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

SAMUEL JUSTO WASKOW

Aprendizado por Reforço utilizando *Tile Coding* em Cenários Multiagente

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

Profa. Dr. Ana Lúcia Cetertich Bazzan
Orientador

Porto Alegre, junho de 2010

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Waskow, Samuel Justo

Aprendizado por Reforço utilizando *Tile Coding* em Cenários Multiagente / Samuel Justo Waskow. – Porto Alegre: PPGC da UFRGS, 2010.

89 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2010. Orientador: Ana Lúcia Cetertich Bazzan.

1. Inteligência artificial. 2. Sistemas multiagentes. 3. Aprendizado por reforço. 4. Aproximação de funções. I. Bazzan, Ana Lúcia Cetertich. II. Título.

“The machine does not isolate man from the great problems of nature but plunges him more deeply into them.”

— ANTOINE DE SAINT-EXUPÉRY

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

PD programação dinâmica, do inglês *dynamic programming*

GPI iteração de política generalizada, do inglês *Generalized Policy Iteration*

IA inteligência artificial

JAL *joint action learner*

IL *independent learner*

MARL aprendizado por reforço multiagente, do inglês *multiagent reinforcement learning*

SMA sistemas multiagentes, do inglês *multiagent systems*

MDP processos de decisão de Markov, do inglês *Markov decision processes*

MSE erro médio quadrático, do inglês *mean square error*

OPPORTUNE *opportunistic coordination learning*

RL aprendizado por reforço, do inglês *reinforcement learning*

TD diferença temporal, do inglês *temporal difference*

TJ teoria dos jogos

CTVU controle de tráfego veicular urbano

LISTA DE FIGURAS

LISTA DE ALGORITMOS

RESUMO

Atualmente pesquisadores de inteligência artificial buscam métodos para solucionar problemas de aprendizado por reforço que estão associados a uma grande quantidade de recursos computacionais. Em cenários multiagentes onde os espaços de estados e ações possuem alta dimensionalidade, as abordagens tradicionais de aprendizado por reforço são inadequadas. Como alternativa existem técnicas de generalização do espaço de estados que ampliam a capacidade de aprendizado através de abstrações. Desta maneira, o foco principal deste trabalho é utilizar as técnicas existentes de aprendizado por reforço com aproximação de funções através de *tile coding* para aplicação nos seguintes cenários: presa-predador, controle de tráfego veicular urbano e jogos de coordenação. Os resultados obtidos nos experimentos demonstram que a representação de estados por *tile coding* tem desempenho superior à representação tabular.

Palavras-chave: Inteligência artificial, sistemas multiagentes, aprendizado por reforço, aproximação de funções.

Reinforcement Learning using Tile Coding in Multiagent Scenarios

ABSTRACT

Nowadays, researchers are seeking methods to solve reinforcement learning (RL) problems in complex scenarios. RL is an efficient, widely used machine learning technique in single-agent problems. Regarding multiagent systems, in which the state space generally has high dimensionality, standard reinforcement learning approaches may not be adequate. As alternatives, it is possible to use techniques that generalize the state space to enhance the ability of the agents to learn through the use of abstraction. Thus, the focus of this work is to use an existing reinforcement learning technique, namely *tile coding*, that is a better form of state representation. This kind of method is key in scenarios where agents have a high number of states to explore. In the scenarios used to test and validate this approach, our experimental results indicate that the *tile coding* state representation outperforms the tabular one.

Keywords: artificial intelligence, multiagent systems, reinforcement learning, function approximation.

1 INTRODUÇÃO

Como agentes podem aprender o que fazer quando não existe nenhuma entidade dizendo qual ação deve ser executada em cada circunstância? O dinamismo de alguns cenários pode exigir a capacidade de aprendizado para lidar com fatores imprevisíveis existentes nestes cenários.

Nos cenários monoagente, problemas de aprendizado por reforço são popularmente resolvidos com a representação das interações de um agente com o ambiente através de pares estado-ação. Em um cenário multiagentes, além das interações dos agentes com o ambiente, existem as interações entre os próprios agentes. Desta maneira, uma forma de representação convencional destas interações pode gerar uma explosão combinatorial do número de pares estado-ação.

Em um contexto similar ao utilizado por (OLIVEIRA, 2009), ao longo deste texto serão mostrados cenários e situações onde há a interação entre agentes em um ambiente distribuído e na maioria dos casos uma solução centralizada não é possível, seja pela falta de um meio de comunicação entre os agentes ou por restrições de dimensionalidade.

A pesquisa em aprendizado por reforço multiagente, do inglês *multiagent reinforcement learning* (MARL), está concentrada em problemas envolvendo poucos agentes e, normalmente, assumindo ambientes plenamente cooperativos. Em (PANAIT; LUKE, 2005), são mostradas direções de pesquisa que ainda não foram devidamente exploradas:

- Grande número de agentes, de 10 a 1000 agentes ou mais;
- Times heterogêneos: a maior parte da literatura presume que os agentes são idênticos tanto no comportamento quanto nas suas habilidades;
- Agentes com percepções complexas (WOOLDRIDGE, 2002, p.35);
- Cenários dinâmicos.

Dentro destas direções, este trabalho está relacionado com a questão do grande número de agentes, agentes com percepções complexas e cenários dinâmicos. A principal questão abordada por este trabalho é como tornar o aprendizado multiagentes uma solução aplicável em sistemas complexos.

Outro ponto importante de ser ressaltado é que a pesquisa em MARL, normalmente, é realizada sob o ponto de vista de teoria dos jogos (TJ). A TJ é um ramo da matemática que estuda interações e estratégias entre indivíduos em um jogo. Mais sobre TJ será visto no terceiro cenário do Capítulo 3. A aplicação de MARL em jogos tem como objetivo encontrar a melhor estratégia quando um agente tem um ou mais adversários, e normalmente costuma buscar o equilíbrio. Um ponto de equilíbrio estável é um resultado de um

jogo em que nenhuma mudança nas estratégias pode melhorar os resultados. Ele é considerado estável porque, se um jogador escolhe unilateralmente uma estratégia diferente da atual receberá um retorno (ganho) menor ou no máximo igual ao retorno no estado de equilíbrio.

Peter Stone (STONE, 2007) reforça que a aprendizagem multiagentes é um meio-termo entre teoria de jogos e IA e há diversas obras na intersecção dessas duas áreas. Interações entre agentes podem ser caracterizadas como jogos, mas em vários casos, não é apenas que a convergência para um equilíbrio não é um objetivo, mas que a própria formulação como uma forma normal ou forma extensiva de jogo pode trazer pouco ou nenhum progresso no sentido de encontrar uma solução para o problema. A partir de uma perspectiva de IA, aprendizagem multiagentes deve ser considerada uma área mais ampla do que a teoria dos jogos pode endereçar.

Existem dois extremos a serem considerados no aprendizado multiagentes. Em um extremo, existem os *independent learner* (IL)s, (CLAUS; BOUTILIER, 1998), abordagens que consideram outros agentes como parte do ambiente. Na direção oposta, há os *joint action learner* (JAL)s que consideram todas as observações e as ações de cada agente no ambiente.

O tamanho do espaço de busca dos estados é um fator chave para se estabelecer os limites da utilização de algoritmos de aprendizado por reforço em problemas complexos. Abstração de estados, ou agregação de estados, tem sido amplamente estudada. Abstração pode ser considerada como o mapeamento da descrição original de um problema para uma representação mais compacta, permitindo ao agente distinguir a informação relevante das informações irrelevantes.

De uma perspectiva computacional, a abstração de estados é uma técnica para tornar possível a aplicação de algoritmos de aprendizagem em grandes problemas, porém normalmente é um método centralizado. (LI; WALSH; LITTMAN, 2006) apresenta uma revisão desses métodos, dividindo os diferentes enfoques de acordo com dois critérios: se o algoritmo precisa ter pleno conhecimento do processos de decisão de Markov, do inglês *Markov decision processes* (MDP) e da exatidão do método. No estudo de (LI; WALSH; LITTMAN, 2006) é mostrado que existe um meio-termo entre maximização da redução do espaço de estados e a minimização da perda de informações pela seleção de abstrações. Outra questão levantada é que o espaço de estados pode ficar quase tão grande quanto o espaço base em domínios complexos.

Devido a complexidade de cenários multiagentes, onde cada agente deve ser capaz de distinguir o efeito das próprias ações no ambiente, as técnicas comuns de aprendizado por reforço podem não ser adequadas. Assim, o uso de uma técnica de aproximação de funções é fundamental para que se possa abstrair aspectos relevantes do aprendizado e assim, em alguns casos, contornar o problema da maldição da dimensionalidade.

Desta forma, a principal contribuição deste trabalho é a aplicação de um método linear de representação do espaço de estados chamado *tile coding* (SUTTON, 1996) em cenários multiagentes que possuem alta dimensionalidade do espaço de estados. Neste contexto, o método *tile coding* foi aplicado nos seguintes cenários: presa-predador, controle semafórico e jogos de coordenação.

Os resultados obtidos nos experimentos demonstram que o uso de *tile coding* nos 3 cenários acima citados tem desempenho superior ao uso da abordagem tabular, tanto de maneira centralizada quanto descentralizada. De forma centralizada, o reuso de exemplos diminui o espaço de busca para atualização dos valores. De forma descentralizada, o *tile coding* permite a representação de espaços de estados contínuos de forma eficaz.

O restante do presente texto está dividido em quatro capítulos:

- No capitulo 2 são apresentados os principais conceitos e características de aprendizado por reforço, aproximação de funções e aprendizado multiagentes;
- No capítulo 3 encontra-se a descrição da principal contribuição deste trabalho. Neste capítulo é descrito como foi realizada a generalização do espaço de estados nos 3 cenários multiagentes aplicados: presa-predador, controle semafórico e jogos de coordenação;
- No capítulo 4 são apresentados os experimentos e os resultados obtidos;
- No capítulo 5 são sintetizadas as conclusões deste trabalho e apresentados alguns apontamentos para 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 sua probabilidade de repetição 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 frequentes. A partir desta observação, foi proposta a chamada Lei de Efeito que sugere que comportamentos bem-sucedidos, isto é, comportamentos que levem o animal a consequências satisfatórias (recompensas), tendem a se tornar mais frequentes. Da mesma forma, comportamentos que levem a situações não-satisfatórias ou desagradáveis (punições) tendem a ser extintos.

Embora atualmente se conheçam 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 IA em que o aprendizado de comportamentos ocorre sem tutores, isto é, sem a indicação de quais os comportamentos corretos a cada momento. Esta área é hoje conhecida hoje como aprendizado por reforço (aprendizado por reforço, do inglês *reinforcement learning* (RL)), 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. De forma resumida, o crítico fornece sinais de reforço ou de punição ao aprendiz com base no tipo de comportamento (ação) efetuado em cada situação. A consequência do RL 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 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 predição; 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 apresentar *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 IA, essa dificuldade é conhecida como o problema da atribuição de crédito. De certa forma, todas abordagens baseadas em reforço são tentativas de resolver este problema.

2.1 Elementos do aprendizado por reforço

Antes da 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 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, umas 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 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 de RL com modelos ambiente, e baseiam-se em técnicas de programação dinâmica. As abordagens puramente estatísticas, por outro lado, baseiam-se em métodos numéricos, como os métodos 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 ligações entre essas metodologias. Um exemplo disso, que será melhor 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.

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:

[1] Do inglês *delayed rewards*.

- 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), a função de valor pode ser do tipo $Q(s, a)$, para toda ação a;
- opcionalmente, um método para *aproximar as funções de* $V(s)$ *ou* $Q(s, a)$.

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 com estado único) 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 se tornem desejáveis ao se perceber que levam, 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 satisfatória a longo prazo. A estimativa desta função implica um constante embate entre recompensas imediatas e benefícios futuros.

2.1.1 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.

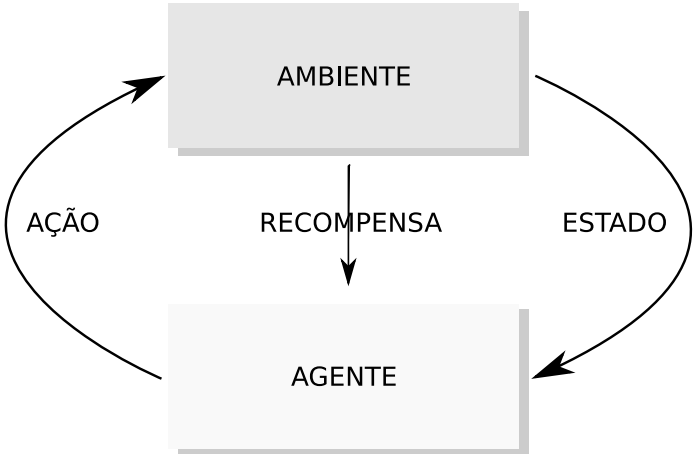


Figura 2.1: Arquitetura tradicional de um sistema de aprendizado por reforço

Formalmente, o agente e o ambiente interagem em uma série de passos de tempo discretos, $t = 0, 1, 2, 3, ...$ 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.

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, às vezes em função de variáveis ocultas às quais o agente não tem acesso.

2.1.2 Modelos de comportamento ótimo

Conforme mencionado 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 neste caso é $E(\sum_{t=0}^{h} r_t)$, onde r_t é a recompensa recebida no tempo t.

Esta recompensa futura esperada 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 $t = 0$, 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 $t = 1$, o agente age de acordo com uma política $(h-1)$-ótima, e assim por diante. A segunda maneira de interpretar a recompensa futura 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 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. Com isso, 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 1, 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. Este modelo é definido por $\lim_{h\to\infty} E(\frac{1}{h}\sum_{t=0}^{h} r_t)$.

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

2.1.2.1 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 escolher uma alavanca vencedora. Supondo que, até o tempo t, a alavanca a tenha sido puxada k vezes, este mecanismo pode ser descrito pela equação 2.1, onde $Q(a)$ é a expectativa de recompensa ao se escolher a.

$$Q_t(a) = \frac{r_1 + r_2 + ... + r_k}{k} \tag{2.1}$$

Pela Lei dos Grandes Números, sabemos que, quando $h \to \infty$, $Q_t(a)$ converge para $Q^*(a)$, ou seja, para a probabilidade correta de a fornecer a recompensa.

Percebe-se 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 conforme a equação 2.2. Esse tipo de atualização, na forma $Estimativa_{(t+1)} \leftarrow Estimativa_{(t)} + PassoAtualizacao(Alvo - Estimativa_{(t)})$ é comum em problemas de aprendizado por reforço.

$$\begin{aligned}
Q_{k+1} &= \frac{1}{k+1}\sum_{i=1}^{k+1} r_i \\
&= \frac{1}{k+1}\left(r_{k+1} + \sum_{i=1}^{k+1} r_i\right) \\
&= \frac{1}{k+1}(r_{k+1} + kQ_k) \\
&= \frac{1}{k+1}(r_{k+1} + kQ_k + Q_k - Q_k) \\
&= \frac{1}{k+1}(r_{k+1} + (k+1)Q_k - Q_k) \\
&= Q_k + \frac{1}{k+1}(r_{k+1} - Q_k)
\end{aligned} \tag{2.2}$$

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 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 são baseados no decaimento deste parâmetro.

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. Este método é baseado na distribuição de Boltzmann e está descrito na equação 2.3, onde n é o número de ações e τ é um parâmetro de temperatura. 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.

$$a = \frac{e^{\frac{E_t(a)}{\tau}}}{\sum_{i=1}^{n} e^{\frac{E_t(b)}{\tau}}} \tag{2.3}$$

2.1.2.2 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

[2] Do inglês *Learning Automata*.

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.

2.1.2.3 Métodos de perseguição

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 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:

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

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.

2.2 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 as $|A|$ 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 MDP.

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 MDP seguem a *hipótese de Markov*. 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 subsequentes 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, considerando $Pr(a|b)$ como a notação para probabilidade condicional da ocorrência de a dado o evento b, para todo estado e toda ação tem-se

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

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, um *modelo de Markov* consiste em:

- um conjunto discreto de estados, S;
- um conjunto discreto de ações, A;
- uma função de transição de estados $T : S \times A \to Pr$, onde $Pr(S)$ é uma distribuição de probabilidade sobre S ; sendo que $T(s, a, s')$ é a probabilidade de ocorrer a transição de s para s', dada a ação tomada;
- uma função de recompensa $R : S \times A \to \mathfrak{R}$.

A resolução de um MDP, no âmbito de RL, consiste em encontrar uma sequência 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.

2.2.1 Funções de valor

Praticamente todos os algoritmos de RL funcionam baseados em estimativas 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 optimalidade delineados na subseção 2.1.2, 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,

$$V^\pi(s) = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \Big| s_t = s \right\} \tag{2.4}$$

e

$$Q^\pi(s, a) = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \Big| s_t = s, a_t = a \right\} \tag{2.5}$$

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. Uma propriedade importante das funções apresentadas em 2.4 e 2.5 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 subsequentes, dado o modelo do ambiente (isto é, dadas as probabilidades de transição T e a distribuição de recompensas R):

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

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 RL, 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.

2.2.2 Funções ótimas de valor

Da mesma forma que definimos, acima, as funções de valor para uma dada política π, também podemos definir as *funções ótimas de valor*. 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:

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

e

$$Q^*(s,a) = R(s,a) + \sum_{s'} T(s,a,s')\gamma max_{a'} Q^*(s',a') \tag{2.7}$$

As equações 2.6 e 2.7 são conhecidas como Equações de Optimalidade 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 pela equação 2.8, simplesmente considerando-se a ação que maximiza a soma da recompensa imediata com a recompensa ótima esperada.

$$\pi^*(s) = arg\, max_a \left(R(s,a) + \gamma \sum_{s'} T(s,a,s')V^*(s') \right) \tag{2.8}$$

De forma semelhante, dada a função $Q^*(s,a)$ para todo s e a, podemos calcular a política ótima π^* através da equação 2.9. Note que essa equação permite o cálculo de π^* sem o conhecimento explícito do modelo de transições T, ou de recompensas, R.

$$\pi^*(s) = arg\, max_a Q(s,a) \tag{2.9}$$

É importante ressaltar que, ao menos teoricamente, pode-se também utilizar métodos padrão de resolução de sistemas de equações não-lineares para calcular os valores de V^*. Isso é possível pois as Equações de Optimalidade de Bellman (equações 2.6 e 2.7), na verdade, definem um *sistema não-linear* de N equações, para N estados.

2.3 Programação Dinâmica

programação dinâmica, do inglês *dynamic programming* (PD), 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 RL. 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, são apresentados 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.

2.3.1 Iteração de política

Dado que a solução dos sistemas de equações não-lineares induzidos pelas equações 2.6 ou 2.7 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 2.6 em uma regra de atualização:

$$V_{k+1}(s) = \sum_a \pi(s,a) \left(R(s,a) + \sum_{s'} T(s,a,s')\gamma V_k(s') \right) \tag{2.10}$$

para todo $s \in S$. Essa equação tem como ponto fixo $V_k = V^\pi$, uma vez que a própria Equação de Optimalidade de Bellman assegura 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 $V_2^\pi, V_3^\pi, ..., V_k^\pi$, até que os valores de $V(s)$ não se alterem mais do que alguma quantidade mínima estipulada (resíduo de Bellman). O algoritmo que implementa o primeiro requisito da Iteração de Política, ou seja, *a avaliação de uma política*, é apresentado no algoritmo 1.

Algoritmo 1: Avaliação iterativa de política

Entrada: a política a ser avaliada (π)
Inicialize $V(s) \leftarrow 0$ para todos os estados
repita
    $\Delta \leftarrow 0$
    para todo $s \in S$ **faça**
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(s,a) \left(R(s,a) + \sum_{s'} T(s,a,s')\gamma V_k(s')\right)$
        $\Delta \leftarrow max(\Delta, |v - V(s)|)$
até $\Delta < \theta$
Saída: $V \approx V^\pi$

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[3]. 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.

[3]Do inglês *full backups*.

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}. Esse 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:

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

O algoritmo completo para a iteração de política consiste em uma série de atualizações de V e de π conforme descrito em 2.11, 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}).

$$\pi_0 \xrightarrow{A} V^{\pi 0} \xrightarrow{M} \pi_1 \xrightarrow{A} V^{\pi 1} \xrightarrow{M} \ldots\ \pi^* \xrightarrow{A} V^* \tag{2.11}$$

O processo completo é apresentado no algoritmo 2. Note que as linhas 2 até 10 do algoritmo 2 equivalem ao passo A do ciclo descrito em 2.11, 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).

2.3.2 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 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:

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

Verificando esse resultado, percebe-se que ele equivale exatamente a transformar a Equação de Optimalidade de Bellman (eq. 2.6) 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.

Algoritmo 2: Iteração de política

para todo $s \in S$ **faça**
    inicialize $V(s)$ e $\pi(s)$ arbitrariamente
repita
    $\Delta \leftarrow 0$
    para todo $s \in S$ **faça**
        $\upsilon \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(s,a)\,(R(s,a) + \sum_{s'} T(s,a,s')\gamma V_k(s'))$
        $\Delta \leftarrow max(\Delta, |\upsilon - V(s)|)$
até $\Delta < \theta$
política-estável $\leftarrow$ verdadeiro
para todo $s \in S$ **faça**
    $b \leftarrow \pi(s)$
    $\pi(s) \leftarrow arg\,max_a R(s,a) + \sum_{s'} T(s,a,s')\gamma V_k(s')$
    se $b \neq \pi(s)$ **então**
        política-estável $\leftarrow$ falso
    se *não política-estável* **então**
        retorne ao passo 3
Saída: $\pi \approx \pi^*$

Algoritmo 3: Iteração de valor

para todo $s \in S$ **faça**
    inicialize $V(s)$ e $\pi(s)$ arbitrariamente
repita
    $\Delta \leftarrow 0$
    para todo $s \in S$ **faça**
        $\upsilon \leftarrow V(s)$
        $V(s) \leftarrow arg\,max_a R(s,a) + \sum_{s'} T(s,a,s')\gamma V_k(s')$
        $\Delta \leftarrow max(\Delta, |\upsilon - V(s)|)$
até $\Delta < \theta$
Saída: π determinística $\mid \pi(s) = arg\,max_a R(s,a) + \sum_{s'} T(s,a,s')\gamma V_k(s')$

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 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.

2.4 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, do inglês *temporal difference* (TD), 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.

2.4.1 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*.

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:

$$V(s_t) \leftarrow V(s_t) + \alpha\Big(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\Big) \tag{2.12}$$

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 (2.12), 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[4].

2.4.2 Q-Learning: controle *off-policy*

Uma das mais importantes formulações 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:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big(r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)\Big) \tag{2.13}$$

No *Q-Learning* o alvo da atualização TD 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 4.

Algoritmo 4: *Q-Learning*

```
Inicializa Q(s, a) arbitrariamente
para cada episódio faça
    Inicializa s
    repita para cada passo do episódio
        Escolhe a ação a para o estado s utilizando uma política derivada de Q
        Executa a ação a, observa o próximo estado s' e a recompensa r
        Q(s, a) ← Q(s, a) + α (r + γ max_a' Q(s', a') − Q(s, a))
        s ← s'
    até s é terminal
```

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 [5].

Uma vez 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^* (WATKINS; DAYAN, 1992). 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 é parcialmente 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 2.4.4.

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

[5] 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.

2.4.3 Métodos Ator-Crítico

Foi visto que a proposta inicial de aprendizado temporal, dada pela equação (2.12), 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 (2.8). 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 2.1.2.3.

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).

