Universidade Federal da Paraíba
Centro de Informática
Programa de Pós-Graduação em Informática

Análise e Implementação de Algoritmos para a Aprendizagem por Reforço

Thiago Rodrigues Medeiros

Proposta de Dissertação submetida à Coordenação do Curso de Pós-Graduação em Informática da Universidade Federal da Paraíba como parte dos requisitos necessários para obtenção do grau de Mestre em Informática.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Computação Distribuída

Andrei de Araújo Formiga
(Orientador)

João Pessoa, Paraíba, Brasil

Thiago Rodrigues Medeiros

ANÁLISE E IMPLEMENTAÇÃO DE ALGORITMOS PARA A APRENDIZAGEM POR REFORÇO

Dissertação de Mestrado apresentada ao Programa de Pós Graduação em Informática da Universidade Federal da Paraíba como requisito para a obtenção do título de Mestre em Informática

Orientador: Prof. Dr. Andrei de Araújo Formiga

João Pessoa
2014

M488a Medeiros, Thiago Rodrigues.
Análise e implementação de algoritmos para a aprendizagem por reforço / Thiago Rodrigues Medeiros.- João Pessoa, 2014.
119f. : il.
Orientador: Andrei de Araújo Formiga
Dissertação (Mestrado) – UFPB/CI
1. Informática. 2. Ciência da computação. 3. Aprendizado de máquina. 4. Aprendizado por reforço. 5. Análise. 6.Biblioteca.

UFPB/BC CDU: 004 (043)

Ata da Sessão Pública de Defesa de Dissertação de Mestrado de **THIAGO RODRIGUES MEDEIROS**, candidato ao Título de Mestre em Informática na Área de Sistemas de Computação, realizada em 14 de Fevereiro de 2014.

Ao décimo quarto dia do mês de fevereiro do ano dois mil e quatorze, às quatorze horas, no auditório do CCEN - Universidade Federal da Paraíba - reuniram-se os membros da Banca Examinadora constituída para examinar o candidato ao grau de Mestre em Informática, na área de "*Sistemas de Computação*", na linha de pesquisa "*Computação Distribuída*", a Sr. **Thiago Rodrigues Medeiros**. A comissão examinadora foi composta pelos professores doutores: ANDREI DE ARAUJO FORMIGA (PPGI-UFPB), Orientador e Presidente da Banca, CLAUIRTON DE ALBUQUERQUE SIEBRA (PPGI-UFPB), examinador interno e LEANDRO BALBY MARINHO (UFCG), como examinador externo. Dando início aos trabalhos, o professor ANDREI DE ARAUJO FORMIGA cumprimentou os presentes, comunicou aos mesmos a finalidade da reunião e passou a palavra ao candidato para que o mesmo fizesse, oralmente, a exposição do trabalho de dissertação intitulado "*Análise e Implementação de Algoritmos para a Aprendizagem por Reforço*". Concluída a exposição, o candidato foi arguido pela Banca Examinadora que emitiu o seguinte parecer: "*Aprovado*". Assim sendo, deve a Universidade Federal da Paraíba expedir o respectivo diploma de Mestre em Informática na forma da lei e, para constar, eu, Alisson Vasconcelos de Brito, Coordenador do PPGI, servindo de secretário, lavrei a presente ata que vai assinada por mim e pelos membros da Banca Examinadora. João Pessoa, 14 de fevereiro de 2014.

Alisson Vasconcelos de Brito

Prof. Dr. Andrei de Araújo Formiga
Orientador (PPGI-UFPB)

Prof. Dr. Clauirton de Albuquerque Siebra
Examinador Interno (PPGI-UFPB)

Prof. Dr. Leandro Balby Marinho
Examinador Externo (UFCG)

Resumo

A Aprendizagem por Reforço é um subcampo do Aprendizado de Máquina e pode ser definido como um problema de aprendizagem. Um sistema inteligente que enfrenta esse problema, entende a partir de recompensas, se as ações que está realizando no ambiente são boas ou ruins. Existem vários métodos e algoritmos encontrados na literatura para resolver os problemas de aprendizagem por reforço, no entanto, cada um deles possuem suas vantagens e desvantagens. A partir disso, esse trabalho apresenta uma análise estatística de alguns algoritmos e uma biblioteca de aprendizagem por reforço, chamada *AILibrary-RL*. A *AILibrary-RL* é uma biblioteca que possui o objetivo de facilitar, organizar e promover a reusabilidade de código, para a implementação de sistemas que possuem esse tipo de problemática. Antes de seu desenvolvimento, foi realizado um levantamento bibliográfico dos principais métodos que solucionam a problemática de AR, visando a análise estatística dos mesmos, com o objetivo de avaliar suas vantagens e desvantagens em ambientes variados. Nesta dissertação está descrito todo o processo deste trabalho, desde o levantamento bibliográfico, análise dos métodos, mecanismos e construção da biblioteca.

Palavras-chave: Aprendizado de Máquina, Aprendizado por Reforço, Análise, Biblioteca.

Abstract

The Reinforcement Learning is a subfield of machine learning and can be defined as a learning problem. An intelligent system that faces this problem, understands from rewards if the actions you are performing in the environment are good or bad. There are several methods and algorithms found in the literature to solve the problems of reinforcement learning. However, each of them have their advantages and disadvantages. From this, this paper presents a statistical analysis of some algorithms and a library of reinforcement learning, called *AILibrary-RL*. The *AILibrary-RL* is a library that has the objective to facilitate, organize and promote reusability of code, to implement systems that have this kind of problem. Before its development, a bibliographic survey of the main methods that solve this problem, aimed at statistical analysis of the data was performed in order to evaluate its advantages and disadvantages in different environments. This dissertation described the whole process of this work, since the survey bibliographic, analysis of the methods, mechanisms and library construction.

Keywords: Machine Learning, Reinforcement Learning, Library, Statistical Analysis.

Agradecimentos

Agradeço primeiramente a Deus por ter me iluminado todos esses anos.

Agradeço aos meus pais Francisco e Suely por terem me apoiado todo o mestrado, sempre me dando conselhos motivadores para continuar firme nas minhas decisões. Vocês são meus alicerces nesta vida, amo muito vocês.

Agradeço a minha Vó Josefa da Paz, que me fez refletir como temos que ser pacientes e legais com o próximo.

Agradeço a meus irmãos Davi, Rafael e Rayssa, por terem me mostrado o grande significado do companheirismo, mostrando que sempre temos que ver cada coisa como um todo.

Agradeço ao Professor Andrei Formiga por ter me orientado neste trabalho, sempre me desafiando e dando conselhos para melhorar e fazer cada vez um trabalho melhor.

Agradeço a todos os professores e funcionários da UFPB, por estarem presentes em todos os momentos que foram precisos, pelos conselhos fornecidos, dúvidas esclarecidas e lições aplicadas.

Agradeço a todos as pessoas que não lembrei agora, mas que de qualquer forma fizeram parte dessa grande momento.

Glossário

AR Aprendizagem por Reforço

DT Diferença Temporal

MC Monte Carlo

PD Programação Dinâmica

PDM Processos de Decisão de *Markov*

PDMPO Processo de Decisão de *Markov* Parcialmente Observável

SARSA *State-Action-Reward-State-Action*

USAF *United States Air Force*

VDBE *Value-Difference Based Exploration*

TRE *Taxa de redução de epsilon*

Conteúdo

Lista de Figuras

Lista de Tabelas

List of Algorithms

Capítulo 1

Introdução

O aprendizado esta ligado diretamente com a inteligência, pois se um sistema é capaz de aprender a realizar determinada tarefa merece então ser chamado de inteligente [Coppin 2010]. O aprendizado de máquina nada mais é que um aprendizado por experiência, onde conforme a tarefa é executada, o programa aprende a melhor maneira de resolver o problema [Fernandes 2003].

De acordo com [Russell e Norvig 2009], o campo de aprendizado de máquina costuma distinguir três subcampos, são eles: o supervisionado, o não-supervisionado e o por reforço. O foco deste trabalho de dissertação está na aprendizagem por reforço.

A aprendizagem por reforço (AR) é definida como um problema de aprendizagem, onde um sistema inteligente sabe através de recompensas ou reforços se as ações que está realizando no ambiente são boas ou ruins. Neste tipo de aprendizagem o sistema age independente de exemplos externos. Essa definição pode ser exemplificada nos jogos de xadrez, onde o reforço é recebido apenas no final do jogo, ou em outras situações em que o reforço é fornecido com mais frequência, como no jogo de *ping e pong*, onde a recompensa pode ser fornecida a cada ponto ganho ou perdido [Russell e Norvig 2009].

O processo de aprendizagem por reforço pode ser caracterizado como um processo de decisão sequencial, onde as sequências de transições entre os estados e o recebimento de recompensas associadas a cada transição determinam o resultado final. As transições de estados em um ambiente de aprendizagem por reforço seguem a regra do Processo de Decisão de Markov (PDM), ou seja, a probabilidade de transição de um estado no ambiente, depende exclusivamente da ação realizada e do estado atual que o agente se encontra, e não

da sequência dos estados previamente visitados. Criando assim uma total independência em relação de qualquer ação e estados executados anteriormente.

Em alguns problemas modelados com o PDM, podem existir o conceito de política comportamental, ou seja, o conjunto de ações escolhidas e realizadas ao longo da execução do experimento. Assim, a aprendizagem por reforço possui como objetivo, usar as recompensas observadas para aprender e definir a política ótima, no ambiente. [Russell e Norvig 2009] [Sutton e Barto 1998] [Szepesvári 2010].

Também é importante destacar duas características dos algoritmos de aprendizagem por reforço. Uma é a *exploração* do espaço de estados do ambiente, com o objetivo de aquisição de mais experiência e a outra é o *aproveitamento* do conhecimento obtido através dessa mesma experiência, com o intuito de maximizar a recompensa e dessa forma encontrar a política ótima.

Esse trabalho dissertação propõe o desenvolvimento de uma biblioteca chamada *AILibrary-RL* e uma análise comparativa entre importantes algoritmos de aprendizagem por reforço. A *AILibrary-RL* é uma biblioteca que possui o objetivo de facilitar, modularizar e promover a reusabilidade de código, para a implementação de sistemas que possuem esse tipo de problemática. Antes de seu desenvolvimento, foi realizado um levantamento bibliográfico dos principais métodos que solucionam a problemática de AR, visando uma análise comparativa entre eles, a partir de métricas previamente selecionadas. O objetivo dessa análise comparativa é gerar estatísticas funcionais que exibam vantagens e desvantagens dos algoritmos em ambientes e condições especificas, para ajudar o usuário a escolher qual algoritmo e valores de parâmetros utilizar na implementação de um novo experimento.

1.1 Motivação

A aprendizagem por reforço pode ser considerada um paradigma computacional de aprendizagem, onde baseado no conceito de reforço o agente inteligente possui como objetivo maximizar uma medida de desempenho do ambiente para alcançar o objetivo final.

Existem vários métodos que fornecem a solução para o problema da aprendizagem por reforço. Os mais utilizados são a programação dinâmica (PD), o método de monte carlo (MC) e a diferença temporal (DT). No entanto, entre os algoritmos desses três métodos, os

que mais se destacam, fazem parte do método de DT, que junta as melhores características dos métodos de programação dinâmica e método de monte carlo [Sutton e Barto 1998] [Szepesvári 2010]. Entre os algoritmos de DT, dois merecem maior destaque, são os precursores: o Q-Learning [Watkins 1989] e o SARSA [Rummery e Niranjan 1994].

No entanto, ao decorrer das ultimas duas décadas, apareceram diversas variações dos algoritmos bases para melhorar o desempenho da aprendizagem existente, sempre buscando otimizar a estratégia de aprendizagem da política ótima, para que o algoritmo ache cada vez mais rápido o objetivo. Entre elas estão as técnicas de *eligibility traces*, *replacing traces*, *Dyna-Q*, *Dyna-H* e mecanismos hierárquicos.

Então, com o crescente número de variação de algoritmos, a comunidade acadêmica começou a trabalhar na tentativa de encontrar e normalizar novas formas de analisar e comparar os diversos tipos de algoritmos. Pode-se ter como referência: [Dutech et al. 2005], [Neruda e Slusný March, 2009], [Pessoa 2011] e [Viet, Kyaw e Chung 2011]. Também, com inicio da comparação dos algoritmos, o estudo e desenvolvimento de bibliotecas que modularizassem, facilitassem e fornecem a reutilização de código para esse tipo de problemática se desenvolveu. Como referência, pode-se citar: Pybrain [Schaul et al. 2010], a SkyAI [Yamaguchi 2011], a RL-Glue [Tanner e White 2009], a RLToolbox [Neumann 2005] e a RLToolkit [Sutton 2011]. Sendo estas as principais motivações deste trabalho de dissertação.

1.2 Objetivo Geral

Modelar e implementar uma biblioteca que forneça facilidade, organização e reutilização de código, na criação de agentes, ambientes e experimentos relacionados a aprendizagem por reforço, chamada *AILibrary-RL*. Para ajudar o usuário a escolher qual algoritmo e valores dos parâmetros utilizar em um novo experimento, realizar uma análise comparativa a partir de métricas previamente selecionadas, entre os principais métodos que solucionam a problemática de AR. Com o objetivo de gerar estatísticas funcionais que exibam suas vantagens e desvantagens em encontrar a política ótima, com ambientes e condições especificas.

1.2.1 Objetivo Específico

- Levantamento bibliográfico e estudo dos principais métodos de AR;

- Escolha de ambientes com características episódicas e contínuas;
- Escolha de métricas compatíveis com cada tipo de ambiente;
- Analisar os parâmetros dos algoritmos de diferença temporal, com o objetivo de verificar quais melhores valores atribuir para os algoritmos atingirem a convergência no ambiente.
- Comparar métodos de aprendizagem por reforço, a partir de métricas previamente definidas, com o objetivo inferir conclusões sobre qual algoritmo obteve melhor desempenho para encontrar a política ótima;
- Validar os resultados obtidos na comparação, através da análise quantitativa e experimental dos dados.
- Avaliar o uso da biblioteca, usando-a em novas implementações.

