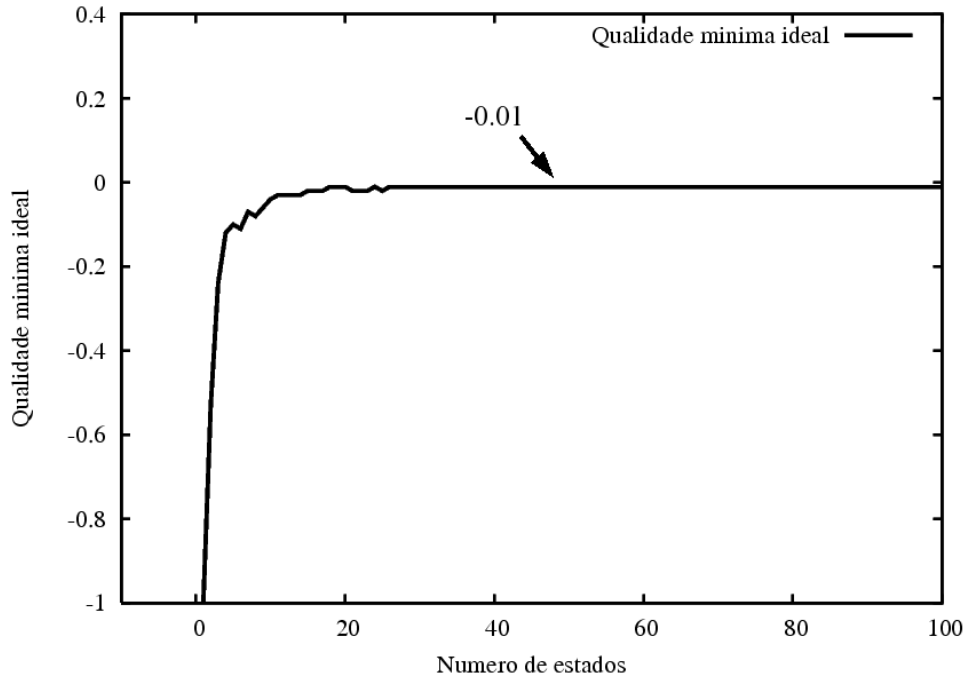

Figura 4.13: Qualidade mínima ideal.

implica um sistema menos propenso à criação desnecessária de modelos. A minimização simultânea de falsos positivos e falso negativos supõe que ambos sejam igualmente prejudiciais. Entretanto, durante as experimentações com o RL-CD, pudemos perceber que erros do tipo falso positivo causam mais danos ao sistema, uma vez que a presença de modelos parciais "inúteis" acaba diluindo a capacidade do sistema corretamente dividir os contextos em modelos estanques. Por essa razão, valores de E_{min} menores que o sugerido pela abordagem teórica acabam sendo justificados, muitas vezes, em aplicações práticas.

Outra razão prática para utilizarmos valores de E_{min} menores do que $E_{min}^{ideal} = -0.01$ é que tal valor foi calculado considerando distribuições de qualidade para os casos em que os próximos contextos são *quaisquer*. Entretanto, uma das suposições do RL-CD é justamente a de que as trocas de contexto são abruptas, ou seja, de que as probabilidades de próximo-estado no contexto resultante são bastante diferentes das probabilidades do contexto atual.

Se quiséssemos obter estimativas para as curvas de tais distribuições, precisaríamos implementar um método Monte Carlo semelhante ao apresentado no algoritmo 15. A diferença para o algoritmo 15 seria que, ao invés de amostrarmos uniformemente dentre todos possíveis próximos contextos, amostraríamos uniformemente dentre todos os próximos contextos que são suficientemente diferentes do atual. Uma maneira de medir este grau de diferença seria considerarmos a distribuição de probabilidades de próximo estado como um vetor V_{atual} do tipo $[p_{s1}, p_{s2}, \ldots, p_{sN}]$, onde $p_{si} = T(s, a, si)$, para um dado par estado-ação atual (s, a). Assim, consideraríamos que houve uma troca de contexto abrupta caso a distância euclidiana entre V_{atual} e o vetor de probabilidades do próximo contexto fosse superior a algum valor pré-determinado, Ω. Note que, caso não tenha ocorrido uma troca de contexto, então $\Omega = 0$; caso a troca de contexto envolva uma alternância entre regimes completamente determinísticos (isto é, nos quais apenas um estado

concentra toda a probabilidade de transição), então $\Omega = \sqrt{2}$. Um valor razoável para Ω poderia ser o ponto intermediário entre estes casos extremos, ou seja, $\Omega = \frac{\sqrt{2}}{2}$.