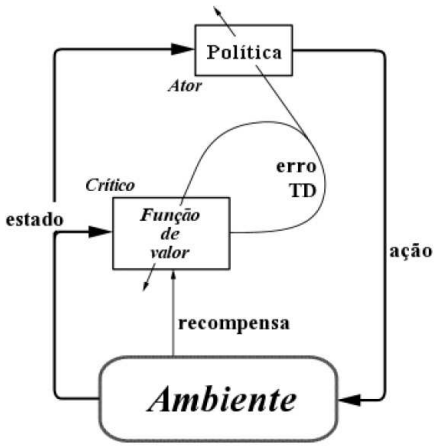


Figura 2.2: Arquitetura tradicional de um sistema Ator-Crítico adaptada de (SUTTON; BARTO, 1998)

O erro TD emitido pelo crítico pode ser tal qual como na equação 2.14 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.

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{2.14}$$

Suponha que a seleção de ações do ator é feita de acordo com parâmetros de preferência p, como nos Métodos de Perseguição (equação 2.15) 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: $p(s_t, a_t) \leftarrow p(s_t, a_t) + \beta\delta_t$.

$$\pi_t(s, a) = Pr(a_t = a | s_t = s) = \frac{e^{p(s,a)}}{\sum_b e^{p(s,b)}} \tag{2.15}$$

Outra formulação possível é atualizar p de forma inversamente proporcional à probabilidade de selecionar a: $p(s_t, a_t) \leftarrow p(s_t, a_t) + \beta\delta_t\Big(1 - \pi_t(s_t, a_t)\Big)$.

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.

2.4.4 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 (Figura 2.3a). Imagine, por exemplo, que um "estado objetivo" s_G (estado de alta recompensa) tenha sido atingido após uma trajetória de estados $s_1, s_2, ..., s_{n-1}, s_n, s_G$. 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_{s-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 (Figura 2.3b).

É visto que se os traços não sofrerem diminuição de intensidade após a última visita do estado, então cada passo de iteração corresponderia a atualizar igualmente todos os

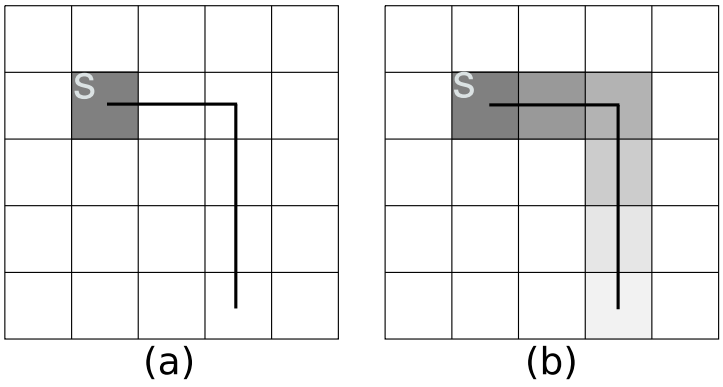


Figura 2.3: Exemplificação de Traços de Elegibilidade

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 deste 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).

2.4.5 Algoritmo Q(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.

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{se} \quad s \neq s_t \\ \gamma\lambda e_{t-1}(s) + 1 & \text{se} \quad s = s_t \end{cases} \tag{2.16}$$

O comportamento do traço e é decair lentamente sempre que um estado não for visitado, e aumentar em uma unidade a cada nova visita. Neste caso, somente é dada importância a estimativa de V, e não com tarefas de controle, já que, tendo apenas a função de valor de estado, seria necessário ainda de um modelo de transições e de recompensas para calcular a política ótima (equação 2.6).

O algoritmo 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 seguida pela seleção de ações, não podemos propagar o erro TD quando acontecem explorações, pois senão estaríamos corrigindo valores estado-ação em direção aos alvos errados. A versão padrão do Q(λ) é apresentada no algoritmo 5.

Algoritmo 5: Algoritmo Q(λ)

Inicializa $Q(s, a)$ arbitrariamente
$e(s, a) \leftarrow 0$
para cada *episódio* **faça**
    Inicializa s, a
    repita para cada passo do episódio
        Executa a ação a, observa o próximo estado s' e a recompensa r
        Escolhe a ação a' do estado s' utilizando uma política derivada de Q
        $a^* \leftarrow arg\, max_b\, Q(s', b)$
        $\delta \leftarrow r + \gamma Q(s', a^*) - Q(s, a)$
        $e(s, a) \leftarrow e(s, a) + 1$
        para todo s, a **faça**
            $Q(s, a) \leftarrow Q(s, a) + \alpha\delta e(s, a)$
            se $a' = a^*$ **então**
                $e(s, a) \leftarrow \gamma\lambda e(s, a)$
            senão
                $e(s, a) \leftarrow 0$
        $s \leftarrow s'$; $a \leftarrow a'$
    até s, a *ser terminal*

2.5 Generalização e Aproximação de Funções

Os métodos citados anteriormente assumem que as estimativa de funções de valor são representadas através de tabelas onde cada entrada destas tabelas representam um estado ou pares estado-ação. Esta representação tabular é indicada somente para tarefas que possuam poucos pares estado-ação. Este tipo de representação se torna rapidamente inviável em domínios com um grande número de estados e/ou ações. Uma possível solução para este problema consiste na utilização de formas alternativas à representação tabular, como generalização e aproximação de funções.

Neste contexto, como a esperiência de um subconjunto limitado do espaço de estados pode ser generalizado de maneira a produzir uma boa aproximação de um subconjunto maior? Em muitas tarefas em que são aplicadas técnicas de aprendizado por reforço, pode ocorrer o fato de que alguns estados não sejam visitados. Isto quase sempre acontece quando o espaço de estados ou ações possui variáveis contínuas ou percepções complexas. A única forma de aprender qualquer coisa em todas estas tarefas é generalizar estados previamente experimentados para um estado que ainda não foi visto.

Inicialmente, os métodos de aproximação de funções estimam a função de valor V^π através da experiência gerada usando uma política π da mesma forma que está descrito na seção 2.2.1. A novidade deste método é que a função de valor aproximada no tempo t, V_t, não é representada como uma tabela, mas como uma forma parametrizada com um vetor de parâmetros $\vec{\theta_t}$. Isto significa que a função de valor V_t depende totalmente de $\vec{\theta_t}$, variando de um passo a outro conforme $\vec{\theta_t}$ variar.

Por exemplo, V_t poderia ser uma função computada por uma rede neural onde $\vec{\theta_t}$ seriam os pesos de conexão. Por ajustar os pesos, há uma grande quantidade de funções V_t que podem ser implementadas por uma rede neural. Ou ainda, V_t poderia ser uma função computada por uma árvore de decisão, onde $\vec{\theta_t}$ são todos os parâmetros que definem os pontos de divisão e os valores das folhas da árvore.

Tipicamente, o número de parâmetros (número de componentes de $\vec{\theta_t}$) é muito menor do que o número de estados, alterando um parâmetro são alterados os valores estimados de muitos estados. Consequentemente, quando um único estado é recuperado, a mudança deste estado é generalizada afetando os valores de muitos outros estados.

Alguns métodos de controle como iteração de política generalizada, do inglês *Generalized Policy Iteration* (GPI), onde as funções de valor e de políticas interagem até chegar a uma solução ótima e assim, consistentes uma com a outra, é visto que frequentemente a busca por aprender Q^π enquanto π muda. Mesmo se a política permanece a mesma, os valores alvo de exemplos de treinamento são não-estacionários se estes forem gerados por métodos que iterativamente realizam *bootstrapping* [6]. Estes métodos que não são facilmente controlados como métodos não-estacionários são menos adequados para RL.

Em RL é importante que o aprendizado ocorra de maneira *on-line*, enquanto interagindo com o ambiente ou com um modelo de um ambiente. Para isso, são necessários alguns métodos de aprendizado eficiente a partir de dados adquiridos incrementalmente e que sejam capazes de controlar funções-alvo não-estacionárias (classificadores que mudam de acordo com o tempo).

Desta forma qual é a medida de desempenho adequada para avaliar os métodos de aproximação de funções? A maioria dos métodos de aprendizado supervisionado busca minimizar o erro médio quadrático, do inglês *mean square error* (MSE) sobre alguma distribuição, P, das entradas. Neste caso as entradas são estados e a função alvo (classificador) é a função de valor verdadeira V^π, assim, o MSE para uma aproximação V_t, usando o parâmetro $\vec{\theta_t}$ conforme a equação 2.17 onde P é a distribuição de pesos dos erros de estados diferentes.

$$MSE(\vec{\theta_t}) = \sum_{s \in S} P(s) \left[V^\pi(s) - V_t(s)\right]^2 \tag{2.17}$$

Esta distribuição é importante porque, usualmente, não é possível reduzir o erro para zero em todos os estados. Após isso, existem geralmente muito mais estados do que componentes em $\vec{\theta_t}$, tornando a flexibilidade de um aproximador de funções um recurso escarso. Uma melhor aproximação de alguns estados pode ser obtida, geralmente, somente através de uma aproximação inferior de outros estados. A distribuição P especifica como estas escolhas conflitantes devem ser feitas.

A distribuição P é usualmente também a distribuição em que os estados nos exemplos de treinamento são traçados, e assim, é realizada a distribuição dos estados em que são feitas substituições. Se deseja-se minimizar o erro sobre uma certa distribuição de estados, então deve-se treinar o aproximador de funções com os exemplos da mesma distribuição. Por exemplo, se deseja-se um nível uniforme do erro sobre todo o conjunto de estados, então deve-se realizar o treinamento com as recuperações distribuídas uniformemente sobre todo conjunto de estados, como em varreduras exaustivas de alguns métodos PD. Entretanto, é conveniente que a distribuição de estados em que são feitas as substituições e que a distribuição dos erros de pesos, P, seja a mesma.

2.5.1 Métodos Lineares

Um dos casos mais importantes de aproximação de função por gradiente descendente é onde a função aproximativa, V_t, é uma função linear do vetor de parâmetros $\vec{\theta_t}$. Correspondendo a cada estado s, existe um vetor coluna de características $\vec{\phi_s} =$

[6] *Bootstrapping* é a geração de estimativas a partir de outras estimativas.

$(\phi_s(1), \phi_s(2), ..., \phi_s(n))^T$ com o mesmo número de componentes de $\vec{\theta_t}$. As características dos estados podem ser construídas de diversas maneiras. Contudo, a função estado-valor aproximada de como as características são construídas é dada pela equação 2.18.

$$V_t(s) = \vec{\theta_t^T}\vec{\phi_s} = \sum_{i=1}^{n} \phi_t(i)\phi_s(i). \tag{2.18}$$

Neste caso a função de valor aproximativa é dita linear nos parâmetros, ou simplesmente linear. É natural usar atualizações gradiente descendente com aproximação de função linear. O gradiente da função de valor aproximativa em função de $\vec{\theta_t}$ neste caso é

$$\nabla_{\vec{\theta_t}} V_t(s_t) = \vec{\phi_{s_t}}. \tag{2.19}$$

Além disso, para o caso linear existe somente um ótimo $\vec{\theta^*}$. Assim, qualquer método com garantia de convergência para próximo de um ótimo local ou para próximo de um ótimo global proporciona uma análise matemática mais favorável. Quase todos resultados úteis de convergência para sistemas de aprendizado de todos os tipos são para métodos de aproximação de funções lineares.

Além destes resultados teóricos, os métodos de aprendizado linear são também interessantes, pois na prática estes são muito eficientes tanto na manipulação de dados como nos métodos de computação. Esta eficiência só depende criticamente de como os estados são representados em termos de características. A escolha de características apropriadas é um passo importante para fornecer conhecimento a priori para sistemas RL. Intuitivamente, as características devem corresponder às características naturais da tarefa, bem como aquelas características onde generalização é a mais adequada.

Por exemplo, se objetos geométricos fossem avaliados, as características mais adequadas a serem exploradas seriam: forma, cor, tamanho ou função. Se fossem avaliados os estados de um robô móvel, seriam avaliadas características como: posições, níveis de bateria restantes, leituras recentes de sona, etc. Em geral, são necessárias combinações destes atributos. Isto é porque a forma linear não possibilita a representação de interações entre características, como por exemplo, a presença de uma característica i está sendo boa somente na ausência de uma característica j.

2.6 Tile Coding

Tile Coding (SUTTON, 1996) é um método linear para aproximar funções de valores em aplicações práticas de aprendizado por reforço, cujos conjuntos de estados e ações são usualmente contínuos, ou seja, tendem a ser conjuntos muito grandes ou infinitos. Esta técnica generaliza o espaço de estados em partições denominadas *tiling*, e cada partição é composta por subpartições denominadas *tiles*. Cada *tile* é um campo receptivo para uma característica binária.

A primeira vantagem em utilizar *tile coding* é que o número global de atributos que estão presentes em um instante de tempo é estritamente controlado e independente do espaço de entrada. Exatamente uma característica está presente em cada *tilings*. Desta forma o número total de características presentes é sempre igual ao número de *tilings*.

O parâmetro de tamanho de passo aprendizado, α, pode ser configurado de uma forma intuitiva. Por exemplo, escolhendo $\alpha = \frac{1}{m}$, onde m é o número de *tilings*. Se o exemplo $s_t \mapsto v_t$ é recebido, para qualquer valor a priori, $V_t(s_t)$, o novo valor será $V_{t+1}(s_t) = v_t$. Usualmente deseja-se que as alterações ocorram de forma mais lenta que isso, justamente

para permitir generalização e variação estocástica nas saídas desejadas. Por exemplo, pode-se escolher $\alpha = \frac{1}{10m}$ para onde deseja-se direcionar a convergência movendo um décimo do caminho do alvo em cada atualização.

Por causa da soma exclusivamente binária $(1 - 0)$ de características que *tile coding* utiliza, a soma dos pesos que produz a aproximação da função de valor é trivial de computar. Ao invés de realizar n multiplicações e adições, um simples cálculo dos índices de $m << n$ características presentes e então, a adição de m componentes correspondentes no vetor de parâmetros. A computação de traços de eligibilidade também é simplificada, pois os componentes do gradiente, $\nabla_{\vec{\theta_t}} V_t(s_t)$, são usualmente 0 e 1.

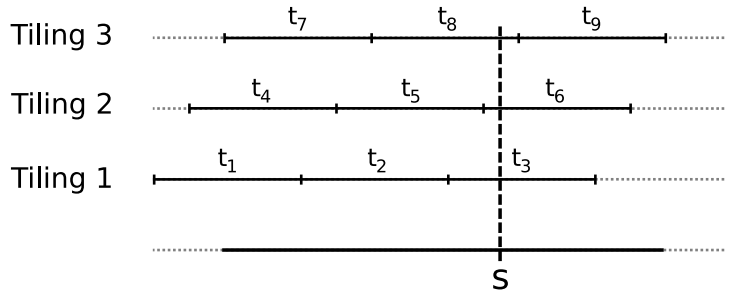


Figura 2.4: *Tilings* sobrepostos em uma dimensão

A combinação de aproximação linear com uma função de mapeamento $\phi_i(s)$ transforma o estado s em um vetor de N características binárias $[\phi_1, ..., \phi_N(s)]^T$. Cada *tile* recebe uma característica binária indicando se o estado atual de um agente está ou não sobre aquele *tile*. O estado atual de um agente pode ser composto por um ou mais *tiles* sobrepostos. Na Figura 2.4 é possível observar os *tiles* que incidem sobre um estado s. Assim, dado um estado s, o i-nésimo *tile* associado com o componente $\phi_i(s)$ de $\vec{\phi}$ é computado através da equação 2.20.

$$\phi_i(s) = \begin{cases} 0 & \text{se } s \notin tile_i, \\ 1 & \text{se } s \in tile_i. \end{cases} \tag{2.20}$$

Desta forma, o valor de $Q(s, a)$ é obtido através de $\phi(s) \times \vec{\theta}$, onde $\vec{\theta}$ é um vetor de N parâmetros associados com a ação a do estado s. Para aproximar a função que determina o valor de um estado utiliza-se sobreposição de *tilings* e, para cada *tiling* são computadas variáveis que indicam o peso de cada *tile*.

2.7 Aprendizado Multiagente

Há diversas definições para sistemas multiagentes, no entanto, neste texto é seguida a definição mais simples, encontrada em (SHOHAM; LEYTON-BROWN, 2009, p.xiii): "sistemas multiagentes, do inglês *multiagent systems* (SMA), são sistemas que incluem múltiplas entidades autônomas (agentes) com informações divergentes, com interesses divergentes ou ambos." Dentro deste contexto, um SMA cooperativo é formado por um conjunto de agentes autônomos interagindo de modo cooperativo com o objetivo de resolver alguma tarefa, ou um conjunto de tarefas, em conjunto. Por outro lado, em um SMA competitivo os agentes possuem objetivos independentes e buscam maximizar seus ganhos individuais. Mesmo havendo essa distinção clara entre os dois tipos de SMA, de acordo com (HOEN et al., 2006), muitas vezes é difícil definir se os agentes em um SMA são cooperativos ou competitivos, já que na prática os agentes podem ter comportamentos

que tanto podem ser classificados como cooperativos quanto competitivos. Por exemplo, um agente cooperativo pode apresentar um comportamento competitivo em relação a algum recurso do sistema e um agente competitivo pode apresentar um comportamento cooperativo em busca de um ganho maior através da cooperação.

Há grandes restrições na aplicação de técnicas tradicionais de aprendizado em SMA, principalmente porque normalmente os agentes não possuem uma visão global do sistema, restrições impedem que cada agente perceba o ambiente de forma completa. Técnicas tradicionais de aprendizado normalmente não podem ser aplicadas diretamente na maioria dos problemas de aprendizado multiagentes visto que o espaço de busca de políticas que maximizam a utilidade conjunta é muito grande para o aprendizado conjunto. Outro problema é que agentes independentes aprendendo de modo concorrente fazem com que o ambiente apresente um comportamento emergente imprevisível.

O aprendizado multiagentes cooperativo pode ser dividido em duas categorias: aprendizado de times e aprendizado concorrente. No aprendizado de times, um único aprendiz descobre qual deve ser o comportamento de cada um dos n agente, no caso concorrente vários agentes aprendem simultânea e independentemente comportamentos que maximizam a utilidade conjunta. A seguir serão introduzidos esses dois tipos de aprendizado.

2.7.1 Aprendizado de Times

Neste tipo de aprendizado, um agente "aprendiz" busca no espaço conjunto de ações e estados os comportamentos de todos os agentes do time. Este tipo de aprendizado pode utilizar técnicas tradicionais de aprendizado por reforço, já que um agente controla todos os demais, porém este deve conhecer os estados internos de todos os demais. Há problemas de escalabilidade devido ao espaço de ações conjuntas ser muito grande. Por exemplo, suponha um cenário com 2 agentes, onde cada possui 100 estados possíveis, o espaço total de busca do agente "aprendiz" seria 10.000 estados. Outro problema é que todos os dados precisam estar centralizados ou com acesso irrestrito, o que é inviável em cenários intrinsecamente distribuídos.

Existem três tipos de aprendizado de times: homogêneo, heterogêneo e híbrido. Nos times homogêneos, o mesmo comportamento é utilizado para controlar todos os agentes, porém não funciona caso a tarefa exija agentes especializados em sub-tarefas diferentes, por exemplo: ambiente de busca e resgate onde há agentes com tipos diferentes (bombeiros, policiais e ambulância) que devem cooperar entre si afim de resgatar o maior número possível de civis. No aprendizado de times heterogêneos cada agente pode apresentar um comportamento diferente, porém o número de possibilidades que têm que ser buscadas pelo agente aprendiz se torna mais complexo, sendo que se torna possível apenas para ambientes com poucos agentes. E por fim, no aprendizado de times híbridos, os agentes podem ser divididos em subgrupos de agente, onde cada subgrupo é homogêneo e com um aprendiz.

2.7.2 Aprendizado Concorrente

Neste modelo, há múltiplos agentes que aprendem simultaneamente, isso reduz o espaço conjunto, projetando-o em N espaços separados, onde N é o número de agentes. Este tipo de aprendizado é interessante quando o problema pode ser decomposto em subproblemas razoavelmente independentes. No entanto, ele se torna difícil para o agente porque o aprendizado e adaptação acontece em um ambiente onde há outros agentes também aprendendo, inclusive, em função do resultado do aprendizado dos demais agentes.

Os agentes com aprendizado concorrente podem ser classificados de acordo com in-

formação disponível e compartilhada, (MERKE; RIEDMILLER, 2001):

- Agente Caixa Preta: O agente só toma conhecimento dos outros agentes a partir da dinâmica do ambiente;
- Agente Caixa Branca: O agente possui todo o conhecimento dos demais agentes, e das ações tomadas por eles (o mesmo que JAL (CLAUS; BOUTILIER, 1998));
- Agente Caixa Cinza: são agentes do tipo "caixa preta" mas que podem (ou não) comunicar suas ações ou intenções de ações.

2.7.3 Aprendizado multiagentes através de Coordenação Opportunista (OPPORTUNE)

O método *opportunistic coordination learning* (OPPORTUNE) foi proposto por (OLIVEIRA, 2009) e tem como objetivo a utilização do aprendizado por reforço onde os agentes são livres para escolher de modo oportuno quando buscar mais informações sobre o ambiente, bem como quando agir de maneira conjunta com os demais agentes do ambiente.

O OPPORTUNE utiliza o algoritmo Q-Learning como base de seu método. É utilizado aprendizado concorrente, ou seja, os agentes aprendem de modo independente e podem ser classificados como "Caixa Cinza" (Seção 2.7.2). Chamamos esse método de aprendizado de oportunista já que o agente escolhe cooperar de acordo com o que lhe é mais oportuno no momento. Não há uma recompensa global a ser partilhada (mitigando o problema da atribuição de recompensas) e as cooperações visam melhorar as recompensas locais.

No OPPORTUNE, uma grande diferença em relação a estrutura de dados do Q-Learning tradicional está na representação da tabela Q, sendo utilizada uma representação esparsa dos estados. Porém a abordagem de Aprendizagem-Q esparsa e de contextos específicos considera a hipótese de que os agentes possuem *a priori* informações sobre a sua coordenação e dependências em todo o ambiente. Os mecanismos de coordenação surgem a partir das interações entre os agentes, e não há uma predefinição de onde e quando os agentes devem agir de modo conjunto nem as situações onde informações sobre o ambiente devem ser compartilhadas.

Na tabela Q dos agentes OPPORTUNE, as entradas Q armazenam valores relativos aos estados e ações tanto conjuntos quando individuais. Além disso, os agentes não possuem um acesso direto as tabelas dos demais agentes e nem há uma tabela central. Toda a informação é trocada por meio de mensagens e o conteúdo das mensagens possui tamanho limitado. Para isso assume-se que os agentes possuem capacidade de comunicação com os agentes na sua área ou grupo.

A comunicação é utilizada para a cooperação e não é necessário definir um número fixo de possíveis parceiros para a troca de percepções, porém é necessário que haja uma área de comunicação mesmo que possa variar ao longo da simulação.

3 REPRESENTAÇÃO DO ESPAÇO DE ESTADOS EM CENÁRIOS MULTIAGENTES

O objetivo deste capítulo é descrever como foram aplicadas abstrações no cenários multiagentes utilizados neste trabalho. Os cenários são compostos por agentes que estão inseridos no mesmo ambiente e possuem aprendizado concorrente (Seção 2.7.2). Além de contemplar uma abstração do espaço de estados, o principal objetivo do uso de aproximação de funções é a capacidade de abrangência de estados formados a partir de interações entre agentes. Desta forma, neste capítulo serão descritos: a descrição geral da aplicação de abstrações, o mapeamento de pares estado-ação, descrição da aplicação nos cenários e, por fim, serão apresentadas as limitações no uso de *tile coding*.