1.3 Organização da Dissertação

O trabalho encontra-se organizado da seguinte maneira:

- Capítulo 2 ← Apresenta a fundamentação teórica do trabalho, onde está contido os conceitos básicos sobre agentes e ambientes e esclarece como a aprendizagem por reforço é caracterizada;
- Capítulo 3 ← Apresenta a teoria dos mecanismos escolhidos para a análise e implementação na biblioteca;
- Capítulo 4 ← Descreve o funcionamento da biblioteca *AILibrary-RL*, junto com sua arquitetura;
- Capítulo 5 ← Apresenta os ambientes escolhidos para a análise dos algoritmos; apresenta as ferramentas utilizadas para o desenvolvimento dos sistemas e bibliotecas; exibe os resultados dos desempenhos dos algoritmos nos ambientes escolhidos;
- Capítulo 6 ← Descreve os trabalhos relacionados com a ideia proposta nesta dissertação, elencando suas vantagens e desvantagens;

- Capítulo 7 ← Estão as conclusões obtidas dos resultados apresentados, a descrição dos trabalhos futuros e as considerações finais obtidas no desenvolvimento do estudo.

Capítulo 2

Fundamentação

A maioria dos termos técnicos mencionados na introdução e nos capítulos a seguir, irão necessitar de uma fundamentação teórica e uma definição mais formal. Portanto, as seções a seguir terão o objetivo de elucidar esses conceitos.

2.1 Conceitos de Agentes e Ambientes

2.1.1 Agentes

Todo aquele que consegue interagir com o ambiente a sua volta a partir de sensores e atuadores é considerado um agente. Como exemplo é colocado o agente humano que interage com o ambiente através dos olhos e ouvidos, esses representados como sensores e a mão, pernas e boca, representados como atuadores [Russell e Norvig 2009].

Segundo [Russell e Norvig 2009], qualquer sistema que apresenta as características abaixo pode ser considerado um agente:

- Autonomia
- Habilidade Social
- Reatividade
- Pró-Atividade

A autonomia é o fato do agente não necessitar da intervenção humana, ou seja, ele se baseia nos conhecimentos obtidos do ambiente. A habilidade social é caracterizada pela capacidade de comunicação com outros agentes através de uma linguagem comum. A reatividade é a capacidade de adaptação as mudanças ocorridas no ambiente. Já a pró-atividade é a manifestação de uma iniciativa por parte do agente em resolver determinado objetivo [Rehm et al. 2009].

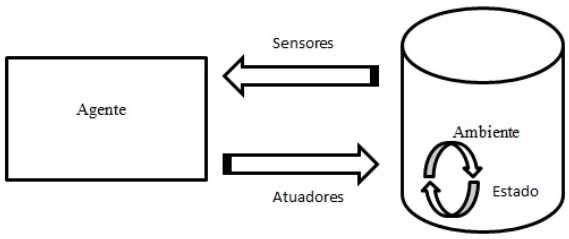

Figura 2.1: Interação agente com ambiente.

Como observado na Figura 2.1, o agente recebe informações do ambiente através de sensores e interage com o mesmo através de atuadores. A interação que ocorre entre o agente e o ambiente é chamada de ciclos de percepção ação e geralmente ocorre vários ciclos durante a execução do sistema. O termo percepção é usado para referenciar as entradas perceptivas através dos sensores, no entanto o agente também pode obter uma sequência de percepções que está atribuída a história completa de tudo que o agente já percebeu [Russell e Norvig 2009]. Para que a sequência de percepções seja utilizada é necessário que uma função de agente seja implementada com o objetivo de mapea-la em uma ação para o agente.

Os agentes podem ser divididos em dois grupos: os reativos, caracterizados como aqueles que utilizam os dados atuais do ambiente para alcançar algum objetivo, ou seja, não possuem memória ou meios de armazenamento e os cognitivos, onde ao decorrer do tempo no ambiente, o agente vai aprendendo e adquirindo experiência, criando assim, um sistema mais inteligente. No entanto, é possível definir um agente como híbrido. Sendo esse uma mesclagem dos dois tipos citados anteriormente. O seu objetivo é solucionar o problema de ações mais rápidas para os agentes cognitivos e prover ao agente reativo a capacidade de raciocínio e planejamento, no momento em que o ambiente diverge muito das condições iniciais [Rehm et al. 2009].

Com esses conceitos pode-se observar que para um agente inteligente possuir uma boa

interatividade é necessário, dependendo do sistema ou ambiente, que algumas dessas características estejam presentes.

2.1.2 Ambientes

O ambiente pode ser definido como um conjunto de propriedades, que possui o objetivo de influenciar diretamente nas ações do agente [Rehm et al. 2009]. Uma característica importante do ambiente é seu estado, ele é essencial para o desenvolvimento de certos tipos de agentes. Abaixo, encontra-se algumas descrições de ambientes.

Tipos de Ambiente

Antes de começar a entender como o agente deve se comportar no ambiente, é importante saber em qual ambiente o agente está inserido.

Completamente Observável e Parcialmente Observável

- Completamente Observável: O ambiente é completamente observável, quando o agente pode perceber todo o estado do ambiente, ou seja, o agente pode tomar a melhor decisão em qualquer ponto do ambiente.
- Parcialmente Observável: O ambiente parcialmente observável é exatamente o oposto do completamente observável. O agente não conhece todo o estado do ambiente, com isso a tomada de decisão se torna mais complexa. [Russell e Norvig 2009].

Determinísticos e Estocásticos

- Determinístico: Se o próximo estado do ambiente é completamente determinado pelo estado atual e ações executadas pelo agente.
- Estocástico: O Ambiente é estocástico se as ações do agente geram situação aleatórias no ambiente, ou seja, não se pode determinar com máxima certeza se uma determinada ação irá gerar o efeito desejado no ambiente [Russell e Norvig 2009].

Discreto e Contínuo

- Discreto: O ambiente é discreto se a quantidade de ações que o agente pode realizar e coisas que ele pode perceber são finitas.

- Contínuo: O ambiente é continuo se a quantidade de ações que o agente pode realizar e coisas que ele pode perceber são infinitas [Russell e Norvig 2009].

Benigno e Antagonistas

- Benigno: O ambiente é benigno se o próprio ambiente não possuir nenhum objetivo que contraste com o objetivo do agente, ou seja, o ambiente não tenta impedir as ações do agente, deixando que o mesmo alcance seu objetivo.

- Antagonistas: O ambiente é antagonista quando tenta de alguma maneira atrapalhar o objetivo do agente [Russell e Norvig 2009].

Observar esses tipos de ambiente é muito importante, pois quando a implementação do agente for pensada, as características do ambiente serão essenciais para o desempenho do mesmo.

2.1.3 Estrutura de Agentes

O objetivo da inteligência artificial é projetar o programa de agente. Nesse programa estará implementado a função de agente, que como descrita anteriormente serve para mapear as percepções de entrada em ações para os atuadores. Dependendo de onde o programa for executado, o dispositivo de computação terá seus próprios sensores e atuadores. Esse conjunto é definido como arquitetura. Com isso, chega-se a conclusão que um agente é o programa de agente mais a arquitetura que o mesmo irá ser executado [Russell e Norvig 2009].

Os programas de agentes podem ser implementados de diversas maneiras, no entanto a literatura fornece quatro tipos básicos que incorporam os princípios discutidos a maioria dos sistemas inteligentes, são eles:

- **Agentes reativos simples:** é o tipo de programa mais simples. O seu objetivo é tomar ações imediatas, ou seja, selecionar uma ação tento como base a percepção atual, ignorando completamente o restante do histórico de percepções;

- **Agentes reativos baseados em modelos:** Esses tipos de agentes possuem mais facilidade de se adaptar ao ambiente, pois não tomam ações imediatas. Em vez disso

possuem um estado interno que armazena as percepções, criando assim uma sequência de percepções. Com isso o agente pode refletir alguns aspectos que ainda não foram visualizados no estado do ambiente;

- **Agentes baseados em objetivos:** é o tipo de programa que possui um objetivo a ser alcançado no ambiente em que está inserido, esses objetivos descrevem as situações desejáveis do agente. Essa estratégia pode ser combinada com os resultados obtidos no estado interno de sequência de ações dos agentes reativos, a fim que se possa escolher ações que alcancem os objetivos finais;

- **Agentes baseados em utilidade:** esse tipo de agente não visa apenas o objetivo do agente, mas também mede o desempenho das várias maneiras de se chegar a ele. Podem existir varias sequências de ações que levam o agente ao seu destino, no entanto, algumas são mais rápidas, mais seguras, mais confiáveis e tudo isso pode alterar a maneira de como o mesmo chega ao objetivo final.
  Com isso é necessário que esse tipo de agente implemente uma função de utilidade que mapeie um estado ou sequência de estados em um número que descreva qual grau de felicidade está associado a ele. Tendo esse grau de felicidade o objetivo de verificar qual estado do ambiente, terá maior utilidade para o agente.

Pode-se dizer que os programas de agente básicos recebem percepções dos sensores e decidem como seus atuadores irão atuar. No entanto, esses tipos básicos podem ser convertidos em agentes com aprendizado. Esse tipo de agente pode operar em ambientes inicialmente desconhecidos e se tornar mais competente ao passar de sua execução.

Agentes de aprendizagem podem ser divididos em quatro módulos conceituais, são eles:

- **Elemento de Aprendizagem:** esse módulo é responsável por decidir qual o próximo passo do agente;

- **Elemento de Desempenho:** responsável pela seleção de ações externas. Os programas básicos de agentes são exatamente esse módulo do agente de aprendizagem. O objetivo desse componente é receber as percepções e decidir sobre as ações;

- **Crítico:** o trabalho do crítico é auxiliar o elemento de aprendizagem para verificar se o agente está funcionando de acordo com um padrão fixo de desempenho. Ele

pode intervir no elemento de desempenho para fornecer um melhor desempenho para o agente no futuro. O crítico é muito importante em um agente de aprendizado, pois, somente as próprias percepções do agente não fornecem indicações de sucesso;

- **Gerador de Problemas:** esse módulo é responsável por sugerir ações que levem o agente a novas experiências.

Pode-se concluir que o agente de aprendizagem, possui um módulo onde está incorporado o programa de agente básico, sendo esse módulo o de elemento de aprendizagem. Podendo este, ser alterado pelo programa de agente que melhor satisfaça o problema a ser encontrado no ambiente. Também é observado que todos os agentes podem melhorar seu desempenho através da aprendizagem.

Esse trabalho possui um foco nos agentes baseados em aprendizagem. Tendo como o elemento de desempenho o programa de agente baseado em utilidade.

2.2 Formas de Aprendizagem

> *Aprendizado e inteligência estão intimamente relacionados um ao outro. Há geralmente a concordância de que um sistema capaz de aprender mereça ser chamado de inteligente; e, reciprocamente, de um sistema considerado como inteligente espera-se, geralmente, entre outras coisas, que seja capaz de aprender. Aprendizado sempre tem a ver com auto aprimoramento do comportamento futuro, baseado em experiência passada.*(COPIN, 2010 pg. 233)

Como descrito na seção anterior, podem ser atribuídos no módulo de desempenho do agente de aprendizado, vários componentes de programas de agentes, dependendo de qual a necessidade de resolução do problema. No entanto, existem três aspectos importante para o projeto de agente de aprendizado, são eles:

- A escolha do elemento de desempenho que deve ser utilizado;
- A realimentação usada para auxiliar os componentes de desempenho;
- A Existência de conhecimento a priori.

Os Elementos de desempenho já foram descritos na seção de estruturas de agentes. Na opção de realimentação usada pelo agente ou técnica de aprendizagem, existem três métodos, são eles: a aprendizagem supervisionada, aprendizagem não supervisionada e a aprendizagem por reforço, sendo mais detalhadas abaixo, segundo [Russell e Norvig 2009] e o conhecimento a priore diz respeito ao que o agente está tentando aprender.

- **Aprendizagem Supervisionada:** A aprendizagem supervisionada, como o próprio nome sugere, faz o agente necessitar de uma supervisão de dados externos, mais especificamente um instrutor. A partir desses dados o agente aprende uma função que mapeia os valores de entrada para os de saída. Durante o treinamento o instrutor fornece ao agente os dados corretos para que o objetivo seja alcançado, assim, esse método se torna muito eficiente, pois o sistema pode trabalhar diretamente com informações corretas. Se os valores forem discretos o problema é categorizado como classificação. No entanto, se os valores forem contínuos o problema é categorizado como regressão. Esse método é muito utilizado em agentes para classificação, regressão e estimação de probabilidade condicional.

- **Aprendizagem Não-Supervisionada:** Neste tipo de aprendizagem, não são fornecidas saídas para as entradas das funções de aprendizado do agente, com isso existe incerteza sobre a saída da função de aprendizagem do agente. Possuindo essa ausência dos parâmetros de entrada, este tipo de aprendizagem usado puramente, não fornece condições do agente decidir qual é uma ação adequada ou estado desejado. Assim, são utilizados muitos métodos probabilísticos para simular essa experiência não vivida. Um deles é a aprendizagem *bayesiana* ou redes *bayesianas*.

- **Aprendizagem por Reforço:** Nesse método em vez do agente ser informado sobre o que fazer por um instrutor, o agente de aprendizagem por reforço deve aprender a partir de reforços fornecidos em cima de suas ações realizadas no ambiente e com isso fazer uma hipótese e determinar se essa hipótese foi boa ou ruim, pois os reforços podem significar recompensa ou punição.

A realimentação utilizada neste trabalho é a por aprendizagem por reforço.

2.3 Aprendizagem por Reforço

2.3.1 Definições Formais

Processo de Decisão Sequencial

De acordo com [Puterman 2005], um processo de decisão sequencial pode ser simbolicamente representado de acordo com a Figura 2.2.