Executar a amostragem acima mencionada equivale ao problema de, dado um ponto A em um n-simplex, amostrar uniformemente dentre os pontos do simplex que se encontram a uma distância de A maior do que Ω. Não conhecemos nenhum algoritmo eficiente para implementar este tipo de amostragem. Além disso, para sistemas com grande número de estados, a abordagem intuitiva de rejeição de amostras, na qual se amostram repetidamente possíveis próximos contextos (de acordo com o algoritmo 13) até que se encontre algum que esteja a uma distância maior que Ω, não é eficiente.

Mesmo não sendo eficiente para o caso geral, a abordagem por rejeição de amostras ainda nos permite aproximar as curvas de distribuição de valores de qualidades para sistemas de até aproximadamente 10 estados. Examinando essas curvas e executando uma análise semelhante à apresentada anteriormente para determinação de E_{min}^{ideal}, constatamos que provavelmente $E_{min}^{ideal} < -0.01$. Infelizmente, uma vez que não encontramos um algoritmo eficiente para gerar tal tipo de amostragem de próximos contextos, também não pudemos determinar o comportamento assintótico das distribuições de valores de qualidade no caso de trocas abruptas de contexto. Por esta razão, não podemos fazer nenhum tipo de análise sobre o valor E_{min}^{ideal} esperado nestas situações.

5 CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho, formalizamos e analisamos um método de aprendizado de máquina especificamente criado para lidar com ambientes não-estacionários. O RL-CD, como foi chamado, se mostrou eficiente tanto em ambientes simples quanto em cenários altamente estocásticos e com inúmeras causas para a não-estacionariedade. Em comparações empíricas, o desempenho do RL-CD foi superior ao de métodos clássicos de RL, como Q-Learning e o Prioritized Sweeping, e também ao de métodos especialmente criados para lidar com ambientes não-estacionários, como o MMRL e Compositional Q-Learning.

De forma geral, o RL-CD se apresentou como uma abordagem bastante eficiente para lidar com o problema da não-estacionariedade, pois conseguiu, ao mesmo tempo, criar representações parciais para os vários regimes de funcionamento do ambiente e escolher, a cada momento, o modelo de previsão e atuação mais adequado. Os cenários utilizados na validação do RL-CD variaram desde simulações simples de perseguição ao alvo até cenários complexos e altamente estocásticos de controle de tráfego.

Além da formalização do RL-CD e da execução das comparações acima mencionadas, também foi apresentada, no capítulo 4, uma análise das curvas de distribuição de probabilidades da qualidade instantânea e_m^T sob diversas situações. Através de métodos Monte Carlo, pudemos estimar numericamente a probabilidade de obtermos determinados valores de qualidade instantânea dado que tenha ocorrido (ou não) uma troca de contexto. Este tipo de análise não apenas permitiu que o RL-CD fosse estudado sob o ponto de vista do teste estatístico de hipóteses, como também permitiu encontrarmos o melhor valor teórico para a qualidade mínima aceitável, E_{min}. Tal valor, chamado de E_{min}^{ideal}, corresponde àquele que minimiza a chance da criação desnecessária de modelos e ao mesmo tempo compromete o mínimo possível a capacidade de detecção de trocas de contexto legítimas.