3.1 Descrição Geral da Aplicação de Abstrações

O tipo de representação utilizado é constituído basicamente por abstrações do espaço de estados em função do número de ações de cada agente. Desta maneira, ao invés de ser utilizada uma função de valor $V(s)$ para obter o retorno esperado a partir de um estado s, é utilizada uma função $Q(s, a)$ para obter um retorno esperado quando for tomada uma ação a em determinado estado s. Por utilizar um método linear de aproximação de funções (*tile coding*) e seguindo a nomenclatura proposta por (SUTTON; BARTO, 1998), sempre que forem citadas as "características" que representam determinado par estado-ação, em *tile coding* estas "características" significam o mapeamento dos *tilings* que representam este par estado-ação.

A extensão para predição de valor a partir de métodos de predição de estados é relativamente simples de ser realizada. Para isso, a função de valor de ação $Q_t \approx Q^\pi$ é representada como uma forma parametrizada através de um vetor de parâmetros $\vec{\theta_t}$. Além disso, os exemplos de treinamento estão no formato $s_t, a_t \mapsto v_t$. Desta forma, a saída desejada v_t pode ser qualquer aproximação de $Q^\pi(s_t, a_t)$.

Exemplificando, para cada ação possível a que pertença a um estado atual s_t, é possível calcular $Q_t(s_t, a)$ e então encontrar a ação gulosa $a^*_t = argmax_a Q_t(s_t, a)$. A melhoria de política é realizada alterando-se a estimativa de política para a política gulosa (em métodos *off-policy*) ou para uma aproximação suave da política gulosa para a política ε-gulosa (em métodos *on-policy*). Sendo assim, as ações são escolhidas de acordo com a mesma política em métodos *on-policy*, ou uma política arbitrária em métodos *off-policy*.

O algoritmo *Watkins's Q(λ)* (WATKINS, 1989) utiliza uma método linear de aproximação de funções através de características binárias e o método ε-guloso para seleção de ações. O cálculo da características presentes, ϕ_a, correspondem ao estado atual e todas as possíveis ações, a. Se a função de valor para cada ação é uma função linearmente separá-

vel, então os índices de ϕ_a para cada ação são essencialmente os mesmos, simplificando o processo computacional.

A utilização do método *Watkins's Q(λ)* para representação de estados e substituindo traços de elegibilidade conduz ao fato de que, com o uso de aproximação de funções, não existe somente um traço para cada estado, mas um traço para cada componente de $\vec{\theta_t}$, que pode corresponder a um ou muitos estados.

Algoritmo 6: Versão linear, gradiente descendente do algoritmo Watkins's Q(λ)

```
1  Inicializar θ⃗ arbitrariamente
2  para cada episódio faça
3      e⃗ = 0⃗
4      (s, a) ← estado e ação iniciais do episódio
5      φ_a ← conjunto de características presentes em (s, a)
6      repita para cada passo do episódio
7          para todo i ∈ φ_a faça
8              e(i) ← 1
9          Realizar ação a, observar recompensa r, e próximo estado s′
10         δ ← r − Σ_{i∈φ_a} θ(i)
11         para todo a ∈ A(s) faça
12             φ_a ← conjunto de características presentes em (s, a)
13             Q_a ← Σ_{i∈φ_a} θ(i)
14         δ ← δ + γ max_a Q_a
15         θ⃗ ← θ⃗ + αδe⃗
16         se probabilidade (1 − ε) satisfeita então
17             para todo a ∈ A(s) faça
18                 Q_a ← Σ_{i∈φ_a} θ(i)
19             a ← arg max_a Q_a
20             e⃗ ← γλe⃗
21         senão
22             a ← randomaction ∈ A(s)
23             e⃗ ← 0
24     até que s seja terminal
```

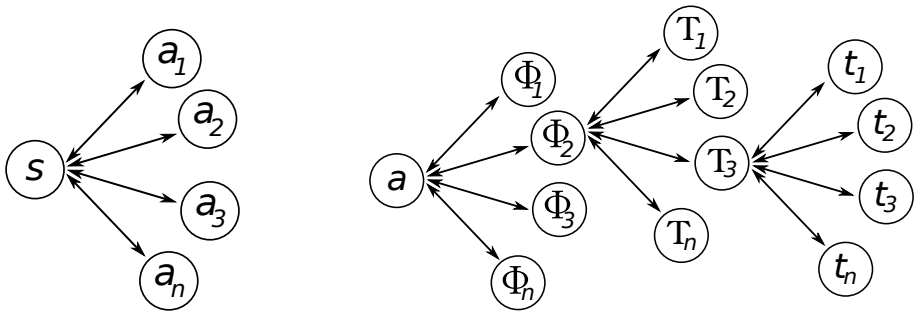
O algoritmo Watkins's $Q(\lambda)$ foi escolhido para aplicação de *tile coding*, pois o algoritmo permite a atualização dos valores $Q(s, a)$. Além disso, o algoritmo também permite que a atualização de diferença temporal (Seção 2.4) seja realizada utilizando traços de elegibilidade.

No Algoritmo 6, as linhas referentes às características presentes em cada par estado-ação (linhas 5, 10, 12, 13, 15 e 18) indicam os processos de busca e atualização das características $\vec{\phi}$ presentes em determinado valor $Q(s, a)$.

3.1.1 Mapeamento Estado-Ação

A representação do espaço de estado e ações através de características $\vec{\phi}$ possui um mapeamento de pares estado-ação diferente em relação ao tradicional mapeamento tabular. Quando é selecionada uma ação a em determinado estado s seguindo uma política π é comum relacionar cada estado s com um vetor de ações $\vec{a}$ que são tomadas a partir do

estado s. Na Figura 3.1a é possível observar como é o relacionamento de um estado s em função das ações pertencentes aquele estado.



(a) (b)

Figura 3.1: Mapeamentos de pares estado-ação (a): mapeamento de um estado s para muitas ações $\vec{a}$. (b): mapeamento de uma ação a para muitas características $\vec{\phi}$

Em *tile coding*, por utilizar abstrações do espaço de estados, o relacionamento tabular de pares estado-ação é substituído por um relacionamento em função de cada ação.

Na Figura 3.1b é mostrado como é construído este relacionamento. Para cada ação a, está relacionado um vetor de características $\vec{\phi}$ que representam as propriedades dos estados relativos à ação a. Para cada característica ϕ_i pertencente ao vetor de características $\vec{\phi}$ há um conjunto de *tilings* $\vec{T}$. Este conjunto de *tilings* $\vec{T}$ representa o conjunto de camadas de *tilings* T_i que são sobrepostos e, for fim, para cada *tiling* T_i, há um conjunto de tiles $\vec{t}$.

3.1.2 Representação do Espaço de Estados

Inicialmente, a abstração de estados proposta necessita de uma definição das características relacionais do espaço de estados e, por consequência, das ações tomadas sobre estes estados.

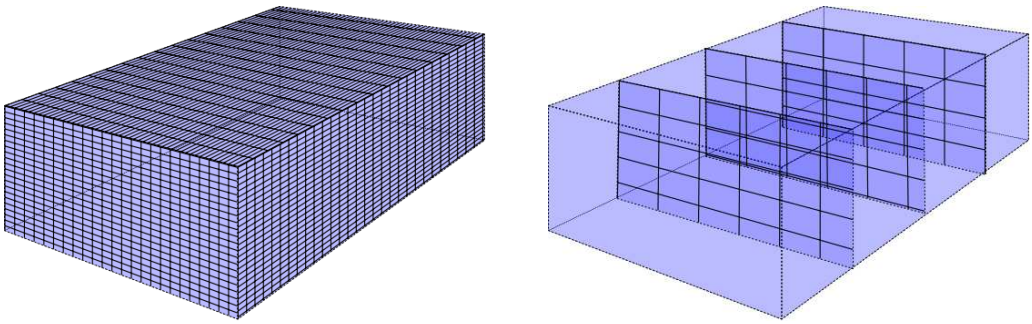
Para cada agente, conforme o mapeamento introduzido na seção anterior (3.1.1), cada relacionamento direto com outro agente possui uma posição ϕ_i no vetor de características $\vec{\phi}$. Por exemplo, dado um conjunto J de agentes, supondo que um agente j_1 interaja com outros dois agentes j_2 e j_3. Assim, j_1 irá possuir um vetor de características $\vec{\phi}$ de duas posições (ϕ_1 e ϕ_2). Este particionamento bidimensional do espaço de estados pode ser visto como a utilização de *tilings* retangulares. Para ilustrar este particionamento, a Figura 3.2a retrata um espaço de estados tabular e a Figura 3.2b retrata um espaço de estados com *tile coding* utilizando poucas partições.

Existem três parâmetros que regulam a distribuição dos *tiles* sobre o espaço de estados:

Número de *tilings* ($|\vec{T}|$) Este parâmetro regula o número de camadas que serão sobrepostas sobre o espaço de estados. Quanto maior é $|\vec{T}|$, mais ampla é a generalização. Quanto menor é $|\vec{T}|$, menos ampla e menos segmentada é a generalização;

Largura de cada *tile* (w) Este parâmetro regula o tamanho das regiões do espaço de estados que estão associadas com um único peso θ (Seção 2.6);

Resolução (r) Este parâmetro é a distância de sobreposição entre cada camada de *tiling*. A resolução r juntamente com o número de *tilings* $|\vec{T}|$ é o que efetivamente divide o espaço de estados em pequenas regiões.



(a) (b)

Figura 3.2: (a): representação tabular do espaço de estados. (b): representação por *tilings* retangulares do espaço de estados.

É necessário que se conheça *a priori* os limites superior e inferior do espaço de estados para que este seja particionado. O particionamento pode ser realizado conforme a necessidade de se obter granularidade sobre estados mais relevantes e ampliar a generalização para estados menos relevantes. Entretanto, para que as camadas de *tilings* fiquem sobrepostas de forma a cobrir uniformemente todo o espaço de estados, neste trabalho o parâmetro da resolução das camadas foi ajustado através da equação 3.1, onde r é a resolução da sobreposição de *tilings*, $\overline{|S|}$ é o limite superior do espaço de estados do espaço de estados, $\underline{|S|}$ é o limite inferior do espaço de estados, w é a largura dos *tiles* utilizados e $|\vec{T}|$ é o número de *tilings* utilizados.

$$r = \frac{(\overline{|S|} - \underline{|S|})}{w} \times \frac{1}{|\vec{T}|} \tag{3.1}$$

É valido ressaltar que o valor do parâmetro r pode ser ajustado independentemente para cada elemento de $\vec{\phi}$. Contudo, quando o número de características $|\vec{\phi}|$ é muito grande e não são conhecidas as partições que possuem informações mais relevantes ao processo de aprendizado, é importante que se cubra todo o espaço de estados. Desta forma, a equação 3.1 possibilita uma abrangência uniforme de todo o espaço de estados.

Nos experimentos realizados no Capítulo 4, os valores dos parâmetros w e r foram relacionados com o número de *tilings* $|\vec{T}|$ e *tiles* $|\vec{t}|$ utilizados. Quando maior é número de *tiles* $|\vec{t}|$, menor é a largura w destes *tiles*. Assim como quanto menor é o valor de $|\vec{t}|$, maior é o valor de w. O valor da resolução r foi calculado conforme a equação 3.1.

O processo de recuperação e atualização dos pesos das características é constituído de uma busca pelo conjunto de *tiles*. No Algoritmo Watkins's Q(λ) (Algoritmo 6), as linhas 5 e 10 fazem referência às características que estão presentes em determinado par estado-ação e as linhas 10, 13 e 18 referenciam a atribuição da soma dos pesos à função Q.

Um estado s pode ser representado por um conjunto de uma ou mais características ϕ_i. Para cada uma dessas características ϕ_i existe uma sobreposição de um ou mais *tilings* T_i. Desta forma, no processo de recuperação são procurados os tiles que estão nos intervalos de valores de cada característica ϕ_i que compõe um estado s.

Conforme a Figura 3.3, o valor aproximado de um determinado estado é obtido pela soma dos pesos θ dos *tiles* (situados em diferentes *tilings*) em que o estado está localizado, sendo que o número de *tilings* sobrepostos influencia diretamente na função que se deseja aproximar e no custo computacional.

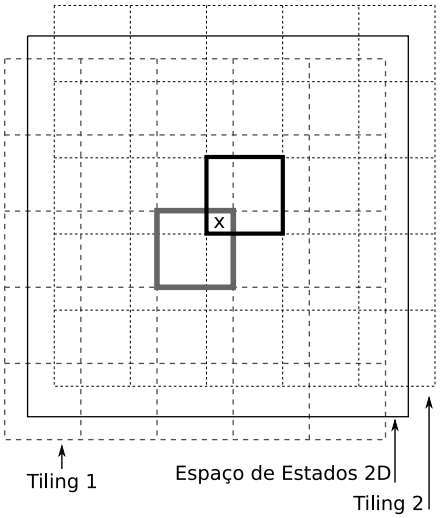


Figura 3.3: *Tilings* sobrepostos, adaptada de (SUTTON; BARTO, 1998)

3.1.3 Limitações no Uso de Abstrações

Uma das principais limitações no uso de abstrações é a necessidade de conhecimento *a priori* dos limites inferior e superior do espaço de estados. Durante o processo de aprendizado, caso o ambiente forneça um retorno que não esteja incluso no intervalo previamente definido, é necessário interromper o processo de aprendizado e realizar um ajuste dos limites do espaço de estados.

A ausência de um mecanismo de distribuição eficaz das camadas de *tilings* sobre o espaço de estados faz com que todos os estados sejam generalizados de maneira uniforme. Desta forma, estados mais relevantes não são representados por abstrações uma granularidade maior e estados menos relevantes não são representados por abstrações mais amplas.

Outra limitação recorrente ao *tile coding* é que apesar de possuir uma abordagem relacional no agrupamento das propriedades de formação de estados, não são incluídos grupos de *tilings* que contemplem a formação de estados conjuntos de um grupo de agentes de forma esparsa. No modelo atual, a influência das ações de um grupo de agentes sobre os retornos esperados individuais é absorvido pela própria capacidade de abstração. Este tipo de comportamento foi alvo de experimentos na Seção 3.2.1. Contudo, uma estrutura de *tilings* que aborde estados conjuntos formados de maneira esparsa em um grupo de agentes não é considerada por este trabalho.

Pretende-se corrigir ambas limitações em um trabalho futuro através de um método adaptativo de partição do espaço de estados e a inclusão de uma estrutura que seja direcionada para características conjuntas voltadas a um grupo grande de agentes. Contudo, isso será alvo de discussão na Seção 5.2.

3.2 Aplicação de *tile coding* em cenários multiagentes

Com a finalidade de validar a aplicação de *tile coding* neste trabalho foram implementados e realizados experimentos com alguns cenários que possuem espaço de estado e espaço de ação grandes.

O primeiro cenário consiste na captura de um agente presa por parte de dois agentes predadores. Para que o objetivo seja atingido (captura da presa), ambos agentes predadores devem aprender conjuntamente como realizar tal tarefa. O segundo cenário consiste

em controle de tráfego veicular urbano. O objetivo deste cenário é minimizar o número de veículos parados nas vias através da seleção de planos semafóricos em cada um dos semáforos existentes na grade. O terceiro cenário trata de jogos de coordenação, onde agentes devem se coordenar a fim de obter ganhos mútuos.

3.2.1 Jogo cooperativo: Presa e Predadores

Este cenário é composto por agentes "predadores" cujo objetivo é capturar um agente "presa" em uma grade infinita. Em cada unidade discreta de tempo, cada agente (predador ou presa) possui quatro ações possíveis: se movimentar para cima, para a direita, para baixo ou para a esquerda. Há diversas variações para este tipo de jogo em (DENZINGER; FUCHS, 1996). São elas:

- Formato do mapa (tamanho, limitações ou obstáculos);
- Tipos de agentes (movimentação, tamanho, velocidade, capacidades perceptivas e memória);
- Tipo de cooperação (número de predadores necessários para capturar a presa, etc);
- Objetivo dos caçadores (capturar ou "matar", onde matar é quando o predador ocupa a mesma célula da presa).

O cenário proposto por (TAN, 1993) consiste em um caso de tarefa conjunta envolvendo dois agentes predadores cujo objetivo é capturar um agente presa em uma grade toroidal 10×10. Nesta grade não há limitações (paredes) no deslocamento e percepção dos agentes. Não existem restrições quanto aos agentes ocuparem a mesma célula na grade. A cada novo episódio a posição dos agentes é disposta conforme mostrado na Figura 3.4.

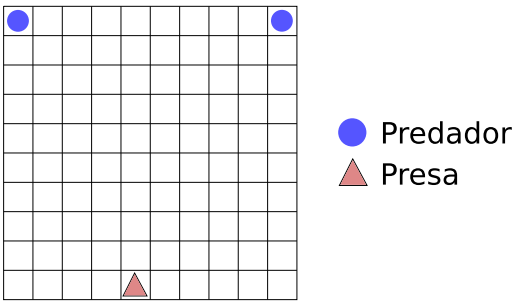


Figura 3.4: Ambiente grade 10 por 10 em seu estado inicial.

O estado terminal de cada episódio (captura da presa) é alcançado somente se ambos predadores estiverem em células adjacentes à presa (Figura 3.5b), sendo que a percepção destes agentes é conjunta. Os dois agentes tem suas ações modeladas conjuntamente, assim como a recompensa recebida é a recompensa total dos agentes. A cada passo, caso a presa não seja capturada, os predadores são penalizados com -1.0 pontos. A presa se desloca pelo ambiente escolhendo uma das 4 ações disponíveis aleatoriamente. Caso o estado objetivo seja alcançado (captura da presa), a recompensa recebida por ambos é 10.0 pontos.

Caso não haja nenhum agente dentro do campo perceptivo de um predador, o estado é arbitrado como estado-nulo conforme (JELLE R. KOK, 2004). Em contrapartida, caso

um ou mais agentes (presa ou predador) entrem no campo perceptivo de um dos agentes predadores, existe a necessidade do mapeamento de um estado correspondente à esta situação.

A percepção de um predador é composta por um número de células em torno da célula na qual o predador se encontra. Nos experimentos realizados o campo perceptivo atinge um raio de 2 células em torno do agente. Assim, conforme a Figura 3.5(a), cada agente percebe 25 células da grade. O estado perceptivo do agente predador é representado por coordenadas cartesianas, onde (x, y) é a posição da presa ou do outro predador em relação à posição do primeiro predador (localizado na posição $(0, 0)$).

Por exemplo, na Figura 3.5a está exemplificado o estado $(2, 2)$ representando que a presa se encontra na célula superior direita em relação ao estado do predador. No caso de ambos predadores estarem situados na mesma célula que a presa, as coordenadas relativas entre os 3 agentes são $(0, 0)$, pois todos se encontram na mesma célula. A modelagem deste cenário como um MDP é realizada para os agentes predadores j_1 e j_2, onde:

- S - Espaço de estados. É composto pela localização dos agentes predadores j_1 e j_2 e do agente presa p no ambiente. O vetor de características $\vec{\phi}$ tem cardinalidade 4. O elemento ϕ_1 representa a posição de p em relação a j_1; ϕ_2 representa a posição de j_2 em relação a j_1; ϕ_3 representa a posição de p em relação a j_2; ϕ_4 representa a posição de j_1 em relação a j_2.
- A - Espaço de ações. Combinação das ações (acima, abaixo, direita e esquerda) dos 2 predadores - $4^2 = 16$;
- R - Uma função de recompensa como $R : S \rightarrow \Re$. Se ambos predadores estiverem em uma posição de captura da presa (Figura 3.5b), então $R = 10$; senão $R = -1$;

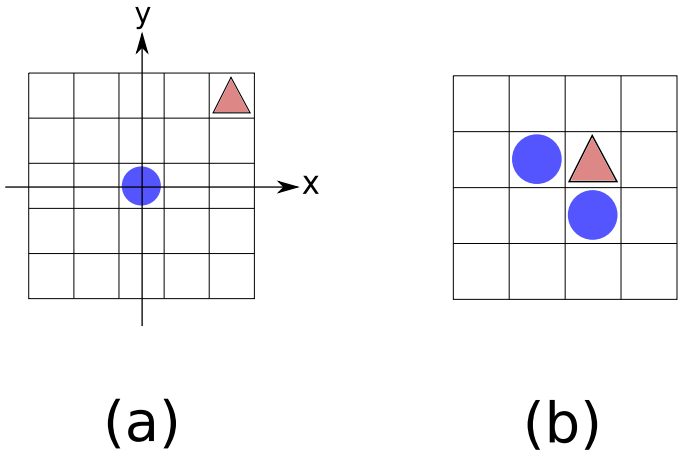


Figura 3.5: agentes: presa (triângulo) e predadores (círculos). (a): estado perceptivo representado por (2,2). (b): possível posição de captura da presa pelos predadores

No cenário anteriormente descrito percebe-se que o processo de coordenação entre os dois agentes predadores exige uma estrutura de dados de grande dimensionalidade. Para um único agente, a dimensão da tabela $Q(s, a)$ é de 26 estados $(25$ células $+ 1$ estado nulo$)$. Considerando que a posição da presa e a posição do outro predador devam ser considerados, a dimensão passa a ser de $676(26^2)$ estados por agente. Desta forma, a combinação de estados entre os dois predadores é totalizada em $456.976(26^4)$ estados. Como

cada agente pode realizar 4 ações, a combinação de ações conjuntas dos dois agentes predadores é 4^2. Ambos os predadores geram uma tabela $Q(s, a)$ conjunta de dimensão $7.311.616$ $(26^4 \times 4^2)$.

Na prática, os requisitos formais para a convergência da aprendizagem Q neste cenário ocorrem de maneira lenta, pois o processo de busca por todo espaço de estados e ações ocorre lentamente. Mesmo conseguindo a representação de todos os estados e ações possíveis, o emprego de *Q-Learning* seria inviável em aplicações deste cenário em tempo real, tendo em vista a visitação frequente de todos os estados e a inexistência de reuso de exemplos aprendidos.

A generalização neste cenário foi modelada a partir da sobreposição de *tilings*. Para isso, os *tilings* das extremidades foram dispostos representando os limites inferior e superior do intervalo do espaço de estados. O limite inferior representa os estados em que ambos os predadores se encontram nas células mais distantes da célula em que presa se encontra. Conforme descrito na seção anterior, o estado que representa o início do limite inferior de *tilings* é o estado-nulo. O limite superior representa os estados em que ambos os predadores estão em células próximas da presa (posição de captura).

Para o mapeamento de estados, o vetor de características $\vec{\phi}$ tem $|\vec{\phi}| = 8$, onde cada posição deste vetor representa a abstração das coordenadas cartesianas relativas entre 2 agentes. Por exemplo, o elemento ϕ_1 representa a distância relativa entre um predador e a presa, o elemento ϕ_2 representa a distância relativa entre um predador e o outro predador e assim por diante. Desta forma, a representação retangular plana não representa eficazmente todas as ações possíveis. Neste caso foi utilizada a representação através de hiperplanos de *tiles*. A partir dos limites de sobreposição de *tilings*, o número de conjuntos de *tilings* utilizados foi definido de acordo com o número de ações possíveis dos agentes.

3.2.2 Controle Semafórico

As redes de tráfego urbano estão cada dia mais saturadas e os congestionamentos são um problema em praticamente todas as áreas urbanas densamente populosas. A expansão da malha viária com a criação de rotas alternativas não é mais uma solução possível na maioria das cidades. Sistemas que controlam o fluxo de veículos estão tendo que lidar com cenários cada vez maiores e mais complexos. O controle deve levar em conta diversos fatores, tais como: tempos de percurso, velocidades médias, segurança de motoristas, etc. Além disso, muitas vezes acidentes ou eventos climáticos alteram significativamente o comportamento do tráfego.

A partir da década de 60 do século XX, sistemas de controle de tráfego urbano vêm sendo desenvolvidos tendo como principais objetivos:

- maximizar a capacidade de fluxo das redes urbanas;
- aumentar a segurança dos usuários;
- diminuir os tempos de percurso;
- diminuir os impactos negativos no meio ambiente e no consumo de energia.