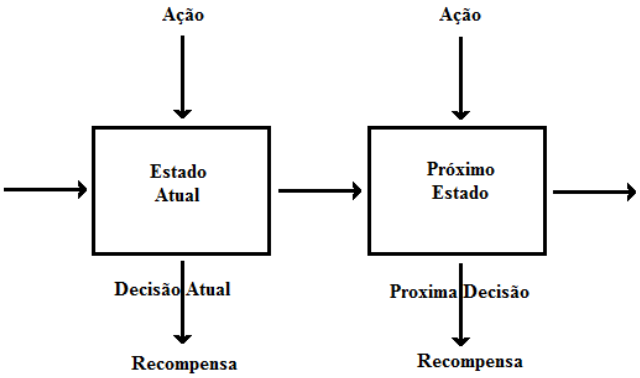

Figura 2.2: Representação simbólica de um processo de decisão sequencial.
(Tradução Livro [Puterman 2005])

Como observado na Figura 2.2, em determinado momento de um sistema computacional, o agente ou controlador se encontra em determinado estado de um ambiente. Baseado nesse estado o agente realiza uma ação que pode produzir dois resultados: uma recompensa positiva ou negativa. Depois, de acordo com a ação tomada, o agente realiza a função de transição de estados e segue para um novo estado. Nesse novo estado o agente enfrenta o mesmo problema descrito anteriormente, no entanto, com novas ações e recompensas.

Um processo de decisão sequencial pode ser definido pelos componentes que fazem parte da quíntupla M = {T,S,A,R,P}. Abaixo encontra-se uma definição mais formal dos elementos:

- **T: Conjunto de Períodos de Decisão:** As decisões de um processo sequencial de decisão são tomadas em períodos de tempo, aqui denominados de períodos de decisão. Esse conceito é representado pelo conjunto discreto de períodos de decisão T = {1,2,...,N} $N \leq \infty$, que pode ser finito ou infinito. Se o tempo relacionado ao problema for discreto, o mesmo é dividido em episódios, e a cada período $t \in T$ equivale

ao início de um episódio e ao momento especifico da tomada de decisão [Puterman 2005].

- **S: Conjunto de Estados do Ambiente:** O processo sequencial assume em cada período de decisão *t* um estado $s \in S$, com isso o agente pode obter informações do ambiente. O conjunto *S* não varia com *t* e pode ser discreto, continuo, finito e infinito.

- **A: Conjunto de Ações Possíveis:** Em cada estado $s \in S$, é escolhido pelo agente uma ação *a* de um conjunto de ações *A(s)* disponíveis no estado *s*. O conjunto *A(s)* não varia com o tempo *t* e pode ser discreto, continuo, finito e infinito.

- **R: Função de Recompensa:** Escolhendo uma ação $a \in A(s)$ a partir do estado *s* no período de decisão *t* o agente recebe uma recompensa $r_t(s,a)$, alterando sua percepção e consequentemente levando-o a um novo estado s_{t+1}, que o conduzirá ao novo instante de decisão *t+1*. Essa função $r_t(s,a)$ representa o valor da recompensa no período de decisão *t* no estado $s \in S$ e $a \in A(s)$. A recompensa pode ser positiva, fazendo com que o agente avance mais rápido para alcançar seu objetivo ou negativa, punindo o agente por algo errado que está fazendo.

- **P: Função de Probabilidade de Transição:** No próximo período de decisão *t+1*, o estado do ambiente é definido pela distribuição de probabilidade $p_t(.|s,a)\ \forall t$. A função de probabilidade de transição $p_t(j|s,a)$, define a probabilidade do processo no estado $j \in S$ no instante *t+1*, tendo o agente escolhido a ação $a \in A(s)$, no estado *s* e tempo *t*.

Em cada período de decisão *t*, o agente necessita escolher uma regra de decisão $d_t(s)$, para a tomada de uma ação especifica. Para isso, uma política π é utilizada pelo agente, para que o mesmo defina uma estratégia de escolha de ações em qualquer estado, ou seja, uma política $\pi = \{d_1, d_2, ..., d_n\}, N \leq \infty$, é um conjunto de regras de decisão. Um problema de decisão sequencial é resolvido quando, uma política ótima π^* que otimize a sequência de retornos a ser obtidos pelo agente é determinada [Puterman 2005].

Na maioria das vezes em um processo sequencial, a escolha de uma ação tomada em um tempo *t+1* qualquer, dependerá do histórico de ações escolhidas anteriormente a *t+1*. Quando isso acontece a dinâmica de comportamento entre o agente e o ambiente pode ser

definida através da expressão de distribuição de probabilidade 2.1 abaixo:

$$Pr\{s_{t+1} = s^{'}\ ,\ r_{t+1} = r | s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, ..., r_1, s_0, a_0\} \tag{2.1}$$

onde *Pr* representa a probabilidade do próximo estado s_{t+1}, dependendo de todos os estados, ações e reforços passados, $s_t, a_t, r_t, ..., r_1, s_0, a_0$.

Processo de Decisão de Markov (PDM)

Um processo de decisão markoviano (PDM) é um processo de decisão sequencial. Entretanto, a probabilidade de transição depende exclusivamente do estado e ação atual e não dos estados e ações visitados previamente. Essa descrição caracteriza a propriedade de *markov* e pode ser definida através da expressão 2.2:

$$P^{a}_{s,s'} = Pr\{s_{t+1} = s' | s_t = s, a_t = a\} \tag{2.2}$$

Essa propriedade permite que as ações de um agente sejam tomadas em função apenas do estado atual, isso é muito importante para a aprendizagem por reforço, pois a escolha de um novo estado pelo agente, apenas depende do estado e ação escolhida no presente. Os estados de um processo de decisão de *markov* pode ser finito ou infinito.

Os PDMs são muito utilizados em ambientes completamente observáveis, onde o agente conhece sua própria localização no ambiente. No entanto, para que as ações do agente possam parecer mais com a realidade é necessário que o ambiente seja parcialmente observável, para isso existe uma variante do PDM chamada Processo de Decisão de Markov Parcialmente Observável (PDMPO).

Processo de Decisão de Markov Parcialmente Observável (PDMPO)

Geralmente problemas que envolvem PDMs possuem ambientes completamente observáveis. Assim, o agente sempre sabe em qual estado se encontra. Isso combinado com a hipótese de *markov* para o modelo de transição de estados, deixa a política ótima π^* dependente apenas do estado atual.

Entretanto, quando o ambiente é parcialmente observável, o agente não sabe em que estado s_t está e qual ação *a* deve ser realizada. Assim, se torna mais difícil executar a política ótima π^*. Além disso a utilidade ou função valor de um estado *s* e a ação ótima à ser realizada em *s* não dependem apenas do estado em si, como ocorre nos PDMs, mas também da quantidade de informação que o agente sabe sobre o próprio estado *s*. Com esses conceitos pode-se dizer que os PDMPOs são muito mais difíceis que os PDMs. No entanto, não se pode evitá-los, pois o mundo real é muito parecido com um PDMPO. [Russell e Norvig 2009].

Nos algoritmos de aprendizagem por reforço em ambientes parcialmente observáveis, muitas vezes não é conhecido a função de recompensa e a função de transição $p_t(j|s,a)$, então, na maioria das vezes é necessário aprendê-las. Na Tabela 2.1, está listado alguns agentes modelados para esses ambientes:

Agente	**O que se sabe sobre o agente**	**O que se deseja aprender**	**O que é usado**
Baseado em Utilidade	A função de transição p(s'ls,a)	A função de recompensa R(s)	U(s)
Q-Learning		A função Q(s,a)	Q
Agente Reflexivo		A política $\pi(s)$	π

Tabela 2.1: Tabela de agentes para aprendizagem por reforço

Como pode ser visualizado na Tabela 2.1, os agentes mais utilizados na aprendizagem por reforço são, o baseado em utilidade ou função valor, o *q-learning* e o agente reflexivo.

- **Baseado em Utilidade (Função Valor):** O agente baseado em utilidade, muitas vezes já conhece o modelo de transição p_t do mundo em que se encontra. No entanto não conhece a função de recompensa *R(s)*. Nesse caso é necessário aprender *R(s)*. Depois que R(s) é descoberto ele pode ser usado juntamente com p_t para encontrar a função utilidade para o estado e com isso usa-la, assim como feito no processo de decisão de *markov*.

- **Q-learning:** O agente *Q-learning* não precisa aprender a função de transição p_t nem a função de recompensa *R(s)*. No entanto, ele precisa aprender uma função de valor

estado-ação, chamada *Q(s,a)*, onde *s* é o estado atual e *a* é a ação a ser realizada. Essa função é muito parecida com a utilidade de um estado. No entanto uma utilidade de estado e ação. O agente *Q-Learning* também não necessita aprender sobre a política ótima π^* para cada estado, ele pode usar a função *Q(s,a)* diretamente.

- **Agente Reflexo:** O agente reflexo também não precisa aprender sobre a função de transição e recompensa. No entanto ele aprende a função π do estado *s* diretamente. A resposta desse agente em questão é puramente baseada em estímulos. Em cada estado é realizada uma determinada ação, não é necessário pensar sobre o modelo do mundo.

Entendendo o Problema de Aprendizagem por Reforço

Segundo [Sutton e Barto 1998], quatro elementos principais podem ser identificados em um problema de aprendizagem por reforço, são eles: uma política, um modelo de ambiente, uma função de recompensa e uma função valor ou utilidade para cada estado.

- **Política:** A política é denotada por π_t e é responsável por definir em um estado, qual a probabilidade de selecionar cada possível ação encontrada no mesmo. Isso é realizado através do mapeamento de estados *s* e ações *a*, em um valor $\pi(s, a)$ que é relativo a probabilidade do agente escolher a ação $a \in A(s_t)$, quando o mesmo está no estado $s \in S$.

- **Modelo do Ambiente:** Na aprendizagem por reforço, o modelo de ambiente em que o agente está inserido é considerado um processo de decisão de markov parcialmente observável (PDMPO). Onde dado um estado s_t e ação a_t o PDMPO possibilita a predição do próximo estado $s_t + 1$, pois se é necessário que a IA fique parecida com o comportamento humano e na vida real nem sempre é possível a construção de um modelo completo do ambiente.

- **Função de Recompensa:** Cada ação a_t que o agente realiza no ambiente, gera uma consequência. Com isso o agente recebe uma recompensa $r_{t+1} \in \Re$ e um novo estado, s_{t+1} é obtido. Este reforço é baseado em uma função que representa o objetivo final do agente.

Durante o processo de execução do sistema, o objetivo do agente aprendiz é maximizar a quantidade de reforços recebidos, ou seja, maximizar o valor de retorno durante todo o processo e não apenas maximizar o reforço imediato a receber. Se o problema for discreto, o agente pode usar, no caso mais simples a função de sequência de reforços recebidos que pode ser expressada pela expressão 2.3.

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + ... + r_T \tag{2.3}$$

onde *T* é o tempo final. Quando o agente alcança o objetivo no ambiente em que está integrado e a interação se rompe, isso significa que um episódio de interação foi finalizado. Quando um episódio termina o ambiente é reinicializado para o estado inicial. No entanto, a interação do agente com o ambiente pode não terminar em um episódio e continuar infinitamente. Nestes casos $T \leftarrow \infty$, com isso o valor de retorno tenderá a infinito. Para esse caso, se faz necessário acrescentar à função de sequência de reforços uma variável de fator de desconto, que tem como objetivo amortizar os valores futuros, controlando a influência que esses valores possuem sobre o retorno total esperado, definida pela expressão 2.4.

$$R_t = r_{t+1} + \gamma.r_{t+2} + \gamma^2.r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k.r_{t+k+1} \tag{2.4}$$

onde $0 \leq \gamma \geq 1$. Se $\gamma = 0$, o agente passa a ter uma visão míope dos reforços, maximizando apenas reforços imediatos. No entanto, $\gamma = 1$, iguala a importância para reforços imediatos e futuros.

- **Função Valor:** O objetivo da função valor é associar um valor a um estado ou par de estado-ação. Essa função se baseia nas recompensas esperadas pelo agente ao longo do processo de aprendizagem, possuindo como ponto inicial *s*. A função valor que considera apenas o estado *s* é representada por *V(s)* e é denominada função valor-estado. Já a função valor que considera o estado-ação é representada como *Q(s,a)* e denominada função valor estado-ação. A quantidade de reforços que o agente espera receber no futuro depende de quais ações serão escolhidas, a partir disso, a função valor é definida em relação a uma política.

Considerando uma política π, uma decisão em cada estado $s \in S$ e ação $a \in A(s)$, com a probabilidade $\pi(s, a)$ para realizar a ação *a* em determinado estado *s*. O valor do estado *s* para a política π é representada por $V^{\pi}(s)$ e pode ser denotado pela expressão 2.5:

$$V^{\pi}(s) = E_{\pi} R_t | s_t = s = E_{\pi} \sum_{k=0}^{\infty} \gamma^k r_t + k + 1 | s_t = s \quad (2.5)$$

onde E_{π} é o valor esperado, levando em conta que o agente seguiu a política π em qualquer instante de tempo *t*. Uma política que maximiza o valor esperado é considerada uma política ótima e pode ser representada pela fórmula 2.6:

$$V^{*}(s) = max_{\pi} V^{\pi}(s) \quad (2.6)$$

O mesmo método é aplicado para a função valor estado-ação *Q(s,a)*. Que é representada por $Q^{\pi}(s, a)$ e pode ser denotada pela função 2.7:

$$Q(s, a)^{\pi} = E_{\pi} R_t | s_t = s, a_t = a = E_{\pi} \sum_{k=0}^{\infty} \gamma^k r_t + k + 1 | s_t = s, a_t = a \quad (2.7)$$

no entanto, agora o valor não é mais atribuído apenas ao estado s_t e sim ao estado e ação a_t, caracterizando o comportamento do agente pela política π. Da mesma forma com a função valor estado, o valor estado ação ótima pode ser definido pela expressão 2.8:

$$Q(s, a)^{*}(s) = max_{\pi} Q^{\pi}(s, a) \quad (2.8)$$

Pode-se resumir um problema de aprendizagem por reforço como um processo interativo entre o agente e o ambiente. Essa interação ocorre em uma sequência de passos de tempo discretos *t* = 0,1,2,3,...n. Em cada etapa de tempo *t*, o agente interage com uma representação do ambiente (um estado) $s_t \in S$ onde *S* é o conjunto de possíveis estados do ambiente. Nesse estado s_t o agente seleciona uma ação $a_t \in A(s_t)$, onde $A(s_t)$, é o conjunto de ações possíveis no estado presente s_t. Em consequência da ação realizada, o agente recebe uma recompensa numérica r_{t+1}, referente a mudança de estado do ambiente. Na Figura 2.3 abaixo pode ser observado como essa interação ocorre:

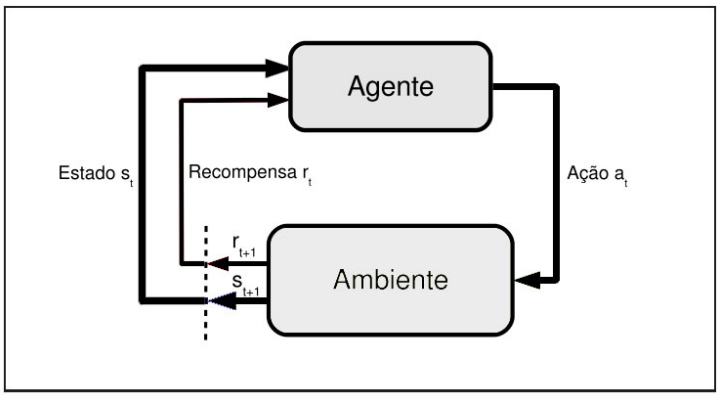

Figura 2.3: Representação da interação entre agente e ambiente em um problema de aprendizagem por reforço.