Embora os experimentos implementados confirmem que o RL-CD funciona bastante bem *caso as suposições exigidas sejam obedecidas*, ainda são necessários estudos sobre sua aplicabilidade quando tais suposições forem relaxadas. Em especial, é preciso haver uma análise de impacto sobre o aprendizado quando na presença de ambientes em que os contextos compartilham um número significativo de vetores probabilidades de transição. Nesses casos, o RL-CD se torna ineficiente devido à desnecessária replicação de modelos completos do espaço de transições. Uma possível solução para lidar com contextos não completamente separáveis seria considerar a *localidade espacial* no cálculo da qualidade, ou seja, associar medidas de qualidades a subregiões específicas do espaço de estados. Entretanto, a formulação geral de um Processo de Decisão de Markov não especifica nenhuma maneira simples (ou mesmo possível e consistente) de se definir métricas de distância entre estados. Por essa razão, a idéia de localidade espacial em espaços de estado discretos é problemática.

Outro ponto a ser estudado no futuro é a relação exata entre o valor de memória M

e o grau de não-determinismo do ambiente. Em outras palavras, sabemos empiricamente que M deve ser ajustado para valores altos em ambientes estocásticos, a fim de fazer com que a confiança nas estimativas de qualidade crescam mais vagarosamente. Entretanto, conforme discutido na seção 3.4, sabemos que valores altos de M podem ser bastante danosos em cenários determinísticos, e por isso uma análise teórica se faz necessária. Especificamente, seria interessante obter uma relação analítica entre o quão estocástico é um ambiente e o valor de M. A relação entre o valor de M e a freqüência de trocas de contexto também pode ser importante, uma vez que valores altos de memória impedem que o agente ganhe confiança suficiente em contextos com baixos tempos de duração.

O parâmetro ρ, correspondente à taxa de ajuste do traço de qualidade E_m, também deve ser estudado com mais atenção. Especificamente, sabemos que ρ é capaz de suavizar a variação temporal de E_m, e que isso é interessante em cenários estocásticos. O fato de tanto ρ quanto M estarem associados à quão estocástico é o ambiente sugere a possível existência de uma relação numérica entre tais parâmetros. É possível, inclusive, que a obtenção de métricas capazes de mensurar o grau de não-determinismo em um MDP possibilitem a substituição de ambos os parâmetros por um único valor.

Outra possível melhoria no mecanismo atual do RL-CD consistiria na modelagem de transições entre modelos parciais. Isso permitiria com que lidássemos com casos em que a troca de contextos não é independente das ações do agente. Além disso, a troca de contexto poderia ser estimada *previamente*, e não apenas detectada *a posteriori* com base nos erros de previsão. Conforme discutido na seção 3.5, a inclusão de um sistema para criação de modelos parciais exclusivos para a função de previsão de recompensas, R_m, ou de previsão de transições, T_m, tornaria o RL-CD mais robusto também frente a cenários multi-tarefa, ou seja, cenários onde a função de recompensa é alterada mas onde a dinâmica de transições permanece fixa entre contextos. Outra vantagem dessa melhoria é que o RL-CD não mais dependeria do ajuste do parâmetro Ω.

Ademais, o estudo de técnicas de detecção de contexto através de outras abordagens matemáticas robustas, como Seleção Bayesiana de Modelos, pode ser interessante e promissora. A possibilidade de extensão do mecanismo básico do RL-CD para tratar cenários com espaço de estados e ações contínuos ainda é uma questão em aberto.

Por fim, é importante ressaltar novamente que, embora os resultados deste trabalho se mostrem bastante promissores, ainda há muitos pontos de pesquisa a serem desenvolvidos. Entretanto, independentemente do tipo de trabalho que seguirá esta dissertação, os resultados empíricos e as análises teóricas já desenvolvidas indicam que o RL-CD de fato constitui-se como um método bastante eficiente para lidar com uma importante subclasse de ambientes não-estacionários.

REFERÊNCIAS

BELLMAN, R. E. **Dynamic programming**. Princeton: Princeton University Press, 1957.

CASSANDRA, A.; LITTMAN, M.; ZHANG, N. Incremental Pruning: a simple, fast, exact method for partially observable markov. In: ANNUAL CONFERENCE ON UNCERTAINTY IN ARTIFICIAL INTELLIGENCE, UAI, 13., 1997, Providence, RI, USA. **Proceedings...** San Francisco: Morgan Kaufmann, 1997. p.54–61.

CHOI, S. P. M.; YEUNG, D.-Y.; ZHANG, N. L. An Environment Model for Nonstationary Reinforcement Learning. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, 12., 2000. **Proceedings...** [S.l.: s.n.], 2000. p.994–1000.