A maneira mais comum de controlar o tráfego é a instalação de semáforos em cruzamentos. A inserção de controle semafórico em um cruzamento pode resolver problemas de priorização, aumentando a segurança do cruzamento. Porém, em muitos casos, a instalação de um semáforo aumenta fluxo e os tempos de percurso. Para minimizar os efeitos

negativos da instalação de semáforos, é essencial uma boa configuração dos planos semafóricos.

Em redes de tráfego onde não há um padrão bem definido de fluxo, por exemplo picos da manhã e fim de tarde, um sistema de controle de tráfego não adaptativo pode não ser eficaz. Nas grandes cidades os sistemas de tráfego que consideram fluxos bem definidos não podem lidar com fluxos variáveis onde não há uma concentração somente na zona central da cidade, devido à dispersão dos centros de negócios ao longo da rede. Além disso, em algumas cidades, algumas vias consideradas secundárias tornaram-se também vias importantes devido à saturação das vias arteriais.

Na próxima seção serão apresentados conceitos que são essenciais para o entendimento sobre funcionamento dos semáforos. Uma discussão detalhada sobre controle de tráfego veicular urbano (CTVU) e sistemas inteligentes de controle de tráfego é apresentada em (BAZZAN; KLÜGL, 2007).

3.2.3 Conceitos Básicos

A seguir serão definidos alguns conceitos básicos que devem ser apresentados para a melhor compreensão do funcionamento dos semáforos:

Semáforo: pode ser classificado, (LEITE, 1980), como sendo toda a instalação de controle existente em um cruzamento, incluindo os sinais luminosos, os fios, os instrumentos de controle, etc;

Tempo de ciclo: é o número de segundos para uma sequência completa das indicações do sinal, sendo que pode ter um tempo variável ou fixo;

Estágio: movimentos de tráfego permitidos;

Tempo de fase (ou *split*): uma porção do tempo de ciclo (tempo relativo de verde) reservado para qualquer estágio, (PAPAGEORGIOU et al., 2003);

Plano semafórico: um plano é um conjunto único de sequência fases;

Outro conceito importante é o de *fluxo de saturação*. De acordo com (ROESS; PRASSAS; MCSHANE, 2004, p.473), o fluxo de saturação é o volume de tráfego (veículos/hora) que pode passar por um cruzamento sinalizado considerando que o sinal esteja sempre verde. Este valor depende de diversos fatores (DENATRAN, 1979, p.64), dentre os quais: largura da interseção, número de veículos que fazem conversão, declividade da via, estacionamento e a presença de veículos comerciais (ônibus e caminhões). A fórmula padrão deste cálculo é Equação 3.2. Onde F é o fluxo de saturação em unidades de veículos por hora de tempo de verde e L é a largura (em metros) da aproximação.

$$F = 525L \tag{3.2}$$

Analisando o cenário mostrado na Figura 3.6 é possível exemplificar algumas características do problema. A especificação dos estágios é o que determina as movimentações do tráfego em cada parte do ciclo.

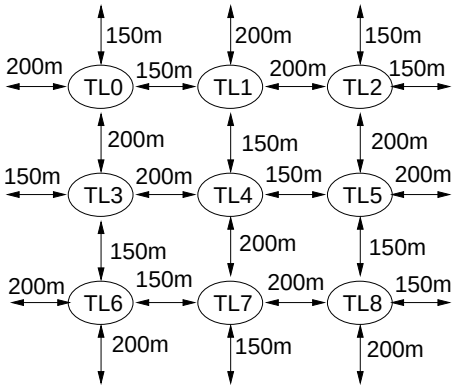


Figura 3.6: Cenário exemplo

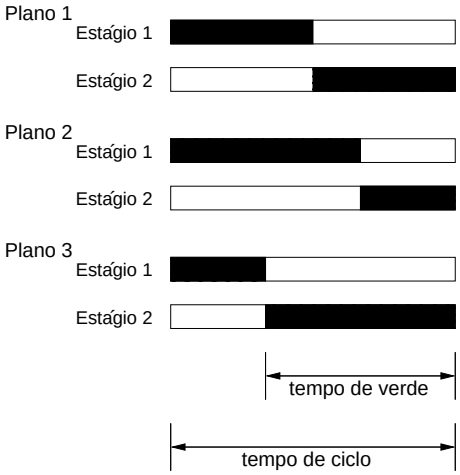


Figura 3.7: Especificação básica de planos semafóricos

A Figura 3.7 mostra três diferentes especificações de planos semafóricos: no “Plano 1” ambos os estágios têm uma porção igual do tempo de ciclo; no “Plano 2”, o estágio 1 tem o dobro do tempo de verde do estágio 2; e no “Plano 3” o estágio 2 tem o dobro do tempo do estágio 1.

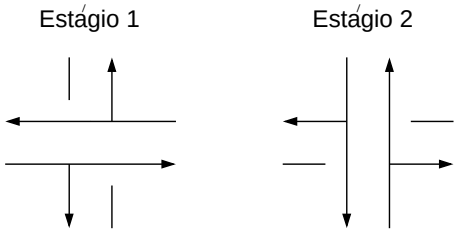


Figura 3.8: Exemplo de movimentos de dois estágios

A Figura 3.8 mostra um exemplo de especificação de dois estágios para um cruzamento. Cada um destes estágios deve ter uma duração relativa de verde, que é uma porção do tempo de ciclo do semáforo. Esta fatia de tempo é chamada de tempo de verde ou *split*.

Como exemplo, considere um veículo que entra no cenário visto na Figura 3.6 pelo Norte em direção ao Sul, que deve cruzar os seguintes semáforos: “TL1", “TL4"e “TL7", mantendo uma velocidade média de 10m/s. Tal veículo levaria 70 segundos ($\frac{700m}{10m/s}$) para

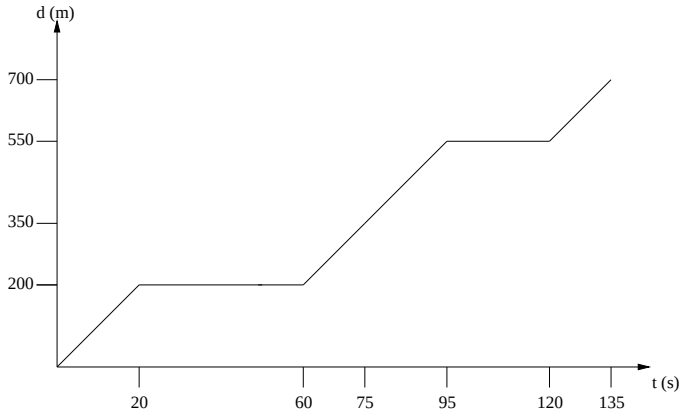


Figura 3.9: Distância x Tempo

cruzar todos os semáforos sem paradas e sair do cenário. Agora, considerando que todos os semáforos do cenário possuem a mesma especificação de estágios, vista na Figura 3.8, e todos estão utilizando o "Plano 3" com um ciclo de 60 segundos de duração. Nestas condições, o veículo levaria 20s para chegar ao primeiro semáforo "TL1" e finalizaria o percurso em 135 segundos. O gráfico da Figura 3.9 mostra este percurso. Este exemplo mostra que a configuração dos tempos dos semáforos pode gerar um atraso considerável para uma ou mais direções em uma via.

De acordo com Diakaki e colegas, (DIAKAKI; PAPAGEORGIOU; ABOUDOLAS, 2002), existem quatro possibilidades de influenciar as condições do tráfego utilizando semáforos:

- Especificação do estágio: para cruzamentos complexos, a especificação do numero ótimo de estágios pode ter impacto na eficiência do cruzamento;
- Tempo de fase: o tempo de verde de cada estágio deve ser dimensionado de acordo com a demanda das faixas envolvidas;
- Tempo de ciclo: tempos de ciclo maiores geralmente aumentam a capacidade do cruzamento, por outro lado, tempos de ciclo menores aumentam os tempos de espera em cruzamentos subsaturados;

3.2.4 Simulação Microscópica com o Simulador ITSUMO

Existem duas abordagens básicas de simulação de tráfego veicular: macroscópica e microscópica. A abordagem microscópica permite uma descrição detalhada de cada veículo da via. De modo geral, o ITSUMO (SILVA et al., 2006) é um simulador microscópico de tráfego que utiliza o modelo de movimentação de Nagel-Schreckenberg (NAGEL; SCHRECKENBERG, 1992), baseado em Autômatos Celulares (AC).

O modelo Nagel-Schreckenberg representa um modelo mínimo que reproduz características básicas do tráfego real. Nesse modelo, cada via é dividida em células de tamanho fixo, que podem conter um veículo. Cada veículo possui uma velocidade que é representada pelo número de células que ele pode percorrer por ciclo de simulação.

Na simulação, cada agente pode ser executado em um processo ou programa independente, onde é possível definir que um agente controla um semáforo ou um conjunto de semáforos da malha viária.

Abaixo, um resumo do protocolo de comunicação entre os agentes controladores e o simulador:

1. O simulador inicia a execução;
2. Os agentes controladores se conectam ao simulador através do envio de uma mensagem de conexão;
3. Após o ITSUMO receber a mensagem de conexão, ele retorna uma mensagem de resposta confirmando a conexão, composta pelos dados dos semáforos para os quais o agente solicitou controle;
4. Com os passos 2 e 3 a conexão é estabelecida;
5. A cada novo estado, o simulador envia uma mensagem a cada um dos agentes informando o estado atual da simulação no seu ponto controlado;
6. A cada intervalo pré-definido o simulador solicita uma ação de controle dos agentes;
7. Quando o agente recebe o pedido de ação, deve ser enviada uma mensagem com as mudanças (ou ações) desejadas.

3.2.5 Abordagens baseadas em IA

Em (BAZZAN, 2005) uma abordagem baseada em IA é descrita, onde cada semáforo é modelado como um agente. Cada agente possui planos pré-definidos para coordenação com agentes adjacentes. Planos diferentes podem ser escolhidos para haver coordenação em diferentes direções de acordo com a hora do dia.

Os principais benefícios dessa abordagem são que os agentes podem criar subgrupos de sincronização para melhor atender às necessidades do fluxo em alguma direção, não há um controle central e não há comunicação nem negociação direta entre os agentes. No entanto, são necessárias matrizes de recompensa (*payoff*) e essas matrizes devem ser explicitamente definidas pelo projetista do sistema. Isto faz com que a abordagem consuma tempo quando diferentes opções de coordenação são possíveis ou se há um grande número de vias e cruzamentos a serem controladas na rede.

Em (BAZZAN, 2009) é apresentada uma visão geral de controle de tráfego e direções de pesquisa nesta área, tais como: quando considerar o problema de tráfego como cooperativo; a dimensionalidade do problema associado ao aprendizado multiagentes com diversos agentes; coevolução e o papel dos motoristas em uma rede de tráfego veicular. Em (OLIVEIRA, 2009) é apresentado um método onde os agentes são capazes de decidir quando cooperar. Entretanto, são utilizadas heurísticas para realizar discretizações na modelagem de estados podendo omitir informações relevantes como por exemplo, o número de carros parados nas vias.

3.2.6 RL com Aproximação de Funções em Controle Semafórico

A aplicação de RL em um cenário de controle semafórico deve ser detalhada ao ponto de se observar todos os atributos existentes como por exemplo, a quantidade de veículos existente em cada uma das vias. Entretanto, como assegurar que cada agente seja capaz de representar as interações com o ambiente e com os outros agentes sem que haja explosão combinatorial do espaço de estados e ações? Este problema pode ser resolvido através da aplicação de *tile coding*.

O objetivo da aplicação de aprendizado neste cenário é a minimização de veículos parados, tendo em vista a maximização do fluxo nas redes de tráfego. Os estados são modelados conforme uma média de veículos parados em cada semáforo durante t passos.

Como a representação da média de veículos é contínua, isto geraria um número infinito de estados. Além disso, mesmo que estas representações de estados fossem previamente discretizadas, caso fosse utilizado um cenário com um número de semáforos muito grande seria ineficiente o uso deste tipo de heurística.

A modelagem de um problema de controle semafórico como um MDP é essencial para aplicação de mecanismos de RL. Esta modelagem é realizada para cada agente j, onde:

- S - Espaço de estados contínuo. Porque o semáforo impõe atrasos no fluxo de veículos, não faz sentido associar o número de veículos parados em cada passo de tempo. Assim, para cada semáforo, o estado é associado com a média de veículos parados em cada via que chega no cruzamento em 1 ciclo do plano semafórico. A duração deste ciclo pode ser, por exemplo, 60 passos de tempo;
- A - Espaço de ações discretas. As ações são definidas de acordo com planos semafóricos (Figura 3.7). Cada agente de controle seleciona um plano a ser executado a cada 120 passos de tempo;
- R - Uma função de recompensa como $R : S \rightarrow \Re$. Se a média de veículos parados tem mais do que 95% da capacidade da via, então $R = -1$; se a média de veículos parados tem entre 30% e 95% da capacidade da via, então $R = -0.01$; se a média de veículos parados tem menos do que 30% da capacidade da via, então $R = 1$;

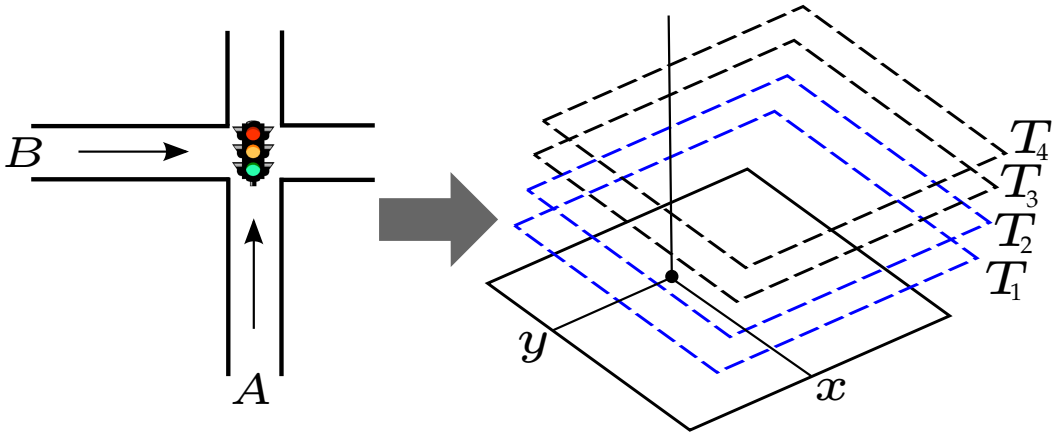


Figura 3.10: Mapeamento das características das vias de um cruzamento

Nos ambientes simulados (grade 3×3 e 9×9), em cada cruzamento existe um semáforo que controla o fluxo de veículos. Cada agente controla somente um semáforo no ambiente. Desta forma, as ações de cada agente são mapeadas como os planos semafóricos pertencentes à cada semáforo. Para cada ação, estão mapeados um conjunto de características $\vec{\phi}$.

A Figura 3.10 retrata como os estados das vias vertical e horizontal são representadas através de *tile coding*. A cada passo de tempo t são recebidas as informações do cenário através do simulador ITSUMO. Dentre as informações recebidas está o número de veículos parados em cada via. Desta maneira, em cada tile os valores de x e y são formados a partir da média de veículos parados nas vias vertical e horizontal, respectivamente.

Por exemplo, no caso da Figura 3.10, o valor do ponto x, y incide somente sobre os *tilings* T_1 e T_2. Desta forma, o valor aproximado de $Q(s, a)$ é obtido através da soma dos pesos θ de T_1 e T_2.

3.2.7 Jogos de Coordenação

Os jogos de coordenação são uma formalização de um problema de coordenação que é muito difundido na área de ciências sociais, onde existem situações em que todos os participantes podem obter ganhos, mas apenas tomando decisões coordenadas. Esta formalização é realizada através de TJ, na qual são estudadas situações estratégicas onde jogadores escolhem diferentes ações na tentativa de melhorar seus ganhos.

Os jogos estudados pela TJ são objetos matemáticos bem definidos: onde um jogo consiste de jogadores, um conjunto de ações disponíveis para cada um destes jogadores, e uma definição de recompensa para cada jogador e para cada combinação de ações. Na literatura existem duas formas de representação de jogos: forma normal e forma extensiva. Neste trabalho, será abordada somente a forma normal.

O jogo na forma normal consiste de uma matriz a qual mostra os jogadores, as ações, e as recompensas, onde existem dois jogadores, sendo que um escolherá as linhas e o outro escolherá as colunas. Os valores de recompensa são registrados na matriz.

Como exemplo de um jogo de coordenação, na Tabela 3.1 está a matriz de recompensas de um jogo onde dois jogadores tentam decidir qual o sistema operacional a ser utilizado. Ambos irão obter um ganho maior se escolherem o mesmo sistema operacional, pois os jogadores podem trocar software e experiências sobre o sistema. Assumindo que o sistema operacional O_a é mais amigável do que o sistema operacional O_b, então a recompensa a ser fornecida pelo sistema operacional O_a é maior que a recompensa fornecida pelo sistema operacional O_b.

Tabela 3.1: Matriz de recompensa de um jogo de coordenação com estratégia dominante

		Estudante 2	
		O_a	O_b
Estudante 1	O_a	$2/2$	$0/0$
	O_b	$0/0$	$1/1$

Formalmente, tem-se um conjunto de agentes J, onde cada agente $j \in J$ possui um conjunto de ações individuais A_j. Cada ação individual é denotada por $a_j \in A_J$. Os agentes então realizam jogadas consecutivas nos quais estes selecionam ações individuais. As ações individuais realizadas por todos os agentes em determinada jogada formam uma ação conjunta $A = \{a_1, a_2, ..., a_n\}$, sendo $n = |J|$.

Os jogos de coordenação são uma classe de jogos com múltiplos equilíbrios de Nash. Um equilíbrio puro de Nash é caracterizado como uma ação conjunta ótima se nenhuma outra ação conjunta possui um valor de utilidade maior.

3.2.8 RL com Aproximação de Funções em Jogos de Coordenação

No cenário utilizado para a aplicação de *tile coding*, o principal objetivo é que cada agente escolha uma ação que maximize seu retorno esperado após a realização de sucessivas jogadas com agentes vizinhos. A modelagem deste cenário como um MDP é realizada para cada agente j, onde:

- S - Espaço de estados contínuo. Cada agente possui uma matriz de recompensa idêntica à Tabela 3.1, porém a matriz de recompensa é constituída por valores contínuos.
- A - Espaço de ações discretas. As ações são definidas de acordo com uma matriz de recompensa;

- R - Antes de iniciar uma simulação no cenário, para cada agente é atribuída uma matriz de recompensa. A cada passo de tempo, cada agente realiza jogadas com os 4 agentes vizinhos. Logo, o valor de R é definido em função da matriz de recompensa que cada agente possui ao realizar as 4 jogadas com agentes vizinhos.

Em uma grade de agentes, cada agente j joga com os quatro agentes adjacentes a j. A cada passo de tempo, cada agente escolhe uma ação seguindo uma política ε-gulosa (Seção 2.1.2.1) e joga esta mesma ação com os quatro agentes adjacentes. O ambiente fornece o sinal instantâneo de reforço de acordo com a matriz de recompensas de cada agente. Assim, cada agente aprende a ação que maximiza seu reforço esperado através das quatro jogadas.

Como os valores da matriz de recompensa de cada agente são contínuos, o emprego de *tile coding* é fundamental para lidar com a alta dimensionalidade do espaço de estados deste tipo de cenário.

Conforme o método de aplicação de aproximação de funções proposto neste trabalho, a representação do espaço de estados de cada agente é modelado da seguinte maneira: a cardinalidade do vetor de características $\vec{\phi}$ para cada agente é determinada em função do número jogadas realizadas entre cada agente e os seus vizinhos, ou seja, é reservado um elemento ϕ_i para cada jogada realizada entre dois agentes.

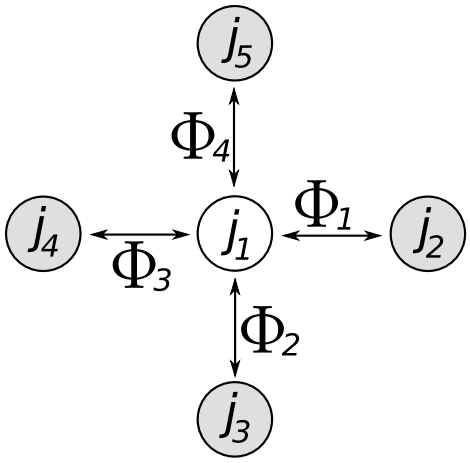


Figura 3.11: Modelagem de Representação do Espaço de Estados

Por exemplo, no cenário abordado, cada agente joga com quatro agentes vizinhos, logo cada agente possui um vetor de características $\vec{\phi}$ de cardinalidade 4. Assim, cada agente tem como armazenar uma aproximação dos valores obtidos nos jogos realizados com cada um dos outros agentes.

Isto é mostrado na Figura 3.11 onde cada círculo representa um agente j. Neste exemplo é analisado o mapeamento do espaço de estados do agente j_1 que está localizado no centro da figura. O vetor de características $\vec{\phi}$, representado por ϕ_1, ϕ_2, ϕ_3 e ϕ_4, possui uma posição para armazenar os valores relativos às jogadas com cada um dos agentes.

Cada posição do vetor de características ϕ_i possui camadas de *tilings* e cada uma destas camadas, por sua vez, são divididos em *tiles*. O número de *tilings* e *tiles* utilizados para o particionamento do espaço de estados varia e será discutido nos experimentos descritos na Seção 4.3.

4 EXPERIMENTOS E ANÁLISE DOS RESULTADOS

Neste capítulo serão apresentados os experimentos realizados nos cenários: presa-predador, controle semafórico e jogos de coordenação. Foram realizados experimentos com intuito de calibragem de parâmetros de generalização e experimentos relativos à comparação do desempenho de métodos de aprendizado.

4.1 Cenário Presa-Predador

Neste cenário os experimentos foram divididos em duas etapas. A primeira parte dos experimentos trata da comparação de diferentes parâmetros de generalização utilizados no método proposto neste trabalho. A segunda parte trata de uma comparação dos algoritmos $Q(\lambda)$ (Algoritmo 6, Seção 3.1) utilizando *tile coding*, algoritmo *Q-Learning* (Algoritmo 4, Seção 2.4.2) utilizando a representação tabular e o método OPPORTUNE (Seção 2.7.3) utilizando a representação tabular esparsa.

Tabela 4.1: Experimentos no cenário Presa-Predador

Experimento	Métodos	$\lvert\vec{T}\rvert$	$\lvert\vec{t}\rvert$	Política da Presa
I	*Tile Coding*	8,12,16	1	movimento aleatório
II	*Tile Coding*	1,10,20	8,10,12	fixa (parada)
III	*Tile Coding*	1,10,20	8,10,12	fixa (parada)
IV	*Tile Coding*	1,10,20	8,10,12	movimento aleatório
V	*Tile Coding*	1,10,20	8,10,12	movimento aleatório
VI	IL, *Tile Coding*, OPPORTUNE	10	10	fixa (parada)
VII	IL, *Tile Coding*, OPPORTUNE	10	10	movimento aleatório

Como mostra a Tabela 4.1 foram realizados sete experimentos neste cenário, sendo que os cinco primeiros experimentos visam comparar o número de parâmetros de generalização e os dois últimos experimentos são destinados à comparação de desempenho das representações tabular e *tile coding*.