(Tradução [Sutton e Barto 1998])

É observado na Figura 2.3, que a interação do agente com o ambiente não é muito diferente da interação mostrada no capítulo que descreve os agentes e ambientes. Com essa interação o agente pode definir, após um determinado número de experimentação, qual a melhor ação à ser escolhida em cada estado. Assim, o agente aprende qual a política ótima de atuação que maximize a estimativa de retorno total esperado para alcançar o objetivo, independente do estado inicial.

2.3.2 Métodos para Solucionar o Problema de Aprendizagem por Reforço

De acordo com [Sutton e Barto 1998], os principais métodos para solucionar problemas de aprendizagem por reforço são, a Programação Dinâmica (PD), o Método de Monte Carlo (MC) e a Diferença Temporal. Abaixo, encontra-se uma descrição mais detalhadas desses métodos na visão de solução para a aprendizagem por reforço, pois é sabido que a PD e MC também podem ser usados para solucionar outros tipos problemas.

Programação Dinâmica

A programação dinâmica na aprendizagem por reforço é representada por uma coleção de algoritmos que possuem o objetivo de calcular a política ótima π^* de um ambiente, dado que o mesmo seja completamente observável como um PDM e finito. Como descrito anterior-

mente, um ambiente é completamente observável quando o agente possui o conhecimento total dos estados, ações, retornos e probabilidades de transição. Assim, se faz necessário o conhecimento das expressões 2.9 2.10:

$$P^a_{s,s'} = Prs_{t+1} = s'|s_t = s, a_t = a \quad (2.9)$$

$$R^a_{s,s'} = Er_{t+1}|s_{t+1} = s', s_t = s, a_t = a \quad (2.10)$$

para todos os $s \in S$, $a \in A(s)$ e $s' \in S^+$, sendo S^+ o estado terminal ou objetivo do agente em um problema episódico. No entanto, o método PD também pode solucionar alguns casos especiais de problemas contínuos.

Algoritmos clássicos de PD possuem um grande limite ao serem utilizados em problemas de aprendizagem por reforço, pois necessitam de um modelo de ambiente completamente observável e um grande custo computacional. No entanto, ainda são muito importantes no quesito de aprendizado teórico. A ideia chave para solucionar o problema de aprendizagem por reforço e dos algoritmos PD é estruturar a procura por políticas ótimas e organizar o ambiente em funções de valor [Sutton e Barto 1998]. Com isso, os métodos PD partem do principio que as políticas ótimas podem ser obtidas a partir da função valor V^* , Q^* as quais devem satisfazer as equações de otimalidade de Bellman, denotadas pelas expressões 2.11 2.12, para a função-valor e 2.13 2.14, para o estado-ação:

Função Valor-Estado

$$V^*(s) = max_a E\{r_{t+1} + \gamma V^*(s_{t+1})|s_t = s, a_t = a\} \quad (2.11)$$

$$= max_a \sum'_s P^a_{s,s'} R^a_{s,s'} + \gamma V^*(s') \quad (2.12)$$

Função Estado-Ação

$$Q^*(s, a) = E\{r_{t+1} + \gamma Q^*(s_{t+1}, a')|s_t = s, a_t = a\} \quad (2.13)$$

$$= \sum'_s P^a_{s,s'} R^a_{s,s'} + \gamma Q^*(s', a')$$

para todos os $s \in S$, $a \in A(s)$ e $s' \in S^+$.

Método de Monte Carlo

De acordo com [Sutton e Barto 1998], o objetivo dos métodos de Monte Carlo é a partir de um ambiente parcialmente observável, fazer o agente estimar a função valor do estado *s* e encontrar a política ótima do ambiente. Diferente dos algoritmos de PD, os algoritmos de MC solucionam os problemas de aprendizagem por reforço utilizando o valor esperado dos retornos totais observados. Assim, para garantir que os retornos totais estejam bem definidos e disponíveis, é assumido que a experiência adquirida do ambiente é dividia em cada conclusão de um episódio. Assume-se que os algoritmos de MC trabalham incrementalmente de episódio em episódio, não passo à passo.

Apesar de algumas diferenças entre os métodos de PD e MC, muitas ideias PD estão contidas nos métodos de MC, entretanto, os algoritmos de MC consideram apenas a experiência, ou seja, necessitam apenas de amostras de sequências de estados, ações e recompensas de interações reais ou simuladas com o ambiente, não o completo conhecimento do mesmo. É calculado a função valor $V^{\pi}(s_t)$ utilizando apenas o valor esperado dos retornos totais R_t, após a visita do estado s_t. Assim, uma boa maneira de estimar o valor de um estado a partir da experiência do agente é tirar a média dos retornos observados após a visita de cada estado, com isso, quanto mais retorno for observado a media deverá convergir para o valor esperado.

Diferença Temporal

De acordo com [Sutton e Barto 1998], os métodos de Diferença Temporal (DT) é uma combinação entre os métodos de Monte Carlo e Programação Dinâmica. Com essas características, esses métodos de diferença temporal podem aprender diretamente com a experiência do ambiente, não necessitam de um modelo completamente observável do ambiente e não precisam esperar até o final de um episódio para atualizar as estimativas de cada estado, trabalhando de forma passo à passo, como é feito nos métodos de PD. Assim, eles atualizam imediatamente o valor de $V^{\pi}(s_t)$ após cada passo, como pode ser visualizado na expressão 2.15:

$$V^{\pi}(s_t) = V^{\pi}(s_t) + \alpha r_{t+1} + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t) \tag{2.15}$$

onde o parâmetro α é a taxa de aprendizagem e γ é o fator de desconto.

A taxa de aprendizagem $\alpha \in [0, 1]$ determina a velocidade com que o agente aprende sobre o ambiente. Já o fator de desconto $\gamma \in [0, 1]$ representa quão guloso o algoritmo será, sendo, quanto maior seu valor, menos guloso o algoritmo será.

A grande vantagem dos métodos de DT é que eles não trabalham de forma episódica e atualizam as estimativas da função de retorno a partir de outras estimativas já aprendidas em estados sucessivos, com isso, não é necessário a descrição precisa de modelo do ambiente, de seus reforços e das distribuições de probabilidades entre os estados, como nos métodos PD.

Os métodos de diferença temporal podem ter suas políticas π classificadas em *on-policy* e *off-policy*.

- ***On-Policy:*** Quando a mesma política é utilizada tanto para o processo de exploração do ambiente quanto para o processamento dos dados obtidos.
- ***Off-Policy:*** Quando são usadas políticas diferenciadas para o processo de exploração quanto para o processamento.

2.4 Conclusão

Este capítulo começou referenciado algumas características introdutórias da inteligência artificial, como os agentes e ambientes. Depois foram citado alguns conceitos na construção de novos agentes, como os reativos e os baseados em aprendizagem e foram descritos os métodos de aprendizagem supervisionada, não-supervisionada e por reforço, que fazem parte do aprendizado de máquina.

Também foi observado que o agente construído neste trabalho, será o baseado em aprendizagem e a aprendizagem utilizada será a por reforço. Assim, foram referidas as características mais importantes observadas na problemática de AR. Posteriormente foram descritos os métodos de Programação Dinâmica, Método de Monte Carlo e Diferença Temporal, que são os mais utilizados para solucionar esse tipo de problemática.

Agora que a maioria dos termos técnicos foram elucidados é possível entender com mais clareza o funcionamentos dos algoritmos que serão referenciados no próximo capítulo.

Capítulo 3

Mecanismos de Aprendizagem por Reforço

Analisando os algorítimos de aprendizagem por reforço é possível observar que todos eles possuem dois importantes elementos. Um é o elemento de **exploração** do ambiente, onde o agente através de métodos de exploração realiza uma escolha entre explorar estados desconhecidos ou continuar explorando estados já conhecidos. Outra é o elemento de **aprendizagem**, onde o agente através de métodos de aprendizagem processa as informações obtidas em cada estado-ação *Q(s,a)* ou valor-estado *V(s)* do ambiente para chegar no seu objetivo.

3.1 Elemento de Exploração

O elemento de exploração esta intimamente relacionado com a política π, pois a cada passo é necessário que o agente escolha uma ação no estado em que se encontra para conseguir progredir no ambiente. Sabendo que o agente possui a propriedade de *markov* e apenas realiza suas decisões a partir do seu estado atual, a decisão entre, explorar estados que ainda não foram explorados (Exploração) ou aproveitar as informações dos estados em que se encontra (Aproveitamento) e seguir em direção ao objetivo necessita ser realizada.

Esses dois conceitos são melhores explicados abaixo:

- **Aproveitamento**: nessa característica o agente explora os estados do ambiente que foram visitados, com o objetivo de melhorar e otimizar suas ações.

- **Exploração**: na exploração o agente realiza suas ações em direção aos estados do ambiente que ainda não são conhecidos, assim, o mesmo não sabe o que irá encontrar no próximo estado s_{t+1}.

Essa é uma questão que pode influenciar no desempenho do agente e pode ser solucionada através de **métodos de exploração**. Esses métodos fornecem ao agente a escolha entre, uma ação que possua maior valor em seu estado atual ou outra que o encaminhe para um estado parcialmente ou nada explorado.

A diferença entre esses conceitos são muito importantes, pois no momento da seleção da ação, o agente alterna entre essas duas ideias, com o objetivo de adquirir um conhecimento total dos estados do ambientes.

Existem vários métodos com esse objetivo e eles variam entre duas vertentes, sendo elas: fazer o agente maximizar a exploração ou o aproveitamento de informação. Na lista abaixo pode ser visualizado alguns exemplos:

- ***greedy:*** Na maior parte do tempo a ação com o maior valor de recompensa é escolhida, chamada de ação gulosa. De vez em quando, uma ação com uma pequena probabilidade ε é escolhida aleatoriamente. As ações são escolhidas de forma uniforme, independente da função estado-ação.

- ***ε-greedy:*** É similar a política *greedy*. No entanto, a melhor ação é escolhida com probabilidade *1-*ε e com probabilidade ε, ações são escolhidas aleatoriamente.

- ***max-suave(softmax):*** essa política resolve uma desvantagem dos métodos *greedy* e ε*-greedy*, que são as escolhas aleatórias, onde a pior ação pode ser selecionada com a mesma probabilidade da segunda melhor. Para isso, é atribuído uma classificação ou peso para cada uma das ações, com a estimativa de um valor de ação.

Sendo o método mais utilizado pelos autores da literatura o ε-greedy [Sutton e Barto 1998] [Russell e Norvig 2009]. A partir disso, o mecanismo que será utilizado para realizar os testes entre os algoritmos neste trabalho será o ε-greedy.

3.1.1 Descrição dos Métodos ε-greedy

Abaixo serão descritos os mecanismos que fazem parte do método ε-greedy.

ε-greedy

O método ε-*egreedy* possui a finalidade de manter o equilíbrio entre os componentes de exploração e de aproveitamento do ambiente. Esse equilíbrio é feito através de uma escolha probabilística entre, uma ação que maximiza o valor da função *Q(s,a)* para o estado atual *s* que o agente se encontra ou qualquer ação aleatória relacionada ao estado *s*. Esse mecanismo foi descrito e usado juntamente com o mecanismo de aprendizagem Q-learning [Watkins 1989].

Definição formal do mecanismo ε-greedy é dada pelo sistema:

$$a_t \begin{cases} a_t^* & \text{com probabilidade 1-}\varepsilon; \\ \text{ação aleatória com probabilidade } \varepsilon \end{cases}$$

A ideia central do mecanismo é selecionar qualquer ação aleatória de determinado estado *s* com probabilidade ε, ou selecionar a ação com maior valor a_t^* na função *Q(s,a)* com probabilidade (1-ε). O parâmetro ε é definido no intervalo [0,1]. Quanto maior o seu valor, mais estados inexplorados o agente visita.

ε-greedy com ε decrescente

O mecanismo ε-greedy com ε decrescente foi inicialmente mostrado no mesmo documento do ε-greedy e é uma de suas variações [Watkins 1989]. Essa técnica possui todas as características da ε-greedy. No entanto, nesse mecanismo o valor de ε decresce gradualmente na execução do algoritmo, fazendo o agente cada vez mais aproveitar a informação já adquirida.

Além do parâmetro ε, esse mecanismo também possui a *taxa de redução de epsilon* **TRE**, um parâmentro que é fixo e pode ser definido no intervalo [0,1]. A sua relação com ε, possui uma razão inversa.

ε-greedy VDBE-Boltzmann

O mecanismo de exploração VDBE-*Boltzmann* (Value-Difference-Based-Exploration) é um dos mais novos a ser proposto para ajudar o agente na exploração do ambiente [Tokic 2010]. Ele associa a cada estado *s* do ambiente um ε que corresponde a probabilidade de exploração e essa se adapta a medida que a função *Q(s,a)* se altera.

A ideia é usar uma função estado-ação que associa um valor ε dinâmico a cada estado *s*, de forma que, quando menor o conhecimento em determinado estado mais explorativo o agente se torna. Analogamente, a medida que o conhecimento em cada estado *s* se tornar maior, a taxa de exploração ε associada a esse estado diminui.