CHOI, S. P. M.; YEUNG, D.-Y.; ZHANG, N. L. Hidden-Mode Markov Decision Processes for Nonstationary Sequential Decision Making. In: SUN, R.; GILES, C. L. (Ed.). **Sequence Learning**: paradigms, algorithms, and applications. Berlin: Springer, 2001. p.264–287.

DOYA, K. Metalearning and neuromodulation. **Neural Networks**, Oxford, UK, v.15, n.4, p.495–506, 2002.

DOYA, K.; SAMEJIMA, K.; KATAGIRI, K.; KAWATO, M. Multiple model-based reinforcement learning. **Neural Computation**, Cambridge, MA, USA, v.14, n.6, p.1347–1369, 2002.

KAELBLING, L. P.; LITTMAN, M.; MOORE, A. Reinforcement Learning: a survey. **Journal of Artificial Intelligence Research**, [S.l.], v.4, p.237–285, 1996.

KINNEY, J. J. **Probability**: an introduction with statistical applications. [S.l.]: John Wiley and Sons, 1997.

LITTMAN, M. L.; DEAN, T. L.; KAELBLING, L. P. On the complexity of solving Markov decision problems. In: ANNUAL CONFERENCE ON UNCERTAINTY IN ARTIFICIAL INTELLIGENCE, UAI, 11., 1995, Montreal, Québec, Canada. **Proceedings...** [S.l.: s.n.], 1995. p.394–402.

MOORE, A. W.; ATKESON, C. G. Prioritized Sweeping: reinforcement learning with less data and less time. **Machine Learning**, [S.l.], v.13, p.103–130, 1993.

NARENDRA, K. S.; THATHACHAR, M. A. L. **Learning automata**: an introduction. Upper Saddle River, NJ, USA: Prentice-Hall, 1989.

OLIVEIRA, D.; BAZZAN, A. L. C.; SILVA, B. C.; BASSO, E. W.; NUNES, L.; ROSSETTI, R. J. F.; OLIVEIRA, E. C.; SILVA, R.; LAMB, L. C. Reinforcement learning based control of traffic lights in non-stationary environments: a case study in a microscopic simulator. In: EUROPEAN WORKSHOP ON MULTI-AGENT SYSTEMS, EUMAS, 4., 2006, Lisbon, Portugal. **Proceedings...** [S.l.: s.n.], 2006. p.31–42.

SILVA, B. C. d.; BASSO, E. W.; BAZZAN, A. L. C.; ENGEL, P. M. Dealing with Non-Stationary Environments using Context Detection. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, ICML, 23., 2006, Pittsburgh, USA. **Proceedings...** New York: ACM Press, 2006. p.217–224.

SILVA, B. C. d.; JUNGES, R.; OLIVEIRA, D.; BAZZAN, A. L. C. ITSUMO: an intelligent transportation system for urban mobility. In: INTERNATIONAL JOINT CONFERENCE ON AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS, AAMAS, 5., 2006, Hakodate, Japan. **Proceedings...** New York: ACM Press, 2006. p.1471–1472. Demonstration Track.

SINGH, S. P. Transfer of Learning by Composing Solutions of Elemental Sequential Tasks. **Machine Learning**, [S.l.], v.8, p.323–339, 1992.

SUTTON, R.; BARTO, A. **Reinforcement Learning**: an introduction. Cambridge, MA: MIT Press, 1998.

SUTTON, R. S. **Temporal credit assignment in reinforcement learning**. 1984. Tese (Doutorado em Ciência da Computação) — University of Massachusetts, Amherst.

SUTTON, R. S. Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 7., 1990. **Proceedings...** [S.l.: s.n.], 1990. p.216–224.

THORNDIKE, E. **Animal Intelligence**. Darien, CT: Hafner, 1911.

TSITSIKLIS, J. N. Asynchronous Stochastic Approximation and Q-Learning. **Machine Learning**, Hingham, MA, USA, v.16, n.3, p.185–202, 1994.

WATKINS, C. J. C. H.; DAYAN, P. Q-learning. **Machine Learning**, [S.l.], v.8, n.3, p.279–292, May 1992.