4.1.1 Comparação de Parâmetros de Generalização

A comparação dos parâmetros de generalização foi realizada através de experimentos relativos ao número de *tilings* e *tiles* utilizados. Os parâmetros de aprendizado utilizados no algoritmo $Q(\lambda)$ foram fixados em $\alpha = 0.01$, $\gamma = 0.9$, $\lambda = 0.9$ e $\varepsilon = 0.1$. O valor do parâmetro α foi escolhido para ter um deslocamento pequeno em direção à função alvo. Os valores dos parâmetros γ e λ foram definidos de acordo com valores comumente utilizados na literatura. O valor do parâmetro ε foi escolhido para permitir uma pequena

característica exploratória e não percorrer somente os estados com as recompensas mais elevadas. Para cada experimento foram executadas 10 simulações e os valores obtidos são os valores médios destas 10 simulações.

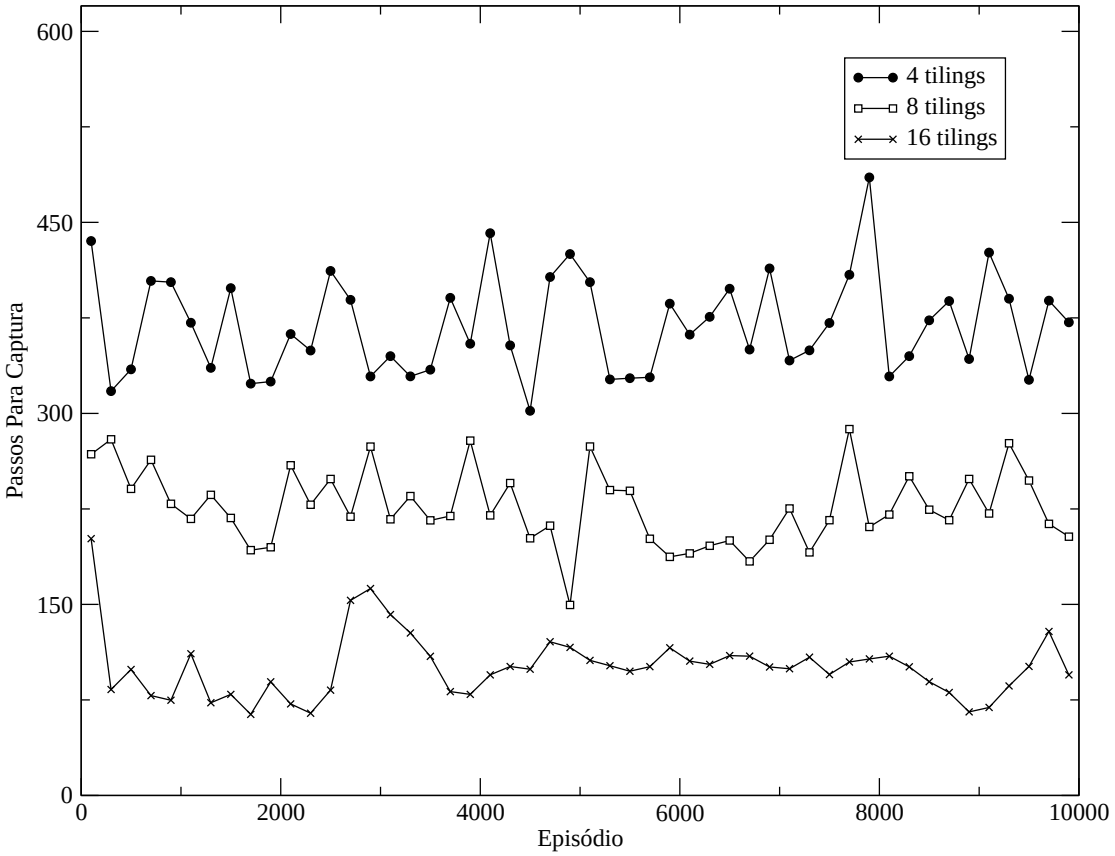


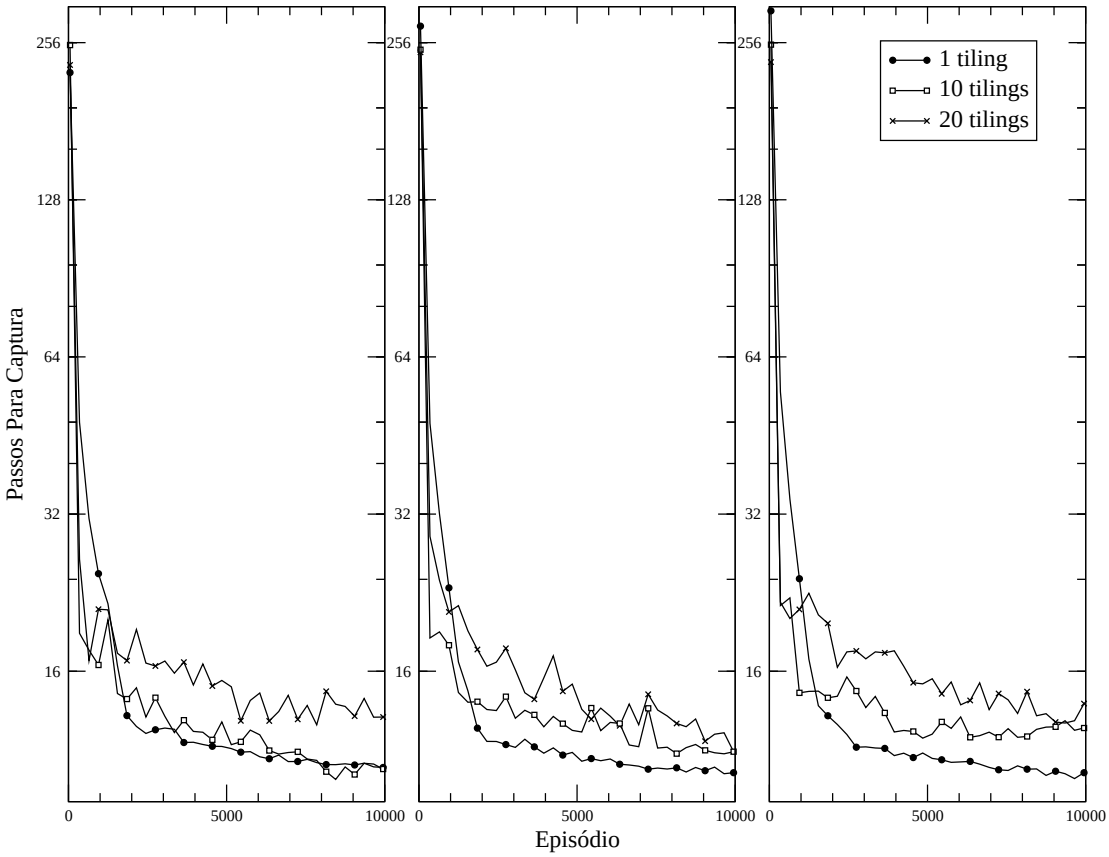
Figura 4.1: Experimento I - Comparação do número de *tilings* sem particionamento

O experimento I de comparação entre os parâmetros de generalização foi realizado com base nas alterações entre os números de *tilings*. Para a realização de tal comparação, foi especificado somente um *tile* por camada, pois o que se quer observar no experimento é o efeito da sobreposição das camadas sem a influência do número de partições (*tiles*).

Na Figura 4.1 é possível observar a comparação entre o número de *tilings* (4, 8, 16) e o número de passos por episódio. Este experimento foi realizado com a realização da captura da presa em movimento. De forma geral, quanto maior for o número de *tilings*, mais detalhada é a representação dos estados e, consequentemente, o processo de aprendizagem é agilizado em função do menor número de iterações para atingir o estado terminal. Entretanto, o excesso no número de *tilings* pode ocasionar um sobre-ajuste de características e fazer com que o processo de aprendizagem tenha um desempenho inferior. Este tipo de comportamento será demonstrado nos experimentos posteriores.

Para os parâmetros de generalização, a quantidade de *tilings* $|\vec{T}|$, foram escolhidos os valores 1, 10 e 20, pois neste cenário era desejada um reprodução dos efeitos de pouca generalização, generalização moderada e um excesso de características para representação de cada estado. Para os números de *tiles*, foram escolhidos os valores 8, 10 e 12, pois como a granularidade do espaço de estados foi ajustada principalmente em função do número de *tilings*, os valores 8, 10 e 12 de partições de cada camada $|\vec{t}|$ não exercem influência significativa.

O experimento II demonstra a utilização de diferentes números de *tilings* $|\vec{T}|$ em função de um número fixo de *tiles*. Desta forma, é demonstrada a variação no número de passos para a captura da presa de acordo com o número de *tilings* utilizados. Este experimento foi realizado utilizando a política da presa fixa e os números de *tilings* (1, 10 e 20). Na Figura 4.2a é mostrado o desempenho quando $|\vec{T}|$ estão particionadas em oito *tiles*. Na Figura 4.2b é mostrado o desempenho quando $|\vec{T}|$ estão particionadas em dez *tiles*. Na Figura 4.2c é mostrado o desempenho quando $|\vec{T}|$ estão particionadas em doze *tiles*.

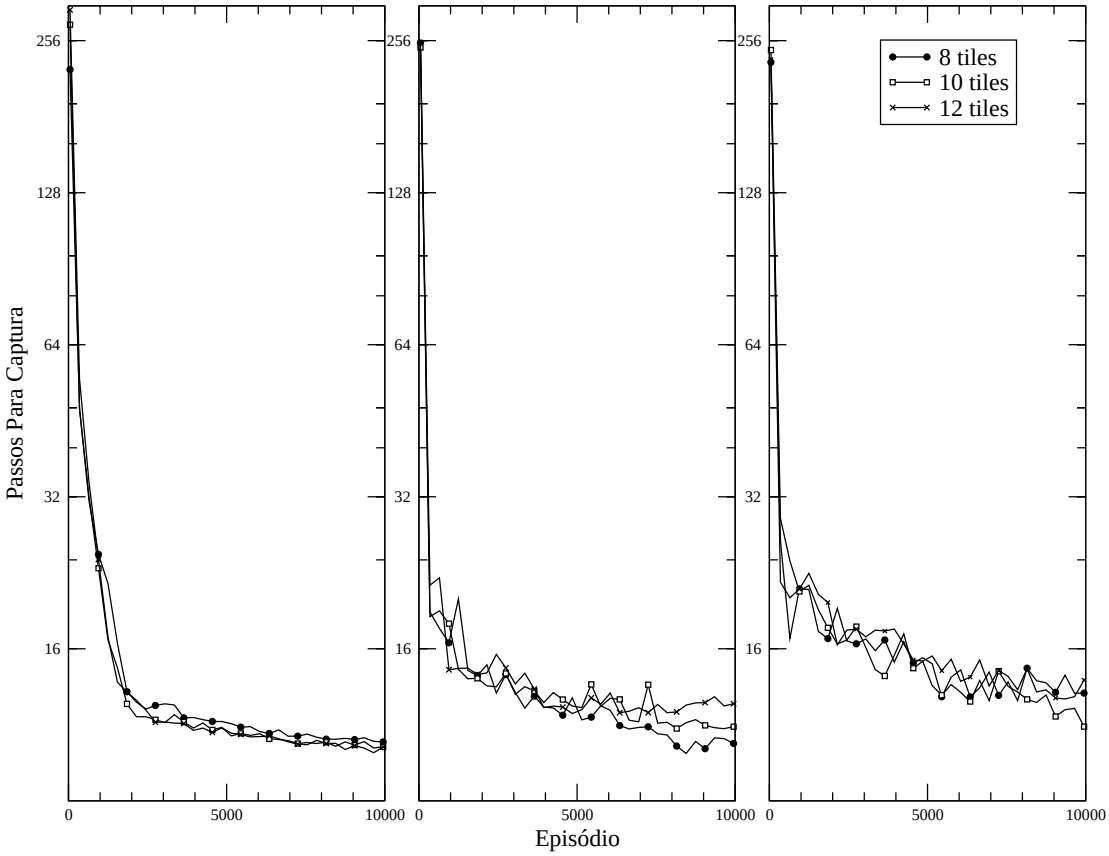


(a) *8 tiles* (b) *10 tiles* (c) *12 tiles*

Figura 4.2: Experimento II - Presa parada - Comparação do número de *tilings* particionados em 8, 10 e 12 *tiles*

É possível observar no experimento II que a generalização com somente 1 *tiling* obteve melhor desempenho neste cenário. Isso deve-se ao fato de que é possível, neste cenário, dizer que há como reduzir o espaço de estados substancialmente sem perda de desempenho. Ainda é possível verificar que conforme são adicionadas mais camadas de *tilings* (10 e 20), há a perda de desempenho. Esse fato caracteriza um sobre ajuste na representação de características, ou seja, o conjunto de características que representam cada estado é muito maior do que o necessário para realizar uma aproximação.

O experimento III utiliza os mesmos dados que o experimento II, porém este compara a utilização de diferentes números de *tiles* dentro de um mesmo número de *tilings*. Este experimento foi realizado utilizando a política da presa fixa e os números de *tiles* (8, 10 e 12). Na Figura 4.3a mostra o desempenho quando somente uma camada de *tilings* está particionada conforme diferentes $|\vec{t}|$. Na Figura 4.3b é mostrado o desempenho quando são utilizadas 10 camadas de *tilings* e com diferentes $|\vec{t}|$. Na Figura 4.3c é mostrado o



(a) *1 tilings* (b) *10 tilings* (c) *20 tilings*

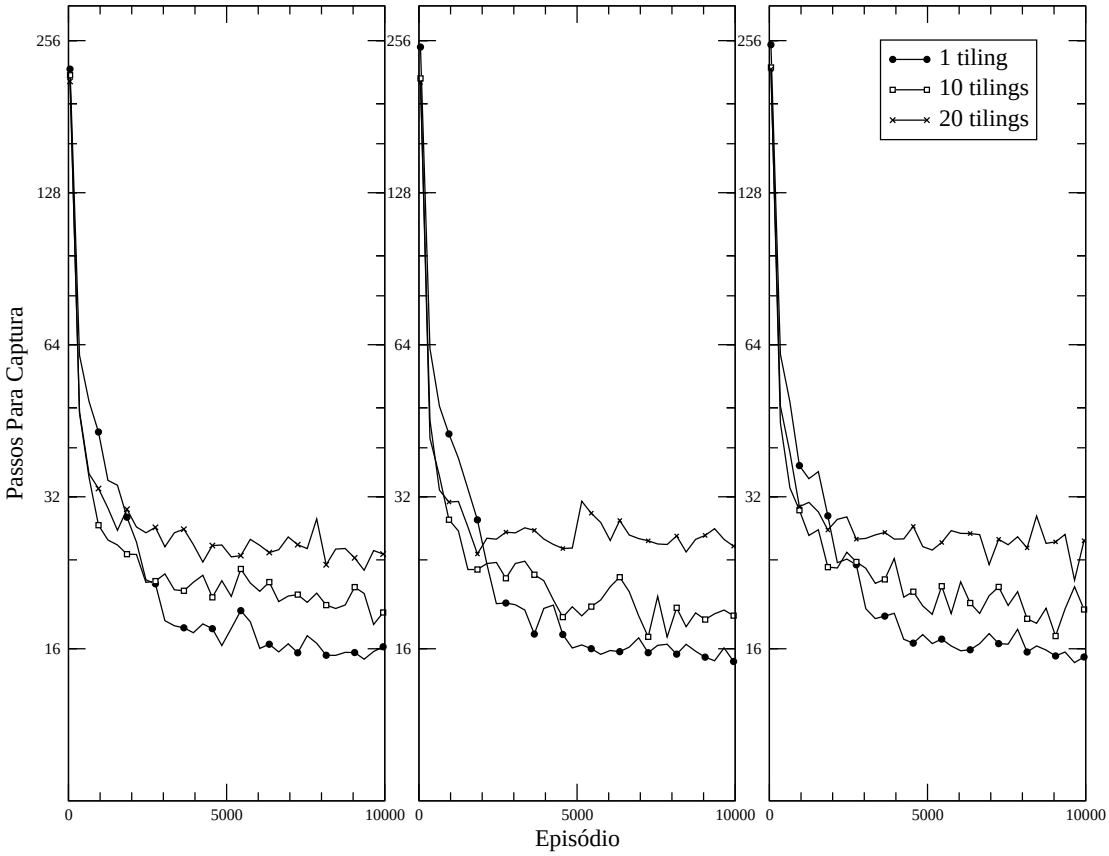
Figura 4.3: Experimento III - Presa parada - Comparação do número de *tiles* com 1, 10 e 20 *tilings*

desempenho quando são utilizadas 20 camadas de *tilings* e com diferentes $|\vec{t}|$.

Na Figura 4.3a é mostrado que, quando os parâmetros de generalização estão bem ajustados, há uma pequena variação utilizando a mesma quantidade de *tilings*. Com exceção de valores distintos como 1 *tile* (experimento I) ou um número muito grande de *tiles*, o número de *tiles* não exerce influência significativa no desempenho do processo de aprendizado. Na Figura 4.3b observa-se uma variação maior no número de tiles e próximo de 10000 episódios é verificado os efeitos de sobre-ajuste das características, onde as simulações com 8 *tiles* tiveram um desempenho melhor do que as simulações com 10 e 12 *tiles*. Desta maneira, na Figura 4.3c está caracterizado o sobre ajuste das características. Em todos os grupos de dados a aproximação realizada obteve um desempenho pior em comparação aos grupos mostrado nas Figuras 4.3a e 4.3b.

O experimento IV foi realizado com a presa se deslocando de forma aleatória e demonstra a utilização de diferentes números de *tilings* $|\vec{T}|$ em função de um número fixo de *tiles*. Desta forma, é demonstrada a variação no número de passos para a captura da presa de acordo com o número de *tilings* utilizados. Este experimento foi realizado utilizando a política da presa fixa e os números de *tilings* (1, 10 e 20). Na Figura 4.4a está demonstrado o desempenho quando $|\vec{T}|$ estão particionadas em oito *tiles*. Na Figura 4.4b está demonstrado o desempenho quando $|\vec{T}|$ estão particionadas em dez *tiles*. Na Figura 4.4c está demonstrado o desempenho quando $|\vec{T}|$ estão particionadas em doze *tiles*.

No experimento IV pode ser verificado nas Figuras 4.4a, 4.4b e 4.4c que a generali-



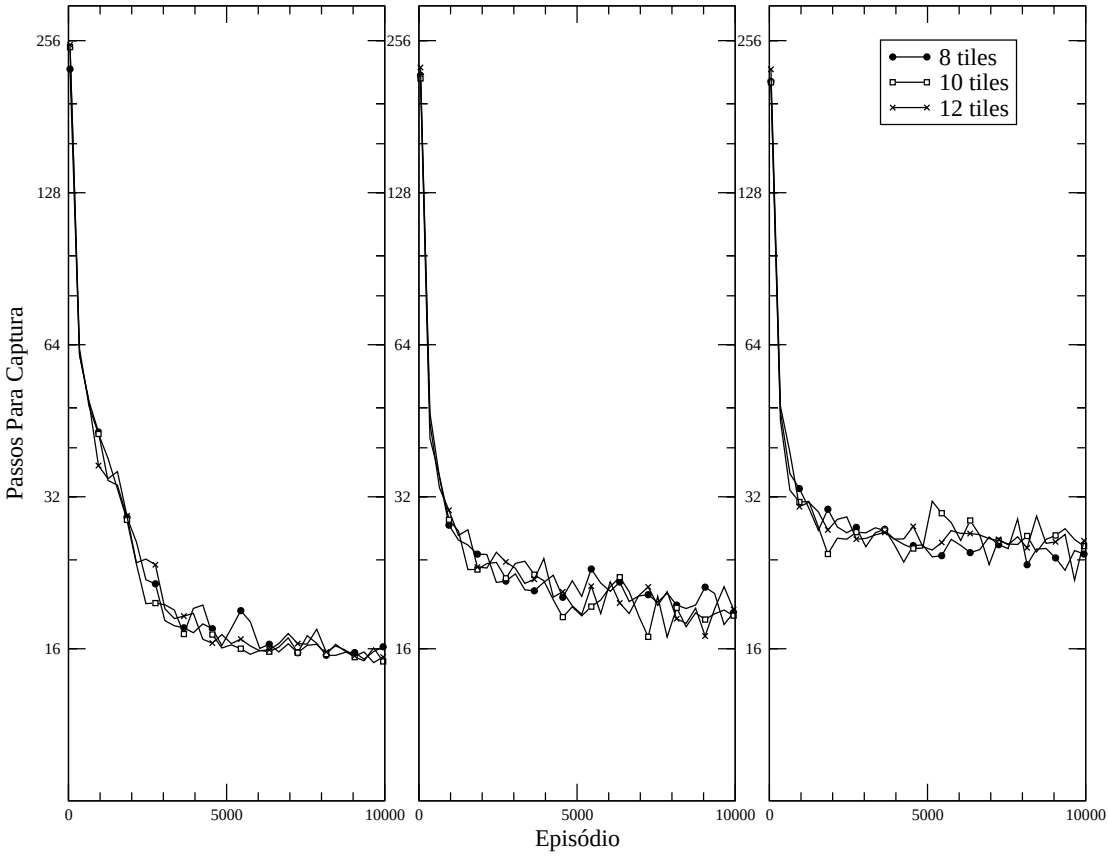
(a) *8 tiles* (b) *10 tiles* (c) *12 tiles*

Figura 4.4: Experimento IV - Presa em movimento - Comparação do número de *tilings* particionados em 8, 10 e 12 *tiles*

zação através de 1 *tiling*, inicialmente, possui um desempenho inferior às generalizações através de 10 e 20 *tilings*. Porém, no transcorrer dos episódios de simulação, o desempenho da generalização através de 1 *tiling* torna-se superior. Esse fato ocorre, pois não foram necessárias sobreposições das partições para a representação dos estados observados inicialmente.

Assim, as generalizações com maior número de *tilings* conseguem obter um desempenho inicial melhor. Entretanto, como o cenário possui um tamanho limitado em 10×10, o grande número de características faz com que o desempenho das generalizações com 10 e 20 *tilings* seja superado pela generalização de somente 1 *tiling* ao longo do experimento. Isso ocorre, pois somente a utilização de somente um *tiling* por ação possui um espaço de busca menor e no caso deste cenário, esta configuração é abrangente suficientemente para o escopo deste cenário.

O experimento V foi realizado com a presa se deslocando aleatoriamente e utiliza os mesmos dados que o experimento IV, porém este compara a utilização de diferentes números de *tiles* $|\vec{t}|$ dentro de um mesmo número de *tilings*. Este experimento foi realizado utilizando a política da presa fixa e os números de *tiles* (8, 10 e 12). Na Figura 4.5a é mostrado o desempenho quando somente uma camada de *tilings* está particionada conforme diferentes $|\vec{t}|$. Na Figura 4.5b é mostrado o desempenho quando são utilizadas dez camadas de *tilings* e com diferentes $|\vec{t}|$. Na Figura 4.5c é mostrado o desempenho quando são utilizadas vinte camadas de *tilings* e com diferentes $|\vec{t}|$.



(a) *1 tilings* (b) *10 tilings* (c) *20 tilings*

Figura 4.5: Experimento V - Presa em movimento - Comparação do número de *tiles* com 1, 10 e 20 *tilings*

No experimento V está demonstrado o mesmo comportamento observado no experimento III. Entretanto, como neste experimento a presa se desloca de maneira aleatória, de maneira geral é possível observar que em todas as configurações mostradas nas Figuras 4.5a, 4.5b e 4.5c, o desempenho de captura da presa é inferior ao experimento III.

Além disso, com a presa se deslocando de forma aleatória, a diferença do número de *tiles* testados não influencia do desempenho de captura. Com isso é possível concluir que os dois parâmetros de generalização: número de *tilings* e número de *tiles* exercem influência significativa na generalização. Entretanto, ao utilizar um número de maior de *tilings* tem-se um espaço de busca menor do que utilizar um número maior de *tiles*.