Abaixo pode-se visualizar a equação que pode ser obtida através da adaptação de uma distribuição de *boltzmann* [Tokic 2010]:

$$f(s,a,\sigma) = \left| \frac{e^{\frac{Q_t(s,a)}{\sigma}}}{e^{\frac{Q_t(s,a)}{\sigma}}+e^{\frac{Q_t(s,a)}{\sigma}}} - \frac{e^{\frac{Q_{t+1}(s,a)}{\sigma}}}{e^{\frac{Q_{t+1}(s,a)}{\sigma}}+e^{\frac{Q_{t+1}(s,a)}{\sigma}}} \right|$$

$$f(s,a,\sigma) = \frac{1-e^{-|\frac{Q_{t+1}(s,a)-Q_t(s,a)}{\sigma}|}}{1-e^{-|\frac{Q_{t+1}(s,a)-Q_t(s,a)}{\sigma}|}}$$

$$\varepsilon_{t+1}(s) = \delta.f(s_t,a_t,\sigma) + (1-\delta).\varepsilon_t(s)$$

Pode-se observar dois parâmetros associados a esse algoritmo, são eles:

- O parâmetro de qualidade de exploração δ, que é responsável por determinar o grau de influência de cada ação na taxa de exploração. O seu valor pode ser definido no intervalo [0,1] e de acordo com o autor do algoritmo [Tokic 2010], é aconselhável utilizar como valor, o inverso do número de ações possíveis para cada estado.

- A sensibilidade inversa σ, que pode ser definida através do intervalo]0,+α[é responsável por controlar o grau de exploração versus a variação de conhecimento obtido. Quanto menor seu valor, maior a influência da taxa de exploração.

3.1.2 Descrição dos Mecanismos *SoftMax*

Nesta seção será apresentado o algoritmo de exploração *SoftMax*.

SoftMax

O mecanismo de exploração *SoftMax* é baseado na formula de distribuição de *Boltzmann* e sua aplicação como método exploratório foi sugerido em 1994 juntamente com Algoritmo SARSA [Rummery e Niranjan 1994]. O mesmo consiste em aplicar valores de probabilidades de seleções especificas a cada uma das possíveis ações de cada estado do ambiente,

esses valores são proporcionais ao peso relativo de cada uma das ações diante o conjunto total delas.

Abaixo pode-se observar a definição do mecanismo:

$$P(a)_t = \frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{k} e^{Q_t(a)/\tau}}$$

onde, τ é o parâmetro de temperatura variando no intervalo de]0, +α[e *e* é o expoente de natural logaritmo. Altas temperaturas tendem à normalizar os pesos entre as ações como todos de igual importância, potenciando a exploração. Baixas temperaturas fazem os pesos associados as ações terem uma evolução bastante acentuada. Quanto menor o valor τ, mais potenciada é a seleção de ações valorizadas e com isso a taxa exploração é reduzida.

Devido ao conhecimento do agente ser pequeno no inicio da execução do algoritmo, o seu comportamento será totalmente exploratório, gerando uma distribuição uniforme de probabilidades de seleção de ações. Mas com o passar do tempo da execução do sistema e a medida que seu conhecimento for aumentando o agente tenderá a aproveitar melhor os valores das ações nos estados do ambiente.

3.2 Elemento de Aprendizagem

Como todos os ambientes descritos nesse trabalho são parcialmente observáveis e o agente não conhece todo o seu estado de ações, o elemento de aprendizagem irá fornecer os mecanismos que propiciarão ao agente uma maneira de processar as informações obtidas no ambiente, a partir disso ele irá conhecer cada vez mais os estados em que se encontre.

A quantidade de algoritmos para solucionar o problema da aprendizagem por reforço é vasta. No entanto, é possível observar que muitos deles são variações dos mecanismos já existentes e possuem o objetivo de melhorá-los. Os algoritmos expostos neste capítulo, apesar de apresentar variações nas suas estratégias e resoluções, possuem suas características fundamentais semelhantes.

Para esse trabalho foram escolhidos mecanismos que enquadram-se nos métodos de DT. Eles estão classificados em três famílias, sendo elas: *Q-Learning, SARSA e Dyna-Q*. Estes algoritmos foram escolhidos por possuírem atualmente uma alta relevância na resolução da problemática de aprendizagem por reforço. Além de não precisarem de um modelo completo

do ambiente e não precisarem esperar até o final do episódio para atualizar os valores das funções, onde estas são as vantagens dos métodos PD e MC, que também são utilizados para solucionar esse tipo de problemática [Sutton e Barto 1998] [Szepesvári 2010].

3.2.1 *Q-Learning*

O algoritmo de aprendizagem *Q-Learning* é um dos algoritmos bases no método de DT. Ele foi descrito inicialmente por *C. Watkins*, na sua tese de doutorado na universidade Cambrige em 1989 [Watkins 1989]. Essa técnica pode ser considerada uma das mais importantes contribuições na aprendizagem por reforço, pois, sua característica de escolher, a cada passo de tempo *t* a ação mais valorizada no estado atual, habilita uma maior velocidade de convergência no aprendizado da função estado-ação *Q(s,a)*. Essa função atualiza a matriz dos Q-valores no algoritmo *Q-learning* e é denotada pela expressão 3.1:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \tag{3.1}$$

onde s_t é o estado atual, a_t é a ação atribuída ao s_t, e r_t é o valor de reforço recebido após executar a ação a_t no estado s_t. s_{t+1} é o próximo estado a ser alcançado, γ é o fator de desconto (0 <= γ < 1) e α (0 <= α < 1) é o coeficiente de aprendizagem. O par de estado-ação *(s,a)* recebe o valor da função *Q(s,a)*, que representa se o valor da função é uma boa escolha para a maximização da função de retorno [Sutton e Barto 1998].

No Algoritmo 1 é apresentado uma forma simplificada do procedimento *Q-learning*:

Algorithm 1 *Q-Learning*

procedure QLEARNING(r,α, ε, γ)
    Inicialize Q(s,a)
    repeat(para cada episódio)
        Inicialize *s*
        repeat(para cada estado do ambiente)
            Escolher *a* de *s* usando a política $\varepsilon - greedy$
            Observe os valores de *r* e *s'*
            Q(s_t,a_t) $\leftarrow$ Q(s_t,a_t) + α(r_t+γmax_a Q(s_{t+1},a)-Q(s_t,a_t))
            $s \leftarrow s'$
        until encontrar o estado final $s \in S$
    until convergência do algoritmo
return Q(s,a)
end procedure

3.2.2 *Q-Learning* λ *(eligibility traces)*

O mecanismo *Q-learning eligibility traces* é uma das variações do *Q-learning*. A integração do conceito de *eligibility traces* no problema de aprendizagem por reforço foi realizada em 1989 por *C. Watkins* em seu trabalho de tese para Cambrige [Watkins 1989]. No entanto, o precursor e idealista dessa técnica foi *Klopf*, que apresentou o seu conceito teórico na USAF (United States Air Force) em um trabalho sobre sistemas adaptativos [Klopf 1972].

A técnica de *eligibility traces* aplicado aos métodos de diferença temporal, pode tornar o aprendizado do agente mais eficiente. Seu principal conceito consiste em utilizar um registro temporário de determinados eventos. Os eventos podem ser caracterizados como a visita a algum estado do ambiente ou a seleção de determinada ação. Esse registro possui como objetivo marcar os parâmetros associados ao evento como elegíveis para que possam passar por alterações. Assim, se durante a execução do algoritmo algum erro ocorrer, esse poderá ter sua origem relacionada com os estados ou ações previamente marcadas como elegíveis.

Essa marcação segue a regra do sistema abaixo:

$$e(s,a)\begin{cases} \gamma\lambda e(s,a) + 1 & se\ \ s = S_t\ \ e\ \ a = a_t \\ \gamma\lambda e(s,a) & \end{cases} \qquad para\ \ todo\ \ s,a$$

No Algoritmo 2 é apresentado uma forma simplificada do procedimento *Q-learning* λ *(eligibility traces)*:

Algorithm 2 *Q-Learning* λ

procedure QLEARNING λ(r,$\alpha, \varepsilon, \gamma, \lambda$)
    Inicializar Q(s,a) arbitrariamente e e(s,a) = 0, para todo *s,a*
    repeat(para cada episódio)
        Inicializar *s,a*
        repeat(para cada estado do ambiente)
            Escolher *a* de *s* usando a política em *Q*
            $a^* \leftarrow argmax_b$ Q(s',b) (se a' for igual ao max, então $a^* \leftarrow$ a')
            $\delta \leftarrow r + \gamma Q(s', a^*) - Q(s, a)$
            e(s,a) $\leftarrow$ e(s,a) + 1;
            for all s,a **do**
                $Q(s, a) \leftarrow Q(s, a) + \alpha\gamma e(s, a)$
                Se *a'* = a^*, então $e(s, a) \leftarrow \gamma\lambda e(s, a)$
                senão $e(s, a) \leftarrow 0$
            end for
            s $\leftarrow$ *s'*
            a $\leftarrow$ *a'*
        until encontrar o estado final $s \in S$
    until convergência do algoritmo

end procedure

Visualizando o algoritmo é possível observar que sua base esta relacionada com o algoritmo *Q-Learning*, mudando apenas a estratégia de resolução do problema.

Este algoritmo possui um novo parâmetro, o $\lambda \in [0,1]$, que por sua vez possui uma relação inversa com a taxa de decaimento da marcação do evento. Quanto maior seu valor, menor será a velocidade que a marcação do evento decai ao longo da execução do experimento.

Também é possível observar que a marcação dos eventos é realizada por uma função estado-ação auxiliar *e(s,a)*, está que depois de incrementada é incorporada na função *Q(s,a)*; linha 11.

Essa é uma técnica acumulativa, pois cada seleção de determinada ação *a* em um determinado estado *s* é realizada de forma unitária e incremental. A partir do momento que o agente passar a não visitar o estado, essa marcação vai desaparecendo gradualmente, como pode ser visualizado na Figura 3.1.

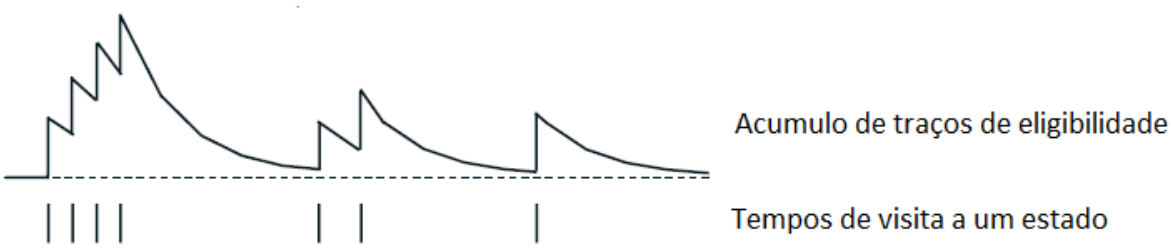

Figura 3.1: Tempo das marcações no algoritmo *eligibility traces*

3.2.3 *Q-Learning* λ *(replacing)*

No conceito de *eligigibility traces* à marcação de um evento é realizada de forma incremental (acumulativa). No entanto, foi observado que se algum evento for visitado e revisitado em um período próximo de tempo, o mesmo vai ser incrementado de forma excessiva e seu valor será maior que 1, com isso esses eventos serão sobreavaliados pelo sistema. Então, observando esse problema *Sutton* em 1996 apresenta uma solução. [SUTTON e SINGH 1996].

A ideia sugerida por ele foi substituir a atribuição de marcação incremental do *eligibility traces* por um valor unitário, não acumulável. Essa alternativa demonstrou uma marcação mais rápida no ambiente e um processo de informação de maior qualidade durante o processamento da aprendizagem.

Essa marcação segue a regra do sistema abaixo:

$$e(s,a) \begin{cases} \gamma\lambda e(s,a) & se\ \ s \neq S_t\ \ e\ \ a \neq a_t \\ 1 & se\ \ s = S_t\ \ e\ \ a = a_t \end{cases}$$

No Algoritmo 3 é apresentado uma forma simplificada do procedimento *Q-learning* λ *(replacing)*:

Algorithm 3 *Q-Learning* λ*(replacing)*

procedure QLEARNING λ(r,$\alpha, \varepsilon, \gamma, \lambda$)
    Inicializar Q(s,a) arbitrariamente e e(s,a) = 0, para todo *s,a*
    repeat(para cada episódio)
        Inicializar *s,a*
        repeat(para cada estado do ambiente)
            Escolher *a* de *s* usando a política em *Q*
            $a^* \leftarrow argmax_b$ Q(s',b) (se a' for igual ao max, então $a^* \leftarrow$ a')
            $\delta \leftarrow r + \gamma Q(s', a^*) - Q(s, a)$
            e(s,a) $\leftarrow$ 1;
            for all s,a **do**
                $Q(s, a) \leftarrow Q(s, a) + \alpha\gamma e(s, a)$
                Se *a'* = a^*, então $e(s, a) \leftarrow \gamma\lambda e(s, a)$
                senão $e(s, a) \leftarrow 0$
            end for
            s $\leftarrow$ *s'*
            a $\leftarrow$ *a'*
        until encontrar o estado final $s \in S$
    until convergência do algoritmo

end procedure

Pode-se observar uma pequena diferença na atribuição da função estado-ação *e(s,a)*, sendo essa uma marcação com valor unitário e não acumulável.

Como pode ser visualizado na figura 3.2, a marcação de forma unitária e não incremental de um evento que é visitado e revisitado em próximo período de tempo, ocorre com um valor máximo e não acumulativo, diferente de como foi descrito na marcação dos eventos da técnica *eligibility traces*.