4.1.2 Comparação de Algoritmos Utilizando Diferentes Representações

No cenário Presa-Predador foram realizados experimentos para testes de desempenho. O experimento VI é um ambiente estático, onde a presa permanece na mesma posição durante toda a simulação e os predadores são recolocados na posição inicial a cada novo episódio (Figura 3.4). O experimento VII é um ambiente dinâmico onde a presa se desloca pelo cenário durante a simulação. Neste experimento a presa segue uma política aleatória de seleção de ações.

Na Figura 4.6 é mostrado um experimento realizado em (TAN, 1993). A situação de *mutual scouting*, onde ambos predadores compartilham percepções, é a mesma situação representada neste trabalho. Esta situação foi escolhida justamente pela alta dimensionalidade do espaço de estados e ações. É possível observar que o desempenho de *mutual*

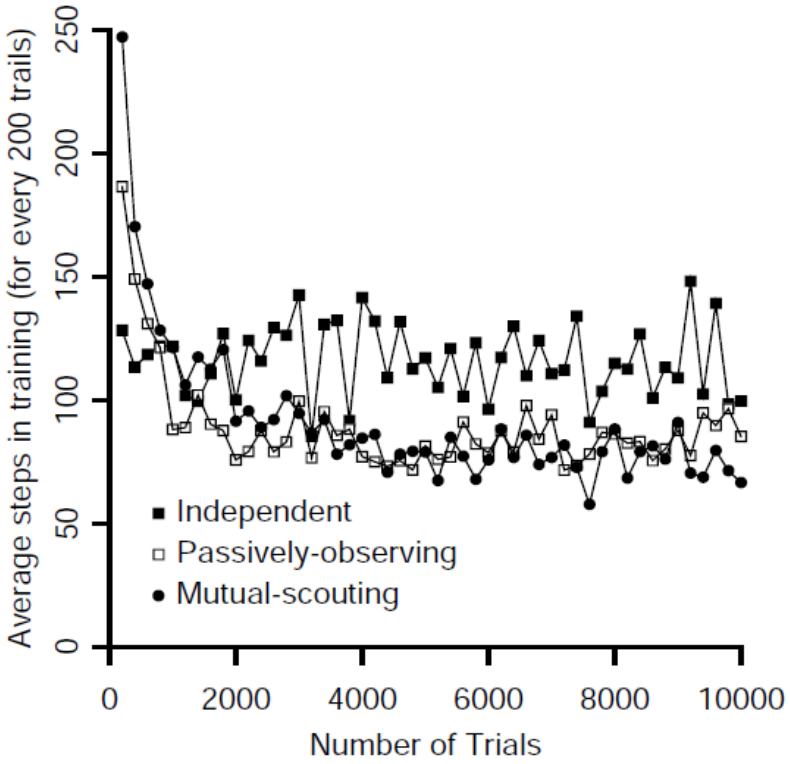


Figura 4.6: Experimentos realizados por (TAN, 1993) - Presa em movimento

scouting estabilizou-se em uma média aproximada de 90 passos por episódio. A abordagem utilizada por (TAN, 1993) demonstrou um desempenho inferior em relação ao desempenho alcançado pelas abordagens utilizadas nos experimentos VI e VII.

Nos experimentos VI e VII foram comparadas diferentes representações (tabular e *tile coding*) através dos algoritmos *Q-Learning* e $Q(\lambda)$, respectivamente. Os parâmetros de aprendizado utilizados no algoritmo *Q-Learning* e no método OPPORTUNE para os experimentos neste cenário foram $\alpha = 0.5$, $\gamma = 0.9$ e $\varepsilon = 0.1$ para política de seleção de ações ε-gulosa. Os parâmetros de aprendizado utilizados no uso de *tile coding* para os experimentos neste cenário foram $\alpha = 0.01$, $\gamma = 0.9$, $\gamma = 0.9$ e $\varepsilon = 0.1$. Os parâmetros de generalização utilizados foram $|\vec{T}| = 10$ e $|\vec{t}| = 10$. Estes valores foram escolhidos para verificar que, mesmo com utilizando valores de parâmetros com desempenho médio (Experimentos II a V), a utilização de *tile coding* produz um desempenho superior.

É possível observar que o parâmetro α utilizado nos experimentos com o algoritmo $Q(\lambda)$ possui um valor α muito menor ($\alpha = 0.01$) do que o valor utilizado no algoritmo *Q-Learning* ($\alpha = 0.5$). Isso deve-se ao fato de que as atualizações realizadas podem representar muitos estados e o tamanho de cada passo em direção à função alvo deve ser lento para permitir a generalização.

A Figura 4.7 mostra o desempenho de captura da presa no experimento VI. É possível observar o desempenho superior de *tile coding* em comparação ao método IL tabular. Esse fato ocorre principalmente pelo fato de que *tile coding* foi empregado utilizando o modelo de agente caixa branca (Seção 2.7.2), ou seja, ambos agentes "predadores" possuem aprendizado conjunto e compartilham inteiramente suas percepções e ações.

O método IL tabular teve desempenho inferior, pois o algoritmo foi aplicado individualmente em cada agente "predador". Não foi possível realizar experimentos com o algoritmo *Q-Learning* em um processo de aprendizado conjunto (JAL) dos dois agentes

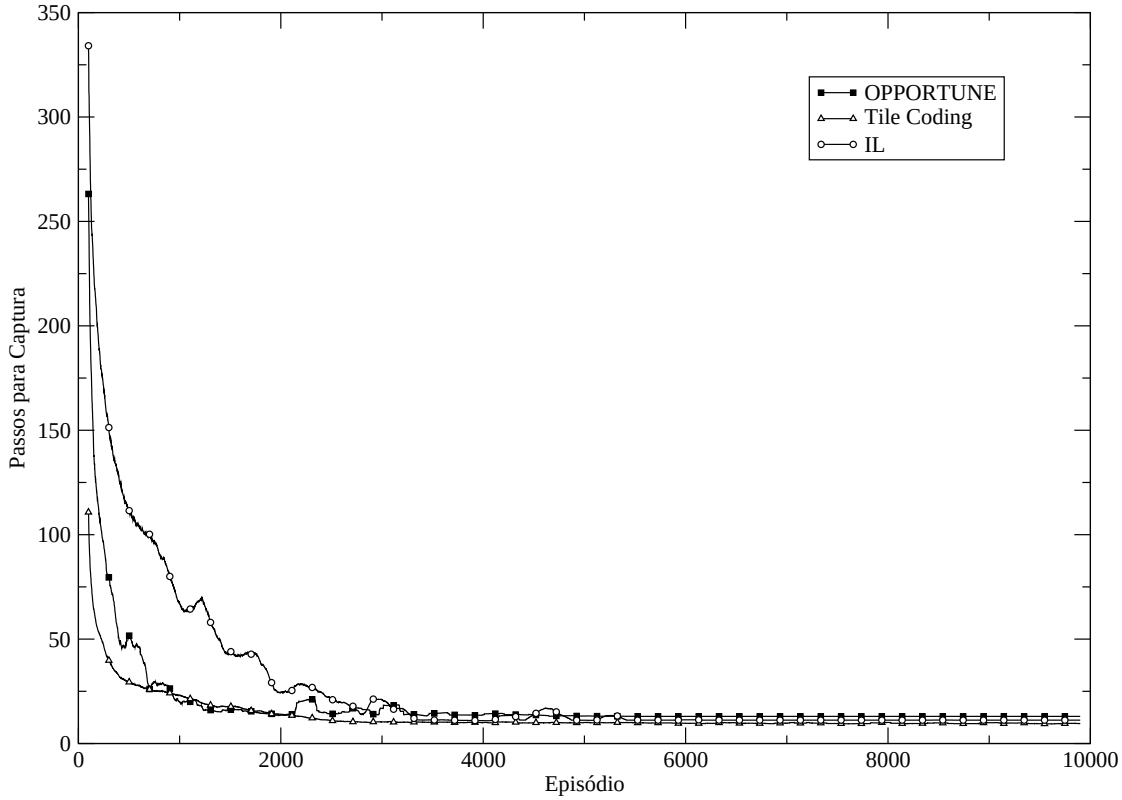


Figura 4.7: Experimento VI - Presa parada - Comparação de desempenho

"predadores" dado que o espaço de estados é muito grande ($7.311.616$ $(26^4 \times 4^2)$). Provavelmente, se fosse possível implementar o algoritmo *Q-Learning* nos mesmos moldes que foi implementado o algoritmo $Q(\lambda)$ o algoritmo *Q-Learning* teria desempenho igual ou mesmo superior ao algoritmo $Q(\lambda)$. O OPPORTUNE demora mais a atingir o número de passos para a captura igual aos demais, estabilizando o número de passos para captura apenas depois de aproximadamente 5 mil episódios. Isto se deve ao tempo necessário em que o OPPORTUNE realiza a adaptação da tabela Q.

A Figura 4.8 refere-se ao experimento VII. É possível observar que o *tile coding* teve um desempenho somente um pouco inferior ao desempenho obtido pelo mesmo método no experimento VI. Obviamente isso foi ocasionado por causa da presença ou não de movimento da presa. Entretanto, o desempenho do IL foi muito inferior ao obtido pelo mesmo método no experimento VI. Isto foi ocasionado pela utilização de um processo de aprendizado individual para realizar a captura da presa se movimentando aleatoriamente.

Igualmente ao experimento VI, esse processo de captura realizado com o IL de maneira individual a cada agente "predador" seria muito mais eficaz se fosse realizado de maneira conjunta. Com isso, a utilização da técnica proposta neste trabalho no algoritmo $Q(\lambda)$ é vantajoso em comparação ao método tabular, tendo em vista que este método possibilita a generalização de espaço de estados muito grandes. Por exemplo, um espaço de estados conjunto formado pelos espaços de estados individuais de dois agentes "predadores".

O OPPORTUNE apresentou desempenho inferior a todos no início da simulação, mas atingiu um nível de desempenho próximo ao desempenho do IL através do algoritmo *Q-Learning* após o período de adaptação. É possível observar que o OPPORTUNE apresentou oscilações no decorrer do episódio, isso ocorre por causa do processo de adaptação

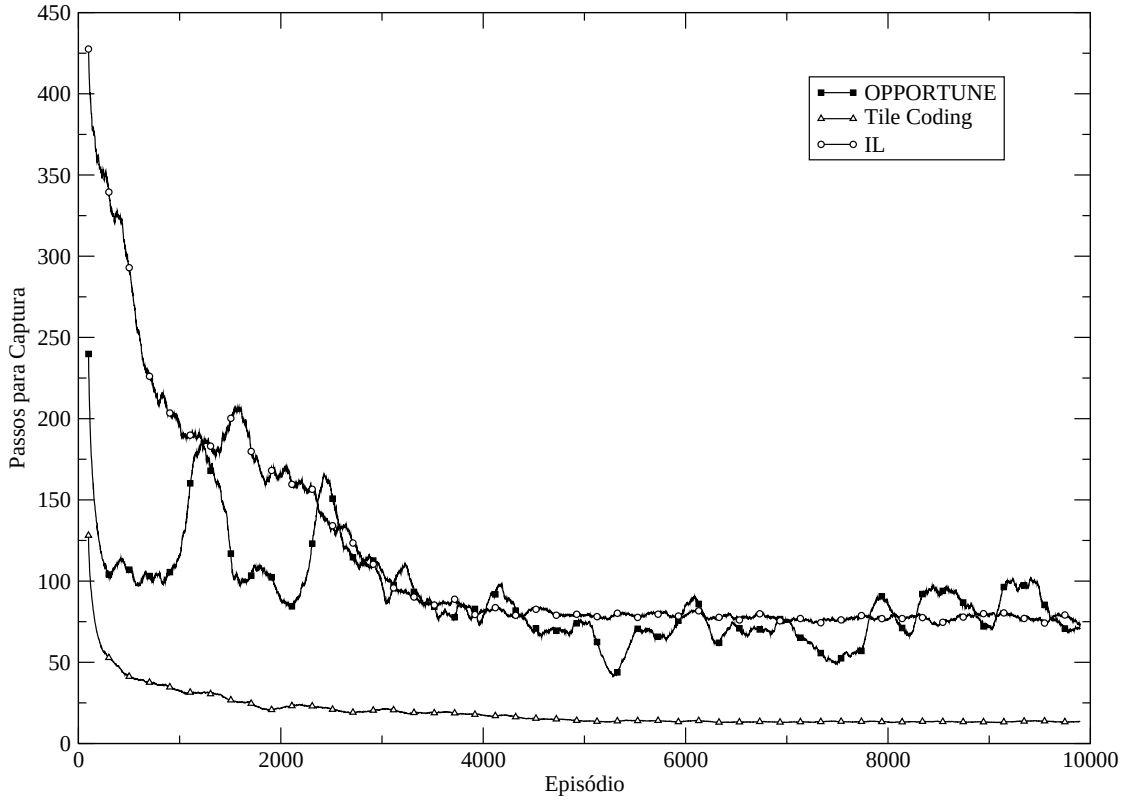


Figura 4.8: Experimento VII - Presa em movimento - Comparação de desempenho

da tabela Q que o OPPORTUNE realiza na tentativa de aumentar a percepção de cada agente.

4.1.3 Análise do Cenário Presa-Predador

No cenário presa-predador foi aplicado *tile coding* e comparado ao algoritmo *Q-Learning* e ao método OPPORTUNE. A questão principal para aplicação de aprendizado neste tipo de cenário é a capacidade de abstração do espaço de estados conjunto dos 2 agentes predadores.

Como mencionado anteriormente, o número de entradas na tabela $Q(s, a)$ é muito grande e são diversas as abordagens para lidar com esse tipo de problema. Em (TAN, 1993) são citadas algumas alternativas como o compartilhamento somente de percepções ou diminuição da percepção, entre outras. (OLIVEIRA, 2009) trata o problema de dimensionalidade utilizando-se uma tabela $Q(s, a)$ e processos de negociação entre agentes. Entretanto, nenhum destes métodos, com exceção de cenário com grande espaço de busca, tendem a não ser tão eficazes quanto o compartilhamento total e irrestrito de percepções ações por parte dos agentes.

Desta forma, a aplicação de um método linear de aproximação de funções como *tile coding* neste cenário mostrou-se eficaz no sentido de proporcionar o tratamento de um espaço de estados e ações de alta dimensionalidade. Como por exemplo, como o espaço de estados e ações que é formado pelas percepções e ações conjuntas dos agentes predadores.

4.2 Cenário de Controle Semafórico

Neste tipo de problema (descrito na Seção 3.2.2), foram utilizadas duas configurações de redes de tráfego, de acordo com o número de cruzamentos semaforizados e controlados: 9 e 81 semáforos. Na Figura 4.9 são mostradas as redes de tráfego, onde as linhas representam as vias e os círculos representam os semáforos (descritos na Seção 3.2.3). O primeiro cenário pode ser visualizado no retângulo destacando uma grade 3×3. O segundo cenário é similar, porém maior: grade 9×9. Em ambos os cenários há um agente controlando cada semáforo, como por exemplo, os semáforos localizados em A1 até I9. Cada via possui apenas um sentido de tráfego, que pode ter duas direções: horizontal e vertical.

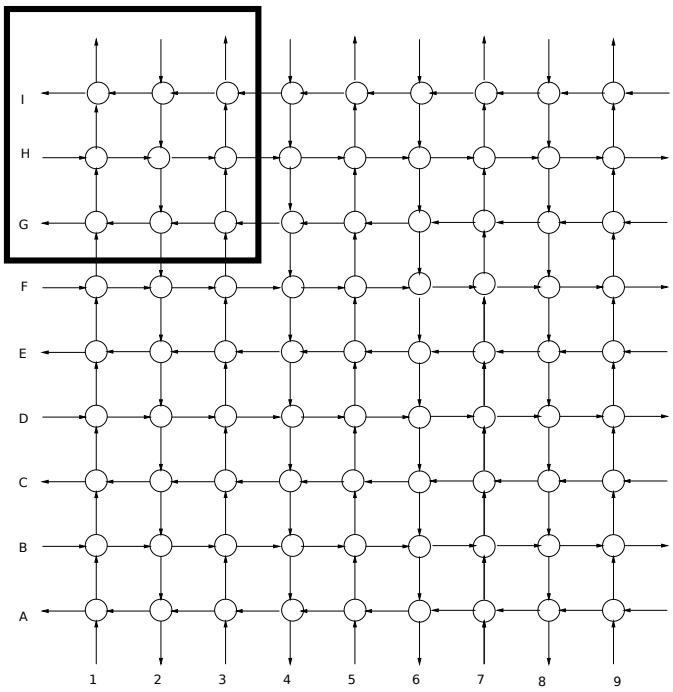


Figura 4.9: Cenários de redes de tráfego: grade 3×3 e grade 9×9

Entre os nodos (cruzamentos) foi configurada a distância de 300 metros, constituindo uma distância razoável para que haja possibilidade de movimentação, sem que se force congestionamento por motivo de distâncias pequenas. De acordo com o modelo de autômato celular utilizado em (SILVA et al., 2006) (Seção 3.2.4), foram utilizadas células de 5 metros, ou seja, cada via (Seção 3.2.3) é formada por 60 células. Com essa configuração pode haver no máximo 60 veículos parados por trecho. Além disso, a velocidade máxima é de 3 células por iteração, o que corresponde a 15 m/s ou aproximadamente 54 Km/h, valor adequado para simulação de tráfego urbano.

Como mostra a Tabela 4.2 foram realizados cinco experimentos neste tipo de cenário. Para ambos cenários (3×3 e 9×9) foram executados experimentos em função dos parâmetros de generalização (experimentos VIII e X) e foram executados experimentos destinados à comparação de desempenho das representações tabular e *tile coding* (experimentos IX e XI). No experimento XII é realizada uma comparação episódica, alterando-se a taxa de inserção de veículos durante a simulação, entre as representações tabular e *tile coding*.

Cada cruzamento possui um semáforo controlando o fluxo das vias com planos semafóricos predeterminados. Foi definido um tempo de ciclo de 60 segundos para todos os

Tabela 4.2: Experimentos no cenário de controle semafórico

Experimento	Métodos	Número de agentes	$\|\vec{T}\|$	$\|\vec{t}\|$	Inserção Horizontal	Inserção Vertical
VIII	*Tile Coding*	9	1,5,9	1,5,9	1008	1008
IX	*Tile Coding*, IL, JAL	9	5	1	1008	1008
X	*Tile Coding*	81	1,5,9	1,5,9	1008	25.2
XI	*Tile Coding*, IL	81	9	9	1008	25.2
XII	*Tile Coding*, IL	81	9	9	1008 1008 25.2	25.2 1008 1008

experimentos. Para os experimentos de VIII até XI foram configurados 2 planos semafóricos e para cada plano as fases foram divididas em tempos de 40 e 20 segundos por ciclo. No experimento XII foram configurados 3 planos semafóricos e para cada plano as fases foram divididas em tempos de 40, 30 e 20 segundos por ciclo.

A cada 120 segundos (dois ciclos), o simulador requisita os planos para os agentes controladores. Os experimentos de VIII até XI aqui mostrados tem um total de 86400 passos e o experimento XII tem um total de 777600, sendo que os primeiros 600 passos servem para iniciar a rede e não há atuação dos agentes. Para cada experimento foram executadas 10 repetições das simulações.

O número de conjuntos de *tilings* (ϕ), conforme o método proposto neste trabalho, foi parametrizado de acordo com o número de ações possíveis de cada agente (no caso deste cenário: 2 ações). Os parâmetros de aprendizado utilizados no algoritmo *Q-Learning* foram ajustados para $\gamma = 0.9$ e α variando conforme os experimentos IX e XI. Os parâmetros de aprendizado utilizados no algoritmo $Q(\lambda)$ foram ajustados para $\alpha = 0.01$, $\gamma = 0.9$ e $\lambda = 0.9$.

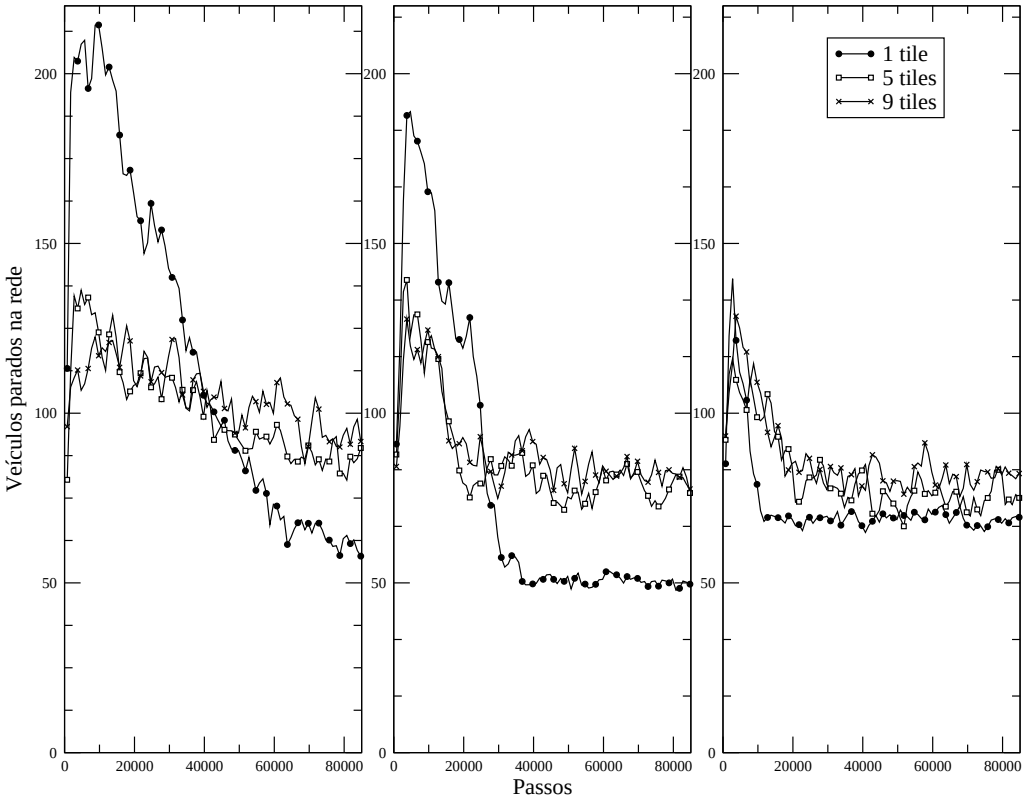
Para ambos algoritmos o valor do parâmetro $\varepsilon = 0.02$ foi escolhido para permitir pouca exploração na seleção de ações. Foi utilizada pouca exploração, pois como este cenário é dinâmico, a utilização de um valor alto de exploração pode fazer com que o algoritmo de aprendizado não convirja para uma ação que minimize o número de veículos parados. Para cada experimento foram executadas 10 simulações e os valores obtidos são os valores médios destas 10 simulações.

Como forma de calibragem de parâmetros, o experimento VIII mostra a utilização de diferentes números de *tilings* em função de diferentes número de *tiles*. Desta forma, é possível analisar-se a variação no número de veículos parados na rede em função dos passos de simulação de acordo com o número de *tilings* e *tiles* utilizados.

Na Figura 4.10a é mostrado o desempenho quando é utilizado $|\vec{T}| = 1$ e $|\vec{t}| = 1$, 5 e 9. Na Figura 4.10b mostra-se o desempenho quando é utilizado $|\vec{T}| = 5$ e $|\vec{t}| = 1$, 5 e 9 e na Figura 4.10c quando é utilizado $|\vec{T}| = 9$ e e $|\vec{t}| = 1$, 5 e 9.

Na Figura 4.10a, é possível observar que no cenário 3×3 há como se obter um bom desempenho de generalização mesmo particionando o espaço de estados sem a utilização de sobreposição de camadas. Entretanto, ao utilizar o recurso de sobreposição (Figuras 4.10b e 4.10c) tem-se um desempenho superior, pois cada agente consegue perceber o estado das vias com um número maior de características.

Além disso, ao contrário dos valores pertencentes a somente 1 tile, cujos valores variam abruptamente, nos experimentos que usam mais de uma camada sobreposta (5 e 9 *tilings*) e mais de uma partição (5 e 9 *tiles*), é possível verificar a suavização dos estados



(a) *1 tiling* (b) *5 tilings* (c) *9 tilings*

Figura 4.10: Experimento VIII - Grade 3×3 - Comparação do número de *tilings* em função do número de *tiles*

representados.

No experimento IX foram comparadas diferentes representações do espaço de estados através dos algoritmos *Q-Learning* e Q(λ). Na Figura 4.11, pode-se verificar que a aplicação do algoritmo Q-Learning em agentes do tipo caixa preta obteve desempenho inferior aos demais métodos, pois utilizando a representação tabular, individualmente não é possível distinguir entre informações do ambiente, informações de outros agentes ou informações do ambiente sob influência de outros agentes.

A aplicação do algoritmo utilizando o método JAL foi executada de forma que todos os semáforos possuem somente um agente controlador, onde os estados são formados através das informações obtidas de todas as vias que incidem sobre cruzamentos no cenário e as ações possíveis são a combinação das ações individuais de todos os agentes em questão.

O algoritmo Q(λ) teve um desempenho superior no experimento IX comparado com os métodos IL e JAL, pois através da utilização do modelo de partição do espaço de estados, cada agente tem como representar as interações com o ambiente e com outros agentes e assim, selecionar a ação que maximiza o reforço esperado de acordo com o estado que efetivamente está sendo representado, mesmo que este estado tenha características contínuas.

Desta forma, pela aproximação de valores cada agente possui subsídios para diferenciar informações oriundas do ambiente de informações oriundas de outros agentes e selecionar uma ação de acordo com o todo o contexto que foi percebido.

Na Figura 4.12 mostra o experimento X. Este experimento, da mesma forma que o

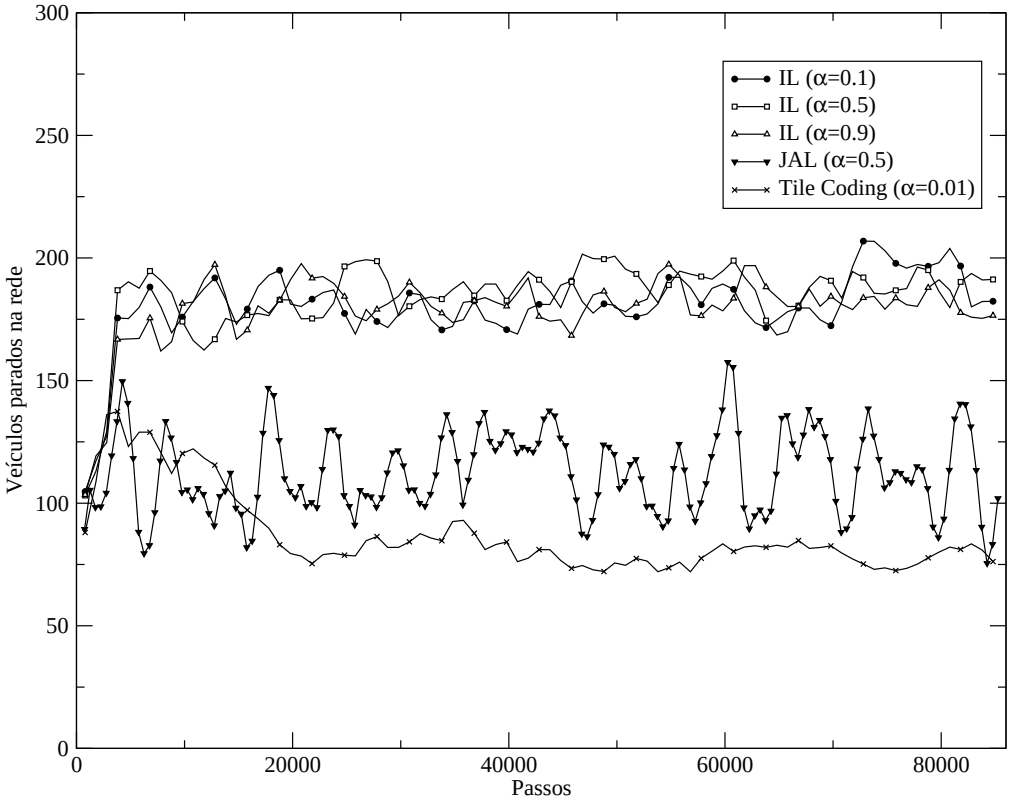
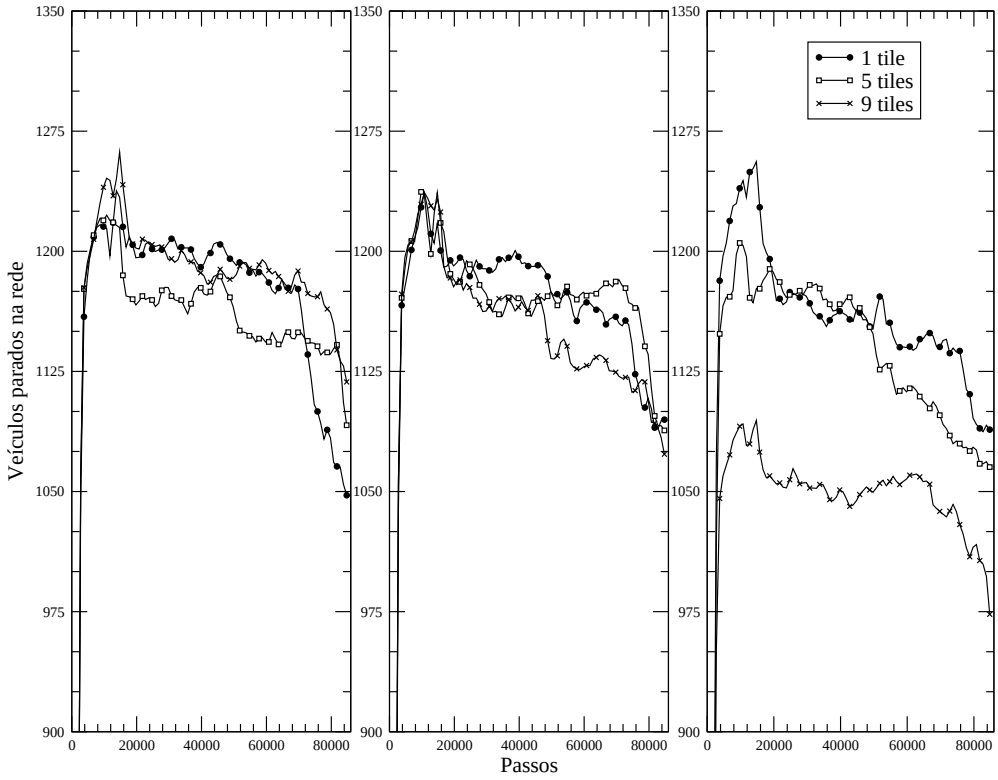


Figura 4.11: Experimento IX - Grade 3×3 - Comparação de desempenho



(a) *1 tiling* (b) *5 tilings* (c) *9 tilings*

Figura 4.12: Experimento X - Grade 9×9 - Comparação do número de *tilings* em função do número de *tiles*

experimento VIII, ilustra a utilização de diferentes números de *tilings* em função de diferentes configurações de particionamento *tiles*. Entretanto, esse experimento foi realizado em um cenário 9×9.

Devido ao tamanho do espaço de estados do cenário 9×9, foi observada uma relação direta do desempenho obtido em função da granularidade do particionamento deste espaço de estados, o que indica que não houve sobre ajuste no número de $|\vec{T}|$ e $|\vec{t}|$, onde justamente os parâmetros com maior número de particionamentos $|\vec{T}| = 9$ e $|\vec{t}| = 9$ (Figura 4.12c) produziram o melhor resultado.

Possivelmente, pelo fato deste cenário gerar uma elevada quantidade de estados e pelo resultados obtidos, seria necessário que os parâmetros de generalização fossem ajustados para um valor muito superior aos valores utilizados pelo experimento VII.

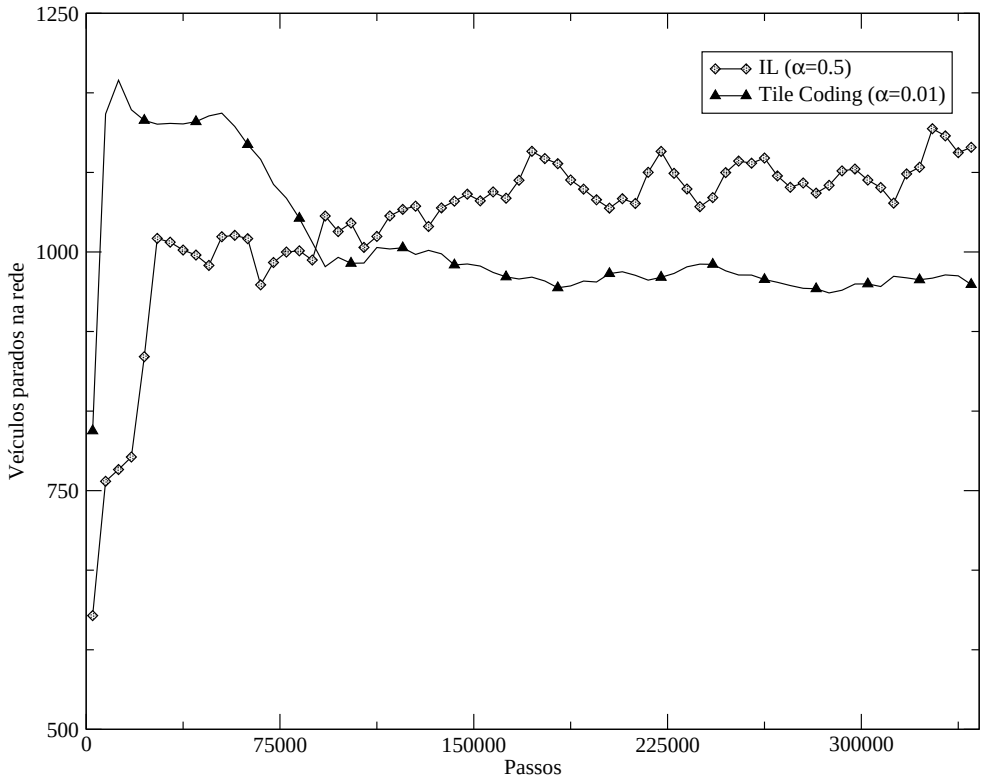


Figura 4.13: Experimento XI - Grade 9×9 - Comparação de desempenho

A Figura 4.13 mostra o número de veículos parados no experimento XI. Neste experimento, foram comparadas diferentes representações do espaço de estados através dos algoritmos *Q-Learning* e Q(λ).

Neste experimento não foi possível utilizar-se aprendizado conjunto (JAL) tendo em vista que uma implementação conjunta em um cenário deste porte facilmente ocasionaria uma explosão combinatorial do espaço de estados e ações. Assim, foram realizadas somente execuções dos algoritmos atuando como agentes individuais, ou seja, cada agente controlando somente um semáforo.

Foi verificado que inicialmente o algoritmo *Q-Learning* possui um desempenho superior, controlando cenário de forma a obter-se menos veículos parados. Contudo, na parte final da simulação foi verificado que, por causa da grande capacidade de representação, o algoritmo Q(λ) controlou o cenário de forma a deixar menos veículos parados nas vias.

A Figura 4.14 mostra o experimento XII. Este experimento foi conduzido de forma episódica alternando-se a taxa de inserção de veículos durante a simulação. O tempo total

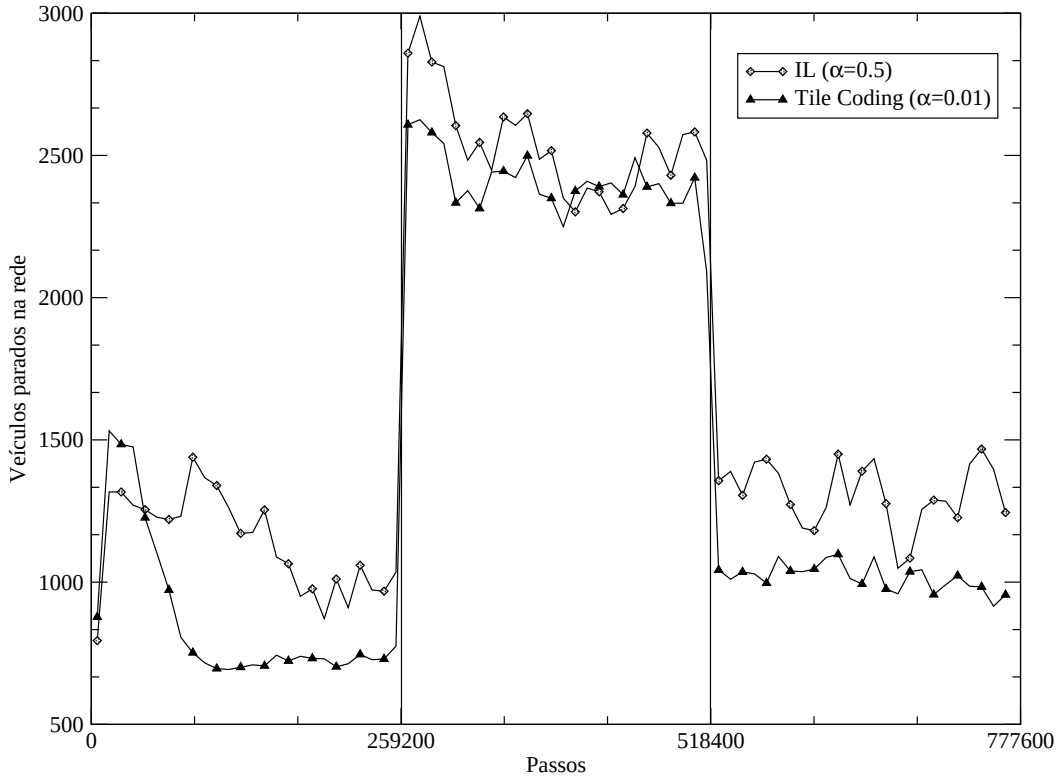


Figura 4.14: Experimento XII - Grade 9×9 - Comparação de desempenho episódico

de simulação é de 9 dias (777600 passos). Cada episódio tem duração de 3 dias (259200 passos). Na Figura 4.14 cada episódio está separado por uma linha vertical em múltiplos de 259200 passos.

É possível observar que nos 3 episódios a aplicação de *tile coding* obteve um desempenho superior à abordagem IL. Entretanto, no segundo episódio ambos os métodos apresentam desempenhos similares e no terceiro episódio praticamente não há tendência de queda no número de veículos parados na rede em ambos os métodos utilizados. Além disso, é possível observar que o número de veículos parados tem um decréscimo muito baixo do início terceiro episódio até o final.

4.2.1 Análise do Cenário de Controle Semafórico

No cenário de controle semafórico foi aplicado *tile coding* e comparado ao algoritmo *Q-Learning* utilizando as abordagens IL e JAL. O foco para aplicação de aprendizado neste tipo de cenário foi a granularidade das informações coletadas junto ao simulador ITSUMO.

Através dos experimentos realizados neste cenário, foi possível observar que uma abordagem IL pode não ser não ser eficiente, haja vista o dinamismo do cenário e a representação simplificada das informações das vias. Esta ineficiência poderia ser contornada através da utilização de comunicação com o algoritmo OPPORTUNE, por exemplo. Contudo, apesar do uso de comunicação ser bem estruturado atualmente, um sistema de controle de tráfego ser plenamente dependente de comunicação poderia entrar em colapso ao enfrentar qualquer falha na comunicação.

A abordagem através de JAL no experimento IX seria superior a todos os outros algoritmos testados dado que todas as ações possíveis são tratadas pela abordagem. Contudo, por ser um método conjunto, a busca pelo espaço de ações conjuntas é muito maior do

que a busca pelo espaço de ações individual. Por consequência, o número de iterações necessárias para convergir para uma ação conjunta que maximize o reforço esperado é muito maior.

A utilização de *tile coding* mostrou-se eficaz no sentido de permitir um aprendizado sem a necessidade de comunicação entre os agentes. Este tipo de representação é poderosa o suficiente para ser aplicado como método de aprendizado independente, pois permite que um agente distingua o efeito de suas próprias ações em um ambiente dinâmico com outros agentes.

4.3 Jogos de Coordenação

Os experimentos em jogos de coordenação foram conduzidos através de jogos na forma normal onde jogadores escolhem suas ações de maneira coordenada.

Neste cenário foram realizados experimentos divididos em: matriz de recompensa arbitrária e matriz de recompensa com equilíbrio Pareto dominante. Foram realizados experimentos no sentido de verificar como cada agente aprende a jogar um jogo de coordenação.

Nos experimentos com matriz de recompensa arbitrária, para cada agente é atribuída uma matriz de recompensa construída aleatoriamente conforme a Tabela 4.4. Nos experimentos com matriz de recompensa com equilíbrio Pareto dominante foi atribuída uma matriz de recompensa conforme a Tabela 4.5.

Em todos os experimentos, cada agente possui uma matriz de recompensa que representa seu estado e este estado muda ao longo do tempo. Sobre essa dinâmica de mudança de estados, a cada passo de tempo, o ambiente realiza a seleção de estados em cada agente pode estar com um estado em t diferente do estado em que este se encontrava em t_1. Utilizando como exemplo a Tabela 4.4, essa mudança de estados é realizada invertendo-se os valores de O_aO_a com os valores de O_bO_b. O ambiente realiza a mudança com uma probabilidade 0.9 para selecionar o estado 1 e com probabilidade 0.1 para selecionar o estado 2.

A Tabela 4.3 mostra os experimentos realizados neste cenário. Foram realizados quatro experimentos, sendo que os experimentos XIII e XIV foram destinados à avaliação de desempenho a partir da aplicação de matriz de recompensa arbitrária e os experimentos XV e XVI foram destinados a avaliação de desempenho a partir de matrizes de recompensa com equilíbrios Pareto dominante.

Em termos de ambiente, os agentes estão dispostos grades de tamanho 4×4 e 24×24. Este tipo de ambiente em grade difere de abordagens tradicionais de TJ interativos, onde somente 2 jogadores executam as jogadas repetidamente. Na abordagem utilizada neste trabalho, cada agente joga repetidamente com os 4 agentes vizinhos na grade.

Para cada experimento foram executadas 10 repetições das simulações. Os valores de $|\vec{T}|$ e $|\vec{t}|$ foram definidos em função destes valores terem sido os melhores resultados obtidos na aplicação de *tile coding* neste cenário. Nos experimentos foram utilizados os algoritmos *Q-Learning* utilizando a abordagem IL (Seção 2.7.2) e Q(λ) utilizando *tile coding*.

4.3.1 Experimentos com matriz de recompensa arbitrária

Os experimentos com matriz de recompensa arbitrária foram conduzidos com a finalidade de verificar se a utilização de *tile coding* proporciona condições de aprendizado em um ambiente em que todos agentes possuam uma matriz de recompensa diferente para

Tabela 4.3: Experimentos no cenário de jogos de coordenação

Experimento	Algoritmo	$\|\vec{T}\|$	$\|\vec{t}\|$	Matriz de Recompensa	Nºde agentes
XIII	IL, *Tile Coding*	9	1	Tabela 4.4	16
XIV	IL, *Tile Coding*	9	8	Tabela 4.4	576
XV	IL, *Tile Coding*	10	5	Tabela 4.5	16
XVI	IL, *Tile Coding*	10	5	Tabela 4.5	576

cada agente.

A seleção destes valores para o conjunto de ações em que ambos agentes escolhem a mesma ação O_aO_a e O_bO_b é feita de maneira aleatória. Entretanto, esta seleção aleatória está contida nos intervalos de valores dispostos na Tabela 4.4.

Para o conjunto de ações em que agentes escolhem ações diferentes como por exemplo O_aO_b e O_aO_b, é atribuído o valor 0. Após ser atribuída uma matriz de recompensas para cada agente, essa matriz é duplicada e durante a simulação com uma probabilidade p, os valores de O_aO_a e O_bO_b podem ser invertidos a fim de trocar os valores da matriz de recompensa de cada agente.

Tabela 4.4: Matriz de recompensas arbitrária

		Agente 2	
		O_a	O_b
Agente 1	O_a	$]1/1[-[2/2]$	$0/0$
	O_b	$0/0$	$[1/1]-]2/2[$

Neste tipo de experimento, para que fosse verificado o quanto cada agente fosse capaz de escolher uma ação que maximizasse seu retorno, foi escolhida a ação que retorna o maior valor da matriz de recompensa. Neste caso, a ação escolhida pelos 2 agentes que maximiza o retorno de ambos foi chamada de ação coordenada.

Desta forma, a porcentagem de escolha de ações coordenadas foi escolhida como métrica, pois não é possível avaliar a recompensa obtida pelo grupo de agentes como um todo, tendo em vista que cada agente possui uma matriz de recompensas arbitrária cujos valores são atribuídos aleatoriamente para cada agente.

A Figura 4.15 mostra uma comparação de desempenho entre os algoritmos de aprendizado no experimento que utiliza matriz de recompensa arbitrária. Os parâmetros de aprendizado utilizados no algoritmo *Q-Learning* foram fixados em $\gamma = 0.9$ e $\alpha = 0.5$. Os parâmetros de aprendizado utilizados no algoritmo $Q(\lambda)$ foram ajustados para $\alpha = 0.01$, $\gamma = 0.9$ e $\lambda = 0.9$. Estes valores foram fixados conforme os valores utilizados nos experimentos nos cenários anteriores. Para ambos algoritmos o valor do parâmetro $\varepsilon = 0.3$ foi ajustado para sofrer decaimento exponencial até o fim da simulação conforme a Equação 4.1, onde $\varepsilon(t)$ é valor atualizado de ε no tempo t, ε_0 é o valor inicial de ε e, t_{total} é número total de passos. Este tipo de decaimento permite relativa exploração no início da simulação e seleção gulosa de ações no término da simulação.

$$\varepsilon(t) = \varepsilon_0 \ exp(-\frac{t}{t_{total}}) \tag{4.1}$$

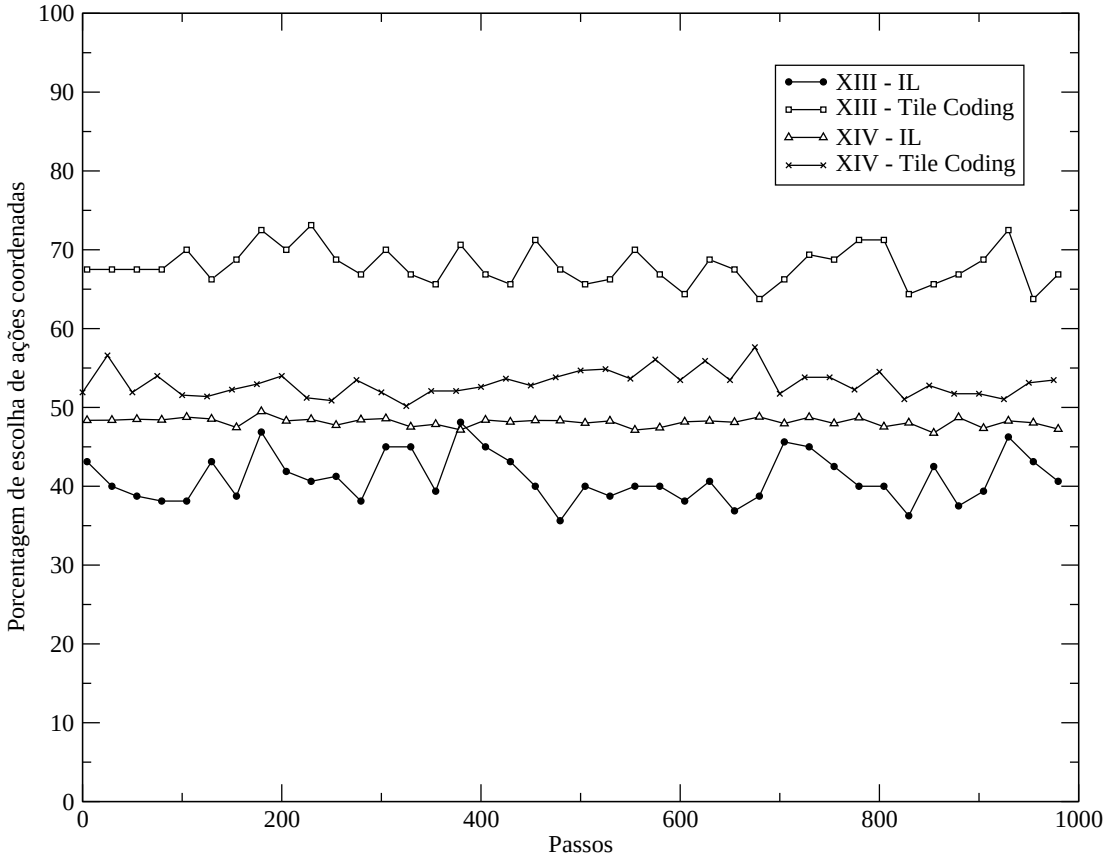


Figura 4.15: Experimentos XIII e XIV - Comparação de desempenho

Os parâmetros do *tile coding* utilizados foram $|\vec{T}| = 9$ e $|\vec{t}| = 1$ para o experimento XIII e $|\vec{T}| = 9$ e $|\vec{t}| = 8$ para o experimento XIV. Estes valores foram escolhidos por terem sido os valores que proporcionaram as porcentagens de escolha de ações coordenadas mais altas dentre os parâmetros testados previamente ($|\vec{T}| = [1, 10]$ e de $|\vec{t}| = [1, 10]$).