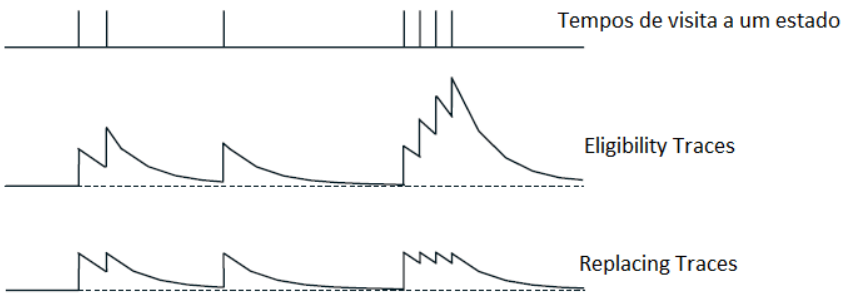

Figura 3.2: Tempo das marcações no algoritmo *eligibility traces* e *replacing*

3.2.4 *SARSA*

O algoritmo *SARSA (State-Action-Reward-State-Action)* é outro mecanismo base do método de DT [Sutton e Barto 1998]. Ele foi descrito inicialmente em 1994 por *Rummery e Niranjan*, na universidade de Cambridge [Rummery e Niranjan 1994]. Sua teoria é muito parecida com a do mecanismo *Q-Learning*, sendo a única diferença, a maneira como a função estado-ação *Q(s,a)* é atualizada. No mecanismo *SARSA* a função estado-ação *Q(s,a)* é atualizada de acordo com o método de exploração e não com o valor da ação mais alta em determinado estado atual. A função que atualiza a matriz dos Q-valores no algoritmo *SARSA* é denotada pela expressão 3.2:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \boldsymbol{Q(s}_{t+1}, a_{t+1}) - Q(s_t, a_t)) \quad (3.2)$$

onde s_t é o estado atual, a_t é a ação atribuída ao s_t, e r_t é o valor de reforço recebido após executar a ação a_t no estado s_t. s_{t+1} é o próximo estado a ser alcançado, γ é o fator de desconto $(0 <= \gamma < 1)$ e $\alpha(0 <= \alpha < 1)$ é o coeficiente de aprendizagem.

Pode-se observar na Figura 3.3 o ambiente *CliffWalking*, onde foi realizado uma análise entre os mecanismos *Q-Learning* e *SARSA* [Sutton e Barto 1998]. O agente para chegar no objetivo possui dois caminhos; um é passar em estados que estão bem próximos de um penhasco e correr o risco de cair e o outro é passar por estados mais distantes do penhasco e seguir em segurança. Como questão de recompensa, o agente receberá -100 se cair no penhasco e -1 para se movimentar nos estados do ambiente.

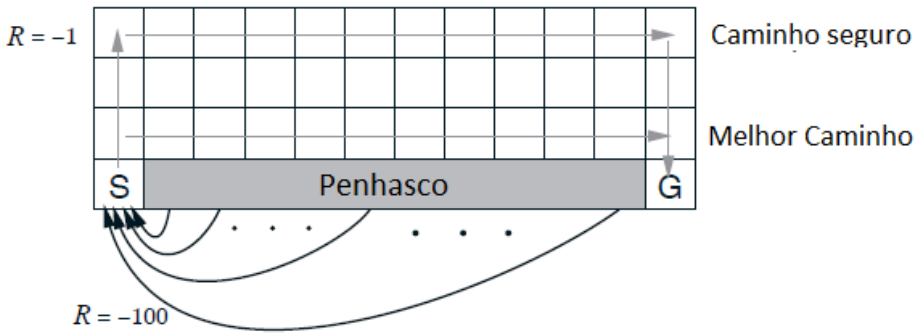

Figura 3.3: Análise entre o algoritmo QLearning e SARSA

Neste ambiente o algoritmo *Q-Learning* é mais eficiente e chega mais rápido ao estado final, pois suas escolhas são baseadas nas ações dos estados que possuem maior valor, ignorando completamente todas as outras. Sendo assim, mesmo correndo o risco de andar pelos estados que estão na beira do penhasco, o agente os escolhe para chegar mais rápido ao objetivo. Já o algoritmo *SARSA* é mais conservador e escolhe o caminho mais seguro, sendo esse, o dos estados que estão mais distantes do abismo. Por consequência ele se torna mais lento nesse ambiente.

No Algoritmo 4 é apresentado uma forma simplificada do procedimento *SARSA*:

Algorithm 4 *SARSA*

procedure SARSA(r,$\alpha, \varepsilon, \gamma$)
    Inicialize Q(s,a)
    repeat(para cada episódio)
        Inicialize *s* State Escolher *a* de *s* usando a política $\varepsilon - greedy$
        repeat(para cada estado do ambiente)
            Observe os valores de *r* e *s'*
            Escolher a_{t+1} de s_{t+1} usando a política $\varepsilon - greedy$
            $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$
            $s \leftarrow s'$
            $a \leftarrow a'$
        until encontrar o estado final $s \in S$
    until convergência do algoritmo
return Q(s,a)
end procedure

3.2.5 *SARSA* (λ *eligibility traces)*

A técnica de *eligibility traces* também foi incorporada no mecanismo *SARSA*, criando assim uma variação desse algoritmo e possuindo o objetivo de torná-lo mais eficiente. Todo o seu conceito é idêntico ao descrito no algoritmo *Q-Learning* λ, também possuindo algumas semelhanças relacionadas a implementação.

A diferença na implementação da técnica de *eligibility traces* no *SARSA* está no seu conceito de valorizar todas as ações possíveis de determinado estado.

No Algoritmo 5 é apresentado uma forma simplificada do procedimento *SARSA* λ *(eligibility traces)*:

Algorithm 5 *SARSA λ (eligibility traces)*

procedure SARSA λ (ELIGIBILITY TRACES)(r,$\alpha, \varepsilon, \gamma, \lambda$)
    Inicializar Q(s,a) arbitrariamente e e(s,a) = 0, para todo *s,a*
    repeat(para cada episódio)
        Inicializar *s*
        Escolher *a* de *s* usando a política em *Q*
        repeat(para cada estado do ambiente)
            $\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$
            Escolher *a* de *s* usando a política em *Q*
            e(s,a) $\leftarrow$ e(s,a) + 1;
            for all s,a **do**
                $Q(s, a) \leftarrow Q(s, a) + \alpha\gamma e(s, a)$
                $e(s, a) \leftarrow \gamma\lambda e(s, a)$
            end for
            s $\leftarrow$ *s'*
            a $\leftarrow$ *a'*
        until encontrar o estado final $s \in S$
    until convergência do algoritmo

end procedure

É possível observar nas linhas que em cada estado existente *s*, são marcados os eventos relacionados a qualquer ação *a* desse estado que tenha sido selecionada pela política associada.

3.2.6 SARSA (λ *replacing)*

Em 1996 *Sutton* [SUTTON e SINGH 1996] também descreve a técnica de *replacing* para o mecanismo *SARSA*. O conceito é o mesmo utilizado para o *Q-learning (replacing)*; usar uma estratégia unitária e não acumulável para a marcação dos eventos. Como pode ser observado pelo algoritmo do mecanismo de aprendizagem 6:

Algorithm 6 *Sarsa λ (replacing)*

procedure SARSA λ *(replacing)*(r,$\alpha, \varepsilon, \gamma, \lambda$)
    Inicializar Q(s,a) arbitrariamente e e(s,a) = 0, para todo *s,a*
    repeat(para cada episódio)
        Inicializar *s*
        Escolher *a* de *s* usando a política em *Q*
        repeat(para cada estado do ambiente)
            $\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$
            Escolher *a* de *s* usando a política em *Q*
            e(s,a) $\leftarrow$ 1;
            for all s,a **do**
                $Q(s, a) \leftarrow Q(s, a) + \alpha\gamma e(s, a)$
                $e(s, a) \leftarrow \gamma\lambda e(s, a)$
            end for
            s $\leftarrow$ *s'*
            a $\leftarrow$ *a'*
        until encontrar o estado final $s \in S$
    until convergência do algoritmo

end procedure

3.2.7 Mecanismos *Dyna-Q*

Dyna-Q

O *Dyna-Q* [SUTTON 1990] é um mecanismo que propõe uma diferente arquitetura para solucionar o problema da aprendizagem por reforço. Sua proposta é acrescentar à solução do problema de aprendizagem por reforço um *modelo interno do mundo*. Esse modelo tem o objetivo de simular o ambiente em que realmente o agente se encontra, ou seja, a partir de um estado *s* e uma ação *a*, o agente poderá obter uma previsão da recompensa *r* e do próximo estado *s'*. O modelo do mundo é construído a cada episódio e a partir dos estados observados pelo agente no decorrer das interações com o ambiente.

Na Figura 3.4 é possível observar que a arquitetura *Dyna-Q* usa o modelo do mundo como um ambiente auxiliar ao utilizado no mecanismo de aprendizagem por reforço (AR). Também é possível visualizar que as etapas de aprendizagem se alternam entre o ambiente do AR e o modelo do mundo. Sendo usado no modelo do mundo os resultados previstos ao invés dos reais. Com essa estratégia, o mecanismo *Dyna-Q* faz cada experiência real ser dividida com várias hipotéticas no modelo do mundo, assim, contribuindo para o desempenho e processo de aprendizagem do agente.

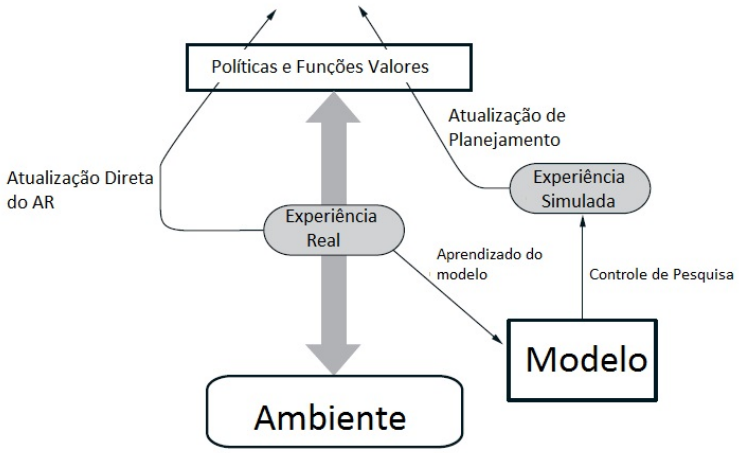

Figura 3.4: Arquitetura Dyna-Q

Acumulando essas experiências a arquitetura consegue uma aproximação maior da política ótima diante o modelo existente, assim criando uma forma de planejamento. Em resumo,

o *Dyna-Q* combina sua arquitetura junto com o mecanismo *Q-learning* e através de um modelo do mundo simula experiências do agente no ambiente AR.

O Algoritmo 7 é apresentado o procedimento *Dyna-Q*:

Algorithm 7 *Dyna-Q*

procedure DYNA-Q(r,$\alpha, \varepsilon, \gamma$)
    Inicializar Q(s,a) e modelo(s,a), para todo *s,a*
    repeat(para cada episódio)
        Inicializar *s*
        Escolher *a* de *s* usando a política em *Q*
        realizar ação *a* e observar *r, s'*
        $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha r_t + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)$
        Model(s,a) $\leftarrow$ *s',r* (Ambiente determinístico)
        repeat(N times)
            $s \leftarrow$ estado randômico observado previamente
            $a \leftarrow$ ação randômica realizada previamente em *s*
            s',r $\leftarrow$ *Model(s,a)*
            $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha r_t + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)$
        until encontrar o estado final $s \in S$
    until convergência do algoritmo

end procedure

É possível observar no algoritmo 7 a existência de uma nova função estado-ação representada por *Model(s,a)* e de um ciclo que ocorre a cada episódio no mecanismo de aprendizagem. O modelo do mundo é definido nessa nova função estado-ação e no ciclo é onde ocorre o planejamento da arquitetura *Dyna-Q*.

O planejamento é realizado a partir de sucessivas atualizações da função estado-ação *Q(s,a)*, juntamente com experiências aleatórias já adquiridas no modelo do mundo. As atualizações são gerenciadas através da variável *N*, que controla quantas interações são executadas a cada episódio do ambiente. Quanto maior seu valor, melhor será o planejamento das ações futuras do agente, devido ao maior número de atualizações sofridas per ele, conhe-

cendo assim melhor o ambiente a partir do modelo simulado.

Com o conceito de planejamento das ações, o algoritmo *Dyna-Q* possui um maior custo computacional em relação aos outros algoritmos descritos neste trabalho, pois além dele atualizar a função *Q(s,a)* nos estados do ambiente em que está inserido, é necessário que ele realize um ciclo de interação com todos os estados que se encontram no ambiente do mundo (ambiente de planejamento), gerando assim uma maior quantidade de processamento.

Dyna-Q com varrimento priorizado

Em 1992 *Peng* e *Williams* [Peng e Williams 1992] encontraram e abordaram em um de seus trabalhos um problema na arquitetura *Dyna*. Analisando o algoritmo *Dyna-Q* é possível observar que as interações realizadas no modelo do mundo são de forma aleatória, ou seja, não existe um critério de escolha do par estado-ação relacionado ao modelo do mundo, eles são escolhidos de forma totalmente randomizadas.

Então em 1993 *Moore* e *Atkeson* [Moore e Atkeson 1993], depois de analisarem esse problema, propuseram uma alternativa chamada de varrimento priorizado. A ideia foi adicionar ao mecanismo *Dyna-Q* uma maneira de priorizar a escolha dos pares estado-ação no momento da iteração com modelo do mundo. Com essa característica as escolhas das ações se tornaram mais controladas ao invés de puramente aleatórias. Com isso, o mecanismo passou a priorizar as escolhas através da implementação de uma fila que armazena os valores *(s,a)* visitados.

No Algoritmo 8 é apresentado o procedimento *Dyna-Q com varrimento priorizado*:

Algorithm 8 *Dyna-Q VP*

procedure DYNA-Q VP(r,$\alpha, \varepsilon, \gamma, \theta$, N)
    Inicializar Q(s,a) e modelo(s,a) para todo *s,a* e zerar PQueue
    repeat(para cada episódio)
        Inicializar *s*
        Escolher *a* de *s* usando a política em *Q*
        realizar ação *a* e observar *r, s'*
        Model(s,a) $\leftarrow$ *a', r*
        p $\leftarrow$ |r + $\gamma max'_a$Q(s',a')-Q(s,a)|
        Se *p* > θ, insira *s,a* dentro PQueue com prioridade *p*
        repeat(*N* vezes, enquanto PQueue não estiver vazia)
            s,a $\leftarrow$ *fisrt(PQueue)*
            s',r $\leftarrow$ *Model(s,a)*
            $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max'_a Q(s',a') - Q(s,a)]$
            repeat(para todo $\bar{s}$, $\bar{a}$ previsto a alcançar *s*)
                $\bar{r} \leftarrow$ recompensa prevista
                p $\leftarrow$ |r + γmax_aQ($\bar{s}, \bar{a}$) $- Q(s,a)$| $Sep>\theta$, insira *s,a* dentro PQueue com prioridade *p*
            until alcançar s
        until N vezes ou Fila zerada
    until convergência do algoritmo

end procedure

É possível visualizar, que agora a escolha dos estados e ações no modelo do mundo é controlada por uma fila *PQueue*. Também é possível visualizar uma nova variável θ que controla quais elementos serão inseridos na fila.