É possível observar na Figura 4.15 que o emprego de *tile coding* teve um desempenho superior na porcentagem de ações coordenadas escolhidas em relação ao algoritmo *Q-Learning* em ambos os experimentos. Entretanto, é válido ressaltar que essa superioridade não é atingida em todos os parâmetros de generalização testados.

4.3.2 Experimentos com matriz de recompensa com equilíbrio Pareto dominante

Os experimentos com matriz de recompensa com equilíbrio Pareto dominante foram conduzidos com a finalidade de verificar se a utilização de *tile coding* é capaz de fazer o agente aprender a escolher a ação que traz a recompensa máxima. Neste sentido, para cada agente foi atribuído um estado conforme a Tabela 4.5. Da mesma forma que os experimentos realizados com uma matriz de recompensa arbitrária, para o conjunto de ações em que agentes escolhem ações diferentes como por exemplo O_aO_b e O_aO_b, é atribuído o valor 0 para ambos os jogadores. Após ser atribuída uma matriz de recompensas para cada agente, essa matriz é duplicada e durante a simulação com uma probabilidade p, os valores de O_aO_a e O_bO_b podem ser invertidos a fim de trocar os valores da matriz de recompensa de cada agente.

Nos experimentos XV e XVI foram realizadas comparações de desempenho em função da recompensa média global obtida por todos os agentes situados nos ambientes do tipo grade. Os parâmetros utilizados para os algoritmos de aprendizado foram os mesmos

Tabela 4.5: Matriz de recompensas com equilíbrio Pareto dominante

		Agente 2	
		O_a	O_b
Agente 1	O_a	$2/2$	$0/0$
	O_b	$0/0$	$1/1$

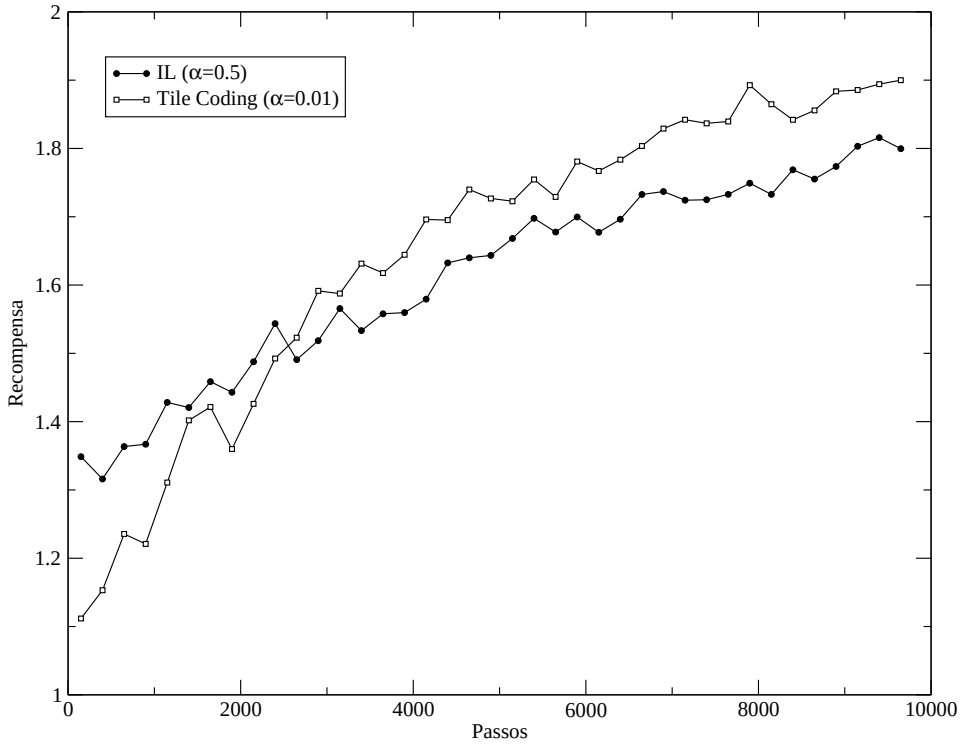


Figura 4.16: Experimento XV - Grade 4×4 - Comparação de desempenho

utilizados nos experimentos realizados na Subseção 4.3.1. Entretanto, os parâmetros de generalização utilizados foram $|\vec{T}| = 10$ e $|\vec{t}| = 5$. Estes valores foram utilizados tendo em vista que foram os valores que proporcionaram os melhores resultados.

A Figura 4.16 mostra os resultados do experimento XV. Este experimento foi realizado em uma grade 4×4. É possível verificar que inicialmente o algoritmo *Q-Learning* utilizando representação tabular possui um desempenho superior em relação à recompensa total obtida pelo grupo de agentes. Entretanto, próximo aos 3000 passos, o algoritmo Q(λ) utilizando *tile coding* supera o desempenho do algoritmo *Q-Learning*.

A Figura 4.17 mostra os resultados do experimento XVI para uma grade 24×24. Com um comportamento similar ao apresentado no experimento XV, inicialmente, o algoritmo *Q-Learning* utilizando representação tabular possui desempenho superior e, em 3000 passos, é superado pelo algoritmo Q(λ) utilizando *tile coding*.

4.3.3 Análise do Cenário de Jogos de Coordenação

No cenário de jogos cooperativos foi aplicado *tile coding* e comparado ao algoritmo *Q-Learning* utilizando a abordagem IL. A questão principal para aplicação de aprendizado neste tipo de cenário é a representação do espaço de estados formado pelos valores

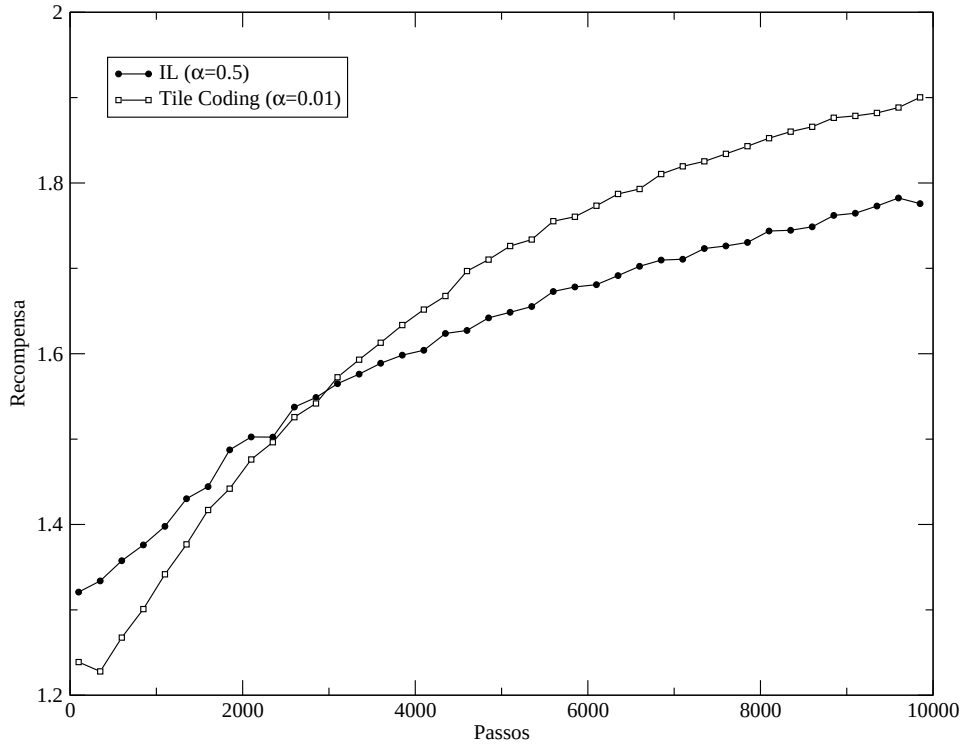


Figura 4.17: Experimento XVI - Grade 24×24 - Comparação de desempenho

das matrizes de recompensa. Nos experimentos XIII e XIV, o uso de *tile coding* obteve desempenho superior. Contudo, este desempenho somente foi alcançado quando são utilizados valores de parâmetros específicos.

O desempenho apresentado nos experimentos XV e XVI por ambos algoritmos (*Q-Learning* e Q(λ)) está relacionado ao tamanho do espaço de estados inicial para cada tipo de representação. Utilizando-se *tile coding*, é necessária a definição *a priori* dos limites inferior e superior. Assim, *a priori* tem-se definido o número de partições a ser utilizado pelo mecanismo de aprendizado e por consequência, a delimitação de todo espaço de estados.

Entretanto, ao utilizar a tradicional representação tabular, é comum a adição de novos estados à tabela Q somente quando estes são percebidos por cada agente. No estágio inicial das simulações, como ainda foram percebidos poucos estados, a atualização TD (Seção 2.4) ocorre eficazmente. Contudo, no decorrer da simulação, o número de estados percebidos tende a crescer e com isso, a representação por *tile coding* é favorecida, pois todos os estados percebidos já foram previamente generalizados.

5 CONCLUSÕES E TRABALHOS FUTUROS

5.1 Principais Conclusões

Neste trabalho, foram apresentadas aplicações de aprendizado por reforço com uso de *tile coding* em cenários onde vários agentes aprendem simultaneamente (aprendizado multiagente). Nestas aplicações, cada agente representa linearmente as percepções decorrentes da interação com o ambiente e com outros agentes.

O uso de uma técnica de aproximação de funções viabiliza a utilização de aprendizado por reforço em cenários com grandes espaços de estados e ações, pois esta técnica fornece subsídios para que se possa abstrair aspectos relevantes do aprendizado e assim, em alguns casos, contornar o problema da maldição da dimensionalidade.

Conforme exposto no Capítulo 2, os sistemas de aprendizado precisam lidar com a geração de estimativas a partir de outras estimativas ao mesmo tempo em que há a necessidade de ignorar dados irrelevantes e qualificar dados importantes. Neste contexto, foram descritos alguns métodos de generalização que visam o tratamento destas características em problemas de RL que possuam variáveis contínuas.

Além de auxiliar na resolução dos problemas acima descritos, existe a possibilidade da utilização de métodos de aproximação de funções simplesmente com o objetivo de tratar a recuperação de cada estado como um exemplo de treinamento. Segundo (SUTTON; BARTO, 1998), os métodos lineares de gradiente-descendente possuem um apelo teórico peculiar e ao mesmo tempo funcionam muito bem na prática se forem fornecidas as características apropriadas.

Neste trabalho foi possível observar que a escolha das características que compõem um estado é um dos requisitos mais importantes para a aquisição de conhecimento do domínio *a priori*. Assim, em um sistema multiagentes, a separação do conjuntos de *tilings* em função das ações disponíveis para agente é uma contribuição fundamental como etapa inicial na realização da representação do espaço de estados.

Através dos experimentos realizados e apresentados no Capítulo 4 observou-se que a utilização de *tile coding* em um problema caracterizado por um espaço de estados muito grande é viável desde que sejam utilizados os parâmetros adequados.

Nos experimentos buscou-se a escolha de parâmetros que demonstrassem o comportamento do emprego de *tile coding* em cada um dos três cenários aplicados. Em todos os cenários foram demonstrados experimentos realizados com diversos parâmetros de generalização.

No cenário Presa-Predador foram realizados experimentos comparando o desempenho dos algoritmos *Q-Learning* e $Q(\lambda)$ e do método OPPORTUNE, sendo que o algoritmo $Q(\lambda)$ foi aplicado utilizando percepção conjunta entre os agentes. Nos cenários de controle semafórico e jogos de coordenação foram realizados experimentos comparando

o desempenho dos algoritmos *Q-Learning* e $Q(\lambda)$.

Além disso, nos experimentos foram observados desempenhos superiores da representação do espaço de estados por *tile coding* em relação à representação tabular. Obviamente, como os parâmetros de generalização podem ser responsáveis pela representação de um ou mais estados, foi observado que estes bons desempenhos estão diretamente relacionados com a escolha dos parâmetros corretos.

É válido ressaltar algumas limitações do uso de *tile coding* que já foram descritas na Seção 3.1.3:

1. Existe a necessidade de conhecimento *a priori* dos limites inferior e superior do espaço de estados a ser particionado;

2. Ausência de um mecanismo de distribuição eficaz das camadas de *tilings* sobre o espaço de estados de forma que estados mais relevantes sejam representados por abstrações uma granularidade maior e que estados menos relevantes sejam representados por abstrações mais amplas.

Adicionalmente, pode-se destacar algumas limitações observadas em função dos experimentos realizados:

- O problema de configuração de parâmetros de aprendizado: é difícil encontrar bons parâmetros de aprendizado como o número de *tilings* e o número de tiles, ainda mais quando o algoritmo Watkins's Q(λ) (Algoritmo 6) demonstra-se muito sensível à variações nos parâmetros utilizados;

- O sobre-ajuste de características. É natural definir um número grande de características para modelar o espaço de estados no intuito de se obter uma representação abundante através de uma generalização granular. Contudo, nos experimentos foi observado que o excesso ou sobre ajuste das características ocasiona a deterioração do modelo e por consequência, a perda de desempenho do mecanismo de aprendizado (Experimento V na Seção 4.1.1 e Experimento VIII na Seção 4.2).

Por fim, é importante enfatizar novamente que, embora os resultados deste trabalho se mostrem bastante promissores, ainda há muitos pontos a serem explorados, principalmente, na escolha de parâmetros de aprendizado e de generalização.

5.2 Trabalhos Futuros

Devido às limitações enumeradas na seção anterior, diversas linhas podem ser seguidas:

- Criação de camadas de *tilings* de forma adaptativa: através de um mecanismo de detecção de estados com maior ou menor relevância, poderia se incluir ou retirar camadas de *tilings* do sistema a fim de tornar eficaz a atualização de valores. Contudo, seria necessário desenvolver adicionalmente um método que generalizasse os pesos θ contidos em partições utilizadas até então para os pesos θ das novas partições;

- Dimensionamento dinâmico de *tiles*: ao se utilizar um número grande de *tiles*, pode identificar quais são os *tiles* que são mais solicitados pelo processo de busca e alterar o tamanho ou subdividir estas partições de maneira a atribuir maior importância

aos *tiles* mais relevantes. Existem algumas abordagens para automatização da escolha do número de *tilings* e *tiles* em (SHERSTOV; STONE, 2005) e (WHITESON; TAYLOR; STONE, 2007). Entretanto, estas abordagens não foram aplicadas a cenários multiagentes;

- Criação de um método estatístico para que sejam encontrados parâmetros de generalização ótimos sem que haja a necessidade de uma verificação empírica: pode ser realizado através de um pré-processamento em relação à função de recompensa desejada verificando quais números de *tilings* e *tiles* possuem chances de se obter melhores resultados no processo de aprendizado;

- Ajuste dinâmico dos limites do espaço de estados: o modelo pode ser alterado para que caso o ambiente retorne um valor que extrapole os limites previamente definidos, o particionamento seja estendido a fim de contemplar o valor contido neste retorno.

REFERÊNCIAS

BAZZAN, A. L. C. A Distributed Approach for Coordination of Traffic Signal Agents. **Autonomous Agents and Multiagent Systems**, [S.l.], v.10, n.1, p.131–164, March 2005.

BAZZAN, A. L. C. Opportunities for Multiagent Systems and Multiagent Reinforcement Learning in Traffic Control. **Autonomous Agents and Multiagent Systems**, [S.l.], v.18, n.3, p.342–375, June 2009.

BAZZAN, A. L. C.; KLÜGL, F. Sistemas Inteligentes de Transporte e Tráfego: uma abordagem de tecnologia da informação. In: KOWALTOWSKI, T.; BREITMAN, K. K. (Ed.). **Anais das Jornadas de Atualização em Informática**. [S.l.]: SBC, 2007.

BELLMAN, R. E. **Dynamic Programming**. Princeton: Princeton University Press, 1957. 366p.

CLAUS, C.; BOUTILIER, C. The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems. In: FIFTEENTH NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE. **Proceedings...** [S.l.: s.n.], 1998. p.746–752.

DENATRAN. **Serviços de Engenharia**: manual de semáforos. Brasília: Concelho Nacional de Trânsito - Ministério da Justiça, 1979. 169p.

DENZINGER, J.; FUCHS, M. Experiments in Learning Prototypical Situations for Variants of the Pursuit Game. In: IN PROCEEDINGS OF THE SECOND INTERNATIONAL CONFERENCE ON MULTI-AGENT SYSTEMS (ICMAS). **Anais...** Menlo Park: AAAI Press, 1996. p.48–55.

DIAKAKI, C.; PAPAGEORGIOU, M.; ABOUDOLAS, K. A Multivariable Regulator Approach to Traffic-Responsive Network-Wide Signal Control. **Control Engineering Practice**, [S.l.], v.10, n.2, p.183–195, February 2002.

HOEN, P. J. 't et al. An Overview of Cooperative and Competitive Multiagent Learning. In: TUYLS, K. et al. (Ed.). **Learning and Adaptation in Multi-Agent Systems**. Berlin: Springer, 2006. p.1–49. (Lecture Notes in Artificial Intelligence, v.3898).

JELLE R. KOK, N. V. Sparse Cooperative Q-Learning. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 21., Banff, Canada. **Anais...** McGraw-Hill, 2004.

LEITE, J. G. M. **Engenharia de Tráfego**: métodos de pesquisa, características de tráfego, intersecções e sinais luminosos. São Paulo: Companhia de Engenharia de Tráfego - CET, 1980.

LI, L.; WALSH, T. J.; LITTMAN, M. L. Towards a Unified Theory of State Abstraction for MDPs. In: NINTH INTERNATIONAL SYMPOSIUM ON ARTIFICIAL INTELLIGENCE AND MATHEMATICS. **Proceedings...** [S.l.: s.n.], 2006. p.531–539.

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., Montreal, Québec, Canada. **Proceedings...** [S.l.: s.n.], 1995. p.394–402.

MERKE, A.; RIEDMILLER, M. A. Karlsruhe Brainstormers - A Reinforcement Learning Approach to Robotic Soccer. In: ROBOT SOCCER WORLD CUP (ROBOCUP 2001). **Proceedings...** Berlin: Springer, 2001. p.435–440. (Lecture Notes in Computer Science, v.2377).

NAGEL, K.; SCHRECKENBERG, M. A Cellular Automaton Model for Freeway Traffic. **Journal de Physique I**, [S.l.], v.2, p.2221, 1992.

NARENDRA, K. S.; THATHACHAR, M. A. L. **Learning Automata**: an introduction. Upper Saddle River, NJ, USA: Prentice-Hall, 1989. 476p.

OLIVEIRA, D. d. **Aprendizado em Sistemas Multiagente Através de Coordenação Oportunista**. 2009. Tese (Doutorado em Ciência da Computação) — Universidade Federal do Rio Grande do Sul.

PANAIT, L.; LUKE, S. Cooperative Multi-Agent Learning: the state of the art. **Autonomous Agents and Multi-Agent Systems**, Hingham, MA, USA, v.11, n.3, p.387–434, 2005.

PAPAGEORGIOU, M. et al. Review of Road Traffic Control Strategies. **Proceedings of the IEEE**, [S.l.], v.91, n.12, p.2043–2067, December 2003.

ROESS, R. P.; PRASSAS, E. S.; MCSHANE, W. R. **Traffic Engineering**. [S.l.]: Prentice Hall, 2004.

SHERSTOV, A. A.; STONE, P. Function Approximation via Tile Coding: automating parameter choice. In: ZUCKER, J.; SAITTA, I. (Ed.). **SARA 2005**. Berlin: Springer Verlag, 2005. p.194–205. (Lecture Notes in Artificial Intelligence, v.3607).

SHOHAM, Y.; LEYTON-BROWN, K. **Multiagent Systems**: algorithmic, game-theoretic, and logical foundations. [S.l.]: Cambridge University Press, 2009.

SILVA, B. C. d. et al. ITSUMO: an intelligent transportation system for urban mobility. In: INTERNATIONAL JOINT CONFERENCE ON AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS, AAMAS, 5. **Proceedings...** ACM Press, 2006. p.1471–1472.

STONE, P. Multiagent learning is not the answer. It is the question. **Artificial Intelligence**, Essex, UK, v.171, n.7, p.402–405, May 2007.

SUTTON, R.; BARTO, A. **Reinforcement Learning**: an introduction. Cambridge, MA: MIT Press, 1998. 322p.

SUTTON, R. S. Generalization in reinforcement learning: successful examples using sparse coding. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, Cambridge, MA. **Anais...** MIT Press, 1996. v.8, p.1038–1044.

TAN, M. Multi-Agent Reinforcement Learning: independent vs. cooperative agents. In: IN PROCEEDINGS OF THE TENTH INTERNATIONAL CONFERENCE ON MACHINE LEARNING. **Anais...** Morgan Kaufmann, 1993. p.330–337.

THORNDIKE, E. L. **Animal intelligence**: experimental studies. New York: Macmillan, 1911. 328p.

WATKINS, C. **Learning from Delayed Rewards**. 1989. Tese (Doutorado em Ciência da Computação) — University of Cambridge.

WATKINS, C. J. C. H.; DAYAN, P. Q-learning. **Machine Learning**, Hingham, MA, USA, v.8, n.3, p.279–292, 1992.

WHITESON, S.; TAYLOR, M. E.; STONE, P. **Adaptive Tile Coding for Value Function Approximation**. [S.l.]: University of Texas at Austin, 2007. (AI-TR-07-339).

WOOLDRIDGE, M. J. **An Introduction to MultiAgent Systems**. Chichester: John Wiley & Sons, 2002.