3.2.8 Algoritmos Implementados

Na área de aprendizado de máquina, mais especificamente a área de aprendizagem por reforço, os algoritmos clássicos ainda são muito utilizados na resolução de problemas com esse

conceito. Com isso, visando uma comparação de eficiência entre esses algoritmos, foram escolhidos alguns dos mais utilizados pela literatura e comparados em diversos ambientes. Possuindo essa análise o objetivo de observar qual dos algoritmos resolve de maneira mais eficiente o problema encontrado no ambiente. Essa eficiência será medida de acordo com algumas métricas, que serão observadas mais adiante neste trabalho.

Devido a grande quantidade de ambientes, os algoritmos mais implementados foram o *Q-Learning*, *SARSA* e suas variações. O *Dyna-Q* também foi implementado, no entanto em apenas "dois" ambientes.

Lista dos algoritmos implementados:

- *Q-Learning;*
- *Q-Learning* λ *(eligibility traces);*
- *Q-Learning* λ *(raplacing);*
- *SARSA;*
- *SARSA* λ *(eligibility traces)*;
- *SARSA* λ (*replacing*);
- *Dyna-Q*

Com os testes desses algoritmos será possível avaliar suas vantagens e desvantagens em ambientes que desafiem suas técnicas de resoluções do problema de aprendizagem por reforço.

Capítulo 4

Biblioteca AILibrary-RL

Neste capítulo será fornecido uma breve introdução sobre o desenvolvimento da AILibrary-RL[1], juntamente com sua arquitetura.

4.1 Arquitetura AILibrary-RL

A ideia de construir uma biblioteca com os algoritmos analisados, foi planejada com o objetivo de criar vantagens em alguns pontos no desenvolvimento de novas soluções para o problema da aprendizagem por reforço, sendo elas:

- **Modularização:** Observar os elementos da aprendizagem por reforço que interagem entre si e a partir disso, criar módulos que dividem o problema e facilite a implementação de novas aplicações.
- **Reutilização:** Facilitar o desenvolvimento de novos experimentos a partir da reutilização de código.

Juntando esses dois conceitos na biblioteca, não se torna mais necessário a cada novo experimento ficar implementando o mesmo código de mecanismo de aprendizagem ou ambiente. Com a *AILibrary-RL*, os códigos poderão ser reutilizados.

A arquitetura da biblioteca pode ser visualizada na Figura 4.1:

[1]Os códigos relacionados a biblioteca AILibrary-RL podem ser encontrados no github:https://github.com/thiagorm/AILibrary-RL

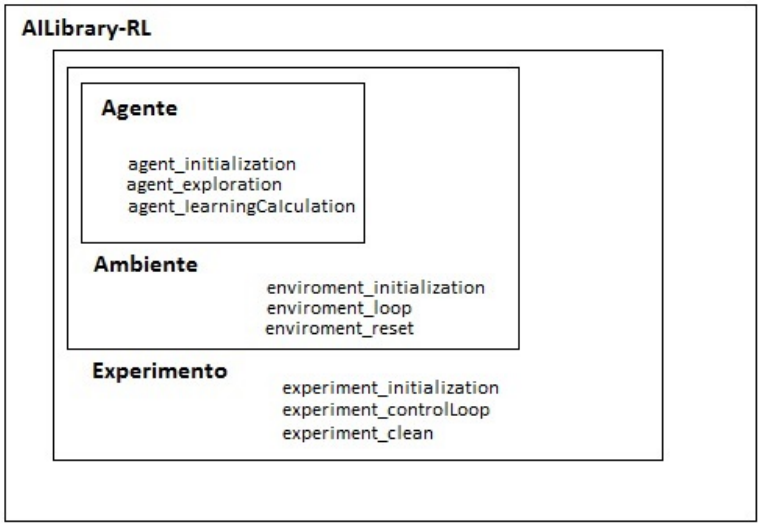

Figura 4.1: Arquitetura Padrão da Biblioteca AILibrary-RL

Como pode ser observado na Figura 4.1, os componentes mais importantes da *AILibrary-RL* são, os Agentes, Ambientes e Experimentos. Onde, cada um deles possuem métodos que devem ser implementados para que a comunicação entre os componentes seja gerenciada e organizada pela biblioteca.

Também pode ser visualizado a integração entre os componentes, notando que cada um deles estão incluídos uns nos outros, ou seja, o componente agente está incluído no componente ambiente, que os dois estão incluídos no componente experimento e por fim todos fazem parte da *AILibrary-RL*.

O diagrama têm como objetivo, explicar que para haver a comunicação entre os componentes é necessário que eles troquem informações através de métodos, para que o experimento ocorra. Por exemplo, depois da implementação do agente e seus métodos é necessário que ele seja integrado ao ambiente, para que seus métodos possam interagir com os métodos do ambiente e assim encontrar uma solução para resolver o problema do ambiente.

4.2 Agente, Ambiente e Experimento

Na Figura 4.2, os componentes podem ser visualizados com mais detalhes:

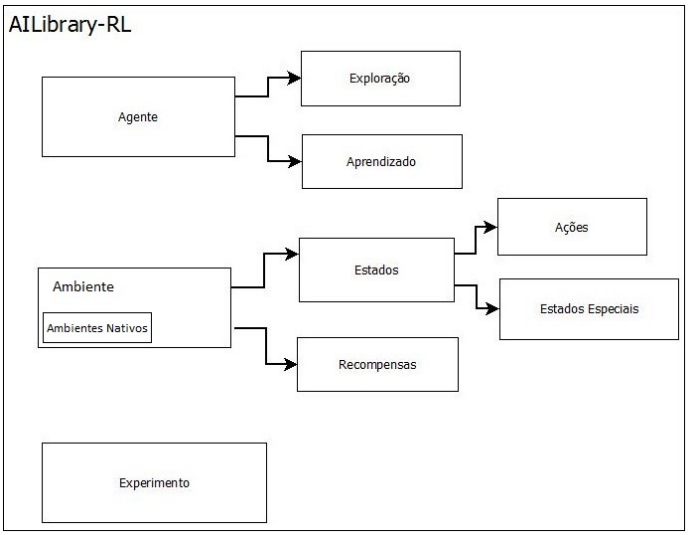

Figura 4.2: Componentes da Biblioteca AILibrary-RL

Para um melhor entendimento da Figura 4.2, cada componente é descrito abaixo:

- **Agente:** O agente decide qual ação vai ser realizada no ambiente a cada passo da execução do experimento. O componente agente é integrado à dois outros componentes: o de exploração, onde são implementados os métodos de exploração que poderão ser utilizados pelo agente e o componente de aprendizagem, onde são implementados os mecanismos de aprendizagem que o agente poderá utilizar.

- **Ambiente:** O ambiente é responsável por controlar os detalhes do mundo que o agente está inserido. Como os estados, estados especiais (estado inicial, final e obstáculos), as transições entre os estado e as recompensas. Nesse módulo também existe a opção de reutilizar ambientes já implementados, ou seja, os ambientes nativos que podem ser inseridos na biblioteca com o objetivo de agilizar ainda mais o desenvolvimento das soluções.

- **Experimento:** O experimento é uma classe principal que controla o *loop* de tempo de convergência, ou seja, onde é controlado o tempo de convergência do mecanismo de aprendizagem, a inicialização do ambiente e agente e a comunicação entre eles.

Para visualizar com mais detalhes como funciona a comunicação entre o agente e o ambiente é possível observar na Figura 4.3 os parâmetros que são trocados entre eles:

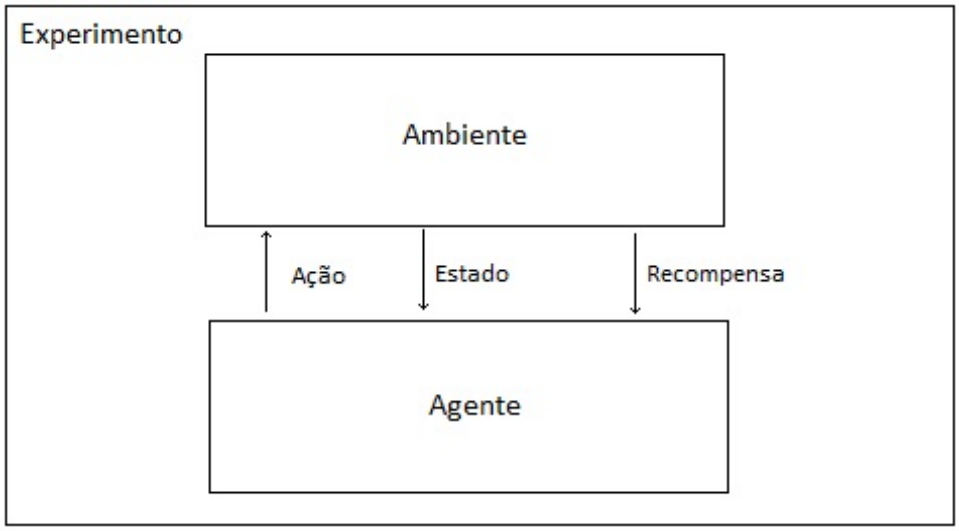

Figura 4.3: Visão dos parâmetros informados entre agente e ambiente

Na Figura 4.3, pode-se visualizar que quando o agente realiza determinada ação, sendo essa fornecida pelo módulo de exploração, o ambiente retorna um estado e uma recompensa referente aquela ação realizada.

4.3 Métodos do Agente

Os métodos que precisam ser implementados pelo agente são:

- **agent_initialization:** Esse método é responsável por inicializar os parâmetros do agente, como o *alfa*, o *gamma* o ε-*greedy*.
- **agent_exploration:** Esse método é responsável por controlar a chamada de um método de exploração.
- **agent_learningCalculation:** Esse método é responsável por controlar a chamada de um mecanismo de aprendizagem.

4.4 Métodos do Ambiente

Os métodos que precisam ser implementados pelo ambiente são:

- **enviroment_initialization:** Esse método é responsável por inicializar os estados do ambiente, definir os estados iniciais, finais e obstáculos e gerar a semente de randomização.

- **enviroment_loop:** Esse método possui o objetivo de controlar o *loop* interno do ambiente. No seu parâmetro o agente é passado para que o mesmo possa interagir com o ambiente. Com isso, fica mais claro o conceito de integração entre os módulos. Assim, dividindo cada trabalho ao seu respectivo módulo.

- **enviroment_reset:** O objetivo desse método é passar informações para o experimento de quantos passos e episódios já foram alcançados. A cada chegada do agente no estado final ou objetivo, o método reinicializa o agente para o estado inicial.

4.5 Métodos do Experimento

O método que precisa ser implementado pelo experimento é:

- **experiment_initialization:** Esse método é responsável por organizar as inicializações do agente, ambiente e variáveis globais do experimento.

- **experiment_controlLoop:** O objetivo desse método é controlar o *loop* do ambiente, ou seja, verificar se o número de passos para o termino do experimento foi alcançado.

- **experiment_clean:** Nesse método é realizada a liberação de memória que tenha sido utilizada pelo agente ou ambiente durante o experimento.

4.6 Exemplo da Aplicação da Biblioteca

Para demonstrar uma aplicação prática da biblioteca, foi construído um ambiente do tipo *tilemap* 3x4 e utilizado o mecanismo *Q-Learning* para guiar o agente até o objetivo. Na Figura 4.4 o ambiente pode ser visualizado com mais detalhes:

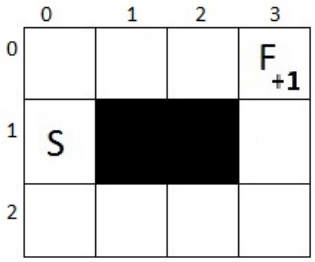

Figura 4.4: Ambiente usado para aplicação prática da biblioteca

É possível observar que o ambiente é bem simples, contendo apenas três linhas e quatro colunas, um estado inicial representado pela letra "S", um estado final representado pela letra "F" e dois obstáculos representado na cor preta. A cada passo o agente recebe uma recompensa de valor "0" para cada estado alcançado, exceto se ele alcançar o estado final, pois nesse estado ele recebe "+1" de recompensa. O objetivo do agente é sair do estado inicial "S" e percorrer o melhor caminho para chegar no estado final "F".

O estado do ambiente pode ser representado pela Figura 4.5:

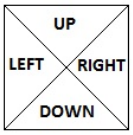

Figura 4.5: Representação do estado do ambiente com suas ações

Como visualizado na Figura 4.5 cada estado do ambiente possui "quatro" ações, sendo elas: "UP", "DOWN", "LEFT", "RIGHT". Na biblioteca *AILibrary-RL* esses estados são representados pela classe *State*. Nela encontra-se a declaração do número de ações que o estado possui ("numberActions") e dois *arrays*, cada um com o número de elementos iguais ao número de ações por estado. Os *arrays* podem armazenar números inteiros ou ponto flutuante, a escolha de qual tipo usar, vai depender das características do ambiente utilizado.

- int actions[numberActions];
- float actions[numberActions];

Dependendo das características do ambiente, novas informações poderão ser adicionadas na classe *State*.

Os principais métodos de cada módulo foram implementados de acordo com os códigos abaixo:

Métodos do Agente:

Código Fonte 4.1: Método Agent_Initialization

```
1 void QLearning::Agent_Initialization(float alpha, float gamma, float eGreedy, float lambda)
2 {
3         this->rlParameters.setAlpha(alpha);
4         this->rlParameters.setGamma(gamma);
5         this->rlParameters.setEgreedy(eGreedy);
6         this->rlParameters.setLambda(lambda);
7 }
```

É possível observar que no método *agent_initialization* 4.1, os parâmetros principais do mecanismo de aprendizagem e método de exploração são inicializados. O *agent_initialization* chama a variável *rlParameters*, que está declarada na classe do agente, para inicializar os parâmetros. Essa variável é do tipo *RLParameters*, ou seja uma classe com mesmo nome, que é responsável por gerenciar e inicializar todos os parâmetros do experimento.

Código Fonte 4.2: Método Agent_Exploration

```
1 ExplorationMethods QLearning::Agent_Exploration()
2 {
3         return this->explorationMethods;
4 }
```

Código Fonte 4.3: Método getActionEgreedy()

```
1  int ExplorationMethods::getActionEgreedy(State state)
2  {
3          float random_eGreedy = 0;
4          int random_action;
5          int action = NULL;
6
7          random_eGreedy = (float) rand()/(float)RAND_MAX;
8
9          if(this->rlParameters.getEGreedy() > random_eGreedy)
10         {
11                 //chosing the random action
12                 action = ((rand()%state.getNumberActions()));
13         }
14         else
15         {
16                 //chosing the max action
17                 action = verifyMaxAction(state);
18         }
19
20         return action;
21 }
```

No *agent_exploration* 4.2 é onde ocorre a chamada do método *getExploration-Method(State state, RLParameters rlParameters)*, onde é passado por parâmetro um estado do ambiente e o *RLParameters*. Esse método está localizado na classe *ExplorationMethods* que é responsável por organizar todos os métodos de exploração, como ε*-greedy* e *softmax*. O retorno deste método é uma ação referente a determinado estado do ambiente.

É possível observar no código do método *getActionEgreedy* 4.3, que a variável *rlParameters* fornece o valor do parâmetro *"Egreedy"*, com o objetivo de que seja realizada a comparação para determinar a escolha entre, a maior ação do ambiente ou uma ação randômica, referente ao método de exploração ε*-greedy*.

Código Fonte 4.4: Método Agent_LearningCalculation

```
1  double QLearning::QLearningCalculation(State state, State nextState, int action, float
       reward)
2  {
3          return (state.actions[action] + this->rlParameters.getAlpha() * (reward + ((this->
               rlParameters.getGamma() * verifyMaxValue(nextState)) - state.actions[action])))
               ;
4  }
```

Visualizando o método *agent_learningCalculation* 4.4, é possível observar que ele é responsável por calcular o aprendizado do agente no decorrer do experimento. Nos seus parâmetros estão, a ação realizada, o estado atual do agente, o próximo estado do agente e a recompensa pela ação realizada. Também é possível visualizar que os parâmetros *alpha* e *gamma* são utilizados a partir da chamada de método *getAlpha()* e *getGamma()*, realizada pela variável *rlParameters* declarada na classe do agente. Com esses parâmetros o mecanismo de aprendizagem calcula a função Q e retorna o valor calculado para a ação do estado no ambiente.

Métodos do Ambiente:

Código Fonte 4.5: Método Enviroment_Initialization

```
1  void ExemploEnviroment::Enviroment_Initialization(int numberActions, float reward, float
       reward_goal)
2  {
3          for(i = 0; i < L; i++)
4          {
5                  for(j = 0; j < C; j++)
6                  {
7                          grid[i][j].setNumberActions(numberActions);
8                          grid[i][j].setActions(grid[i][j].getNumberActions());
9                          for(k = 0; k < grid[i][j].getNumberActions(); k++)
10                         {
11                                 if(i == 1 && j == 1)
12                                         grid[i][j].actions[k] = WALL;
13                                 else if(i == 1 && j == 2)
14                                         grid[i][j].actions[k] = WALL;
15                                 else
16                                         grid[i][j].actions[k] = 0;
17
18                         }
19                 }
20         }
21
22         this->i = 1;
23         this->j = 0;
24         this->reward = reward;
25         this->reward_goal = reward_goal;
26 }
```

O método *enviroment_initialization*, é responsável por inicializar as variáveis do ambiente. Como o ambiente utilizado para teste é do tipo *tilemap*, é possível observar no Método 4.5, que "três" laços de repetição gerenciam a inicialização do ambiente. Os que são con-

trolados pelas variáveis “i” e “j” gerenciam a inicialização da matriz do ambiente, já o que é controlado pela variável “k”, gerencia a inicialização dos valores para os *arrays* das ações, onde também é verificado quais *tilemaps* do ambiente são obstáculos, para que o mesmo seja referenciado como obstáculo. No final é atribuído os valores das recompensas e às variáveis “i” e “j”, o estado inicial.

Código Fonte 4.6: Método Enviroment_Loop

```
1  void ExemploEnviroment::Enviroment_Loop(QLearning qLearning)
2  {
3          action = qLearning.Agent_Exploration().getActionEgreedy(grid[i][j], qLearning.
               rlParameters);
4
5                  switch(action)
6                  {
7                          //UP
8                          case 0:
9                                  if(i - 1 >= 0)
10                                 {
11                                         if(grid[i-1][j].actions[action] != WALL)
12                                         {
13                                                 this->nextState_i = i-1;
14                                                 this->nextState_j = j;
15                                                 if(i-1 == 0 && j == 3)
16                                                         grid[i][j].actions[action] =
                                                               qLearning.
                                                               Agent_LearningCalculation(grid[
                                                               i][j], grid[nextState_i][
                                                               nextState_j], action,
                                                               reward_goal);
17                                                 else
18                                                         grid[i][j].actions[action] =
                                                               qLearning.
                                                               Agent_LearningCalculation(grid[
                                                               i][j], grid[nextState_i][
                                                               nextState_j], action,reward);
19                                                 i--;
20                                                 total_reward+=reward;
21                                         }
22                                 }
23                                 actions++;
24                         break;
25
26                         //DOWN
27                         case 1:
28
```

```
29                          if(i + 1 <= (L-1))
30                          {
31                                  if(grid[i+1][j].actions[action] != WALL)
32                                  {
33                                          this->nextState_i = i+1;
34                                          this->nextState_j = j;
35                                          if(i+1 == 0 && j == 3)
36                                                  grid[i][j].actions[action] =
                                                        qLearning.
                                                        Agent_LearningCalculation(grid[
                                                        i][j], grid[nextState_i][
                                                        nextState_j], action,
                                                        reward_goal);
37                                          else
38                                                  grid[i][j].actions[action] =
                                                        qLearning.
                                                        Agent_LearningCalculation(grid[
                                                        i][j], grid[nextState_i][
                                                        nextState_j], action,reward);
39                                          i++;
40                                          total_reward+=reward;
41                                  }
42                          }
43                          actions++;
44                  break;
45
46                  //LEFT
47                  case 2:
48
49                          if( j - 1 >= 0)
50                          {
51                                  if(grid[i][j-1].actions[action] != WALL)
52                                  {
53                                          this->nextState_i = i;
54                                          this->nextState_j = j-1;
55                                          if(i == 0 && j-1 == 3)
56                                                  grid[i][j].actions[action] =
                                                        qLearning.
                                                        Agent_LearningCalculation(grid[
                                                        i][j], grid[nextState_i][
                                                        nextState_j], action,
                                                        reward_goal);
57                                          else
58                                                  grid[i][j].actions[action] =
                                                        qLearning.
                                                        Agent_LearningCalculation(grid[
                                                        i][j], grid[nextState_i][
```

```
                                                    nextState_j], action,reward);
59                                          j--;
60                                          total_reward+=reward;
61                                  }
62                          }
63                          actions++;
64                  break;
65
66                  //RIGHT
67                  case 3:
68
69                          if(j + 1 <= (C-1))
70                          {
71                                  if(grid[i][j+1].actions[action] != WALL)
72                                  {
73                                          this->nextState_i = i;
74                                          this->nextState_j = j+1;
75                                          if(i == 0 && j+1 == 3)
76                                                  grid[i][j].actions[action] =
                                                      qLearning.
                                                      Agent_LearningCalculation(grid[
                                                      i][j], grid[nextState_i][
                                                      nextState_j], action,
                                                      reward_goal);
77                                          else
78                                                  grid[i][j].actions[action] =
                                                      qLearning.
                                                      Agent_LearningCalculation(grid[
                                                      i][j], grid[nextState_i][
                                                      nextState_j], action, reward);
79                                          j++;
80                                          total_reward+=reward;
81                                  }
82                          }
83                          actions++;
84                  break;
85
86          }
87
88          this->Enviroment_Reset(&i, &j);
89  }
```

No método *enviroment_loop* 4.6, é onde ocorre todo o controle do ambiente e interação com o agente. Primeiro o agente é passado como parâmetro, depois uma ação é escolhida pelo método de exploração que no caso é o ε-*greedy*, podendo essa ação ser o maior valor de uma ação existente em um estado ou uma ação randômica. Com isso, o ambiente verifica qual ação foi escolhida e entra nessa condição. No caso do teste, as ações podem ser "UP", "DOWN", "LEFT", "RIGHT". Depois de entrar na condição da ação escolhida, o ambiente realiza a transição de estados, o agente calcula a função Q para a ação escolhida e o ambiente verifica se aquele estado é o estado final. Pois se for o estado final, o método *enviroment_reset*é chamado, obrigando o agente voltar para o seu estado inicial.

Código Fonte 4.7: Método Enviroment_Reset

```
1 void ExemploEnviroment::Enviroment_Reset(int *i, int *j)
2 {
3         if((*i == 0 && *j == 3))
4         {
5                 count++;
6                 *i = 1;
7                 *j = 0;
8         }
9 }
```

Quando o método *enviroment_reset* 4.7 é chamado, ele verifica se o estado atual do agente é o final, se for um estado final, o agente é enviado para o estado inicial do do ambiente.

Métodos do Experimento:

Código Fonte 4.8: Método Experiment_Initialization

```
1 void ExemploEnviromentExperiment::Experiment_Initialization(float alpha, float gamma, float
      eGreedy, float lambda,float reward, float reward_goal, int numberActions)
2 {
3         this->agent.Agent_Initialization(alpha, gamma, eGreedy, lambda);
4         this->enviroment.Enviroment_Initialization(numberActions, reward, reward_goal);
5 }
```

O método *experiment_initialization* 4.8 é responsável por organizar as inicializações do agente e do ambiente.

Código Fonte 4.9: Método Experiment_ControlLoop

```
1  void ExemploEnviromentExperiment::Experiment_ControlLoop()
2  {
3
4          while(this->enviroment.count != convergence_time)
5          {
6                  this->enviroment.Enviroment_Loop(this->agent);
7          }
8
9          this->enviroment.printEnviroments.print(this->enviroment.grid);
10 }
```

Visualizando o método do *experiment_controlLoop* 4.9 é possível observar que esse método é responsável por controlar o *loop* principal do experimento, ou seja, controlar o número de passos para o mecanismo de aprendizagem atingir a convergência. Como esse ambiente de teste conta o número de passos no momento em que o estado final é alcançado, o ambiente informa ao experimento através da variável *"count"* se o estado final foi alcançado, se a resposta for verdadeira o método *experiment_controlLoop* incrementa "+1" ao contador. Isso se repete até o número máximo para a convergência for atingido.

Código Fonte 4.10: Método Experiment_Clean

```
1  void ExemploEnviromentExperiment::Experiment_Clean()
2  {
3          this->agent.~QLearning();
4          this->enviroment.~ExemploEnviroment();
5  }
```

O método *experiment_clean* 4.10 é responsável por liberar os espaços de memória utilizados no experimento. Nesse ambiente de teste, cada estado possui quatro ações. Essas ações são representadas como um arranjo dinâmico de número ponto flutuante. Então depois do experimento é liberado a memória desse arranjo de ações e das demais variáveis utilizadas. Como pode ser visualizado, para deixar o código mais organizado, os destrutores do agente e do ambiente são chamados.

É possível observar que a biblioteca possui um padrão de organização que facilita o desenvolvimento de novos experimentos. No momento que um novo agente for implementado ele pode ser passado por parâmetro e interagir diretamente com o ambiente, se um novo ambiente for criado ele pode ser integrado com um agente já existente e realizar o experimento. Dessa forma a reutilização de código agiliza o desenvolvimento de soluções para

novos problemas.

Depois que o agente atinge a convergência no ambiente ele obtêm a política descrita na Figura 4.6:

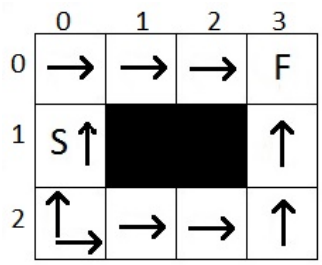

Figura 4.6: Política do Ambiente usado para aplicação prática da biblioteca

Visualizando a Figura 4.6, pode-se observar a sequência das melhores ações para o agente atingir o objetivo, sendo essa sequência: “UP”, “RIGHT”, “RIGHT”, “RIGHT”. Depois de mil interações com o ambiente o agente atinge a convergência com essas quatro ações.

Capítulo 5

Avaliação Experimental

Este capítulo irá descrever os ambientes utilizados para o desenvolvimento das análises e os resultados obtidos na comparação entre cada algoritmo no ambiente de teste.

5.1 Ambientes Usados nas Análises

Para realizar os testes relacionados a performance dos algoritmos propostos neste trabalho, foram escolhidos vários tipos de ambientes, cada um com sua particularidade em relação a um problema. Esses mesmos ambientes também foram implementados na biblioteca *AILibrary-RL*. Nas Figuras abaixo pode-se observar a descrição de cada um deles:

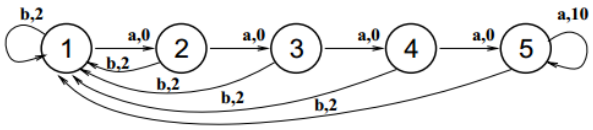

Figura 5.1: Ambiente Chain Domain

Na Figura 5.1, pode ser observado o ambiente descrito como *chain domain* [Dearden, Friedman e Russell 1998]. O mesmo consiste em uma sequência de cinco estados e duas ações *"a"* e *"b"*. A política ótima para esse domínio, é de sempre realizar à ação *"a"*. No entanto, esse ambiente possui uma pequena armadilha, no início da execução do sistema, o método de aprendizagem pode preferir escolher as ações *"b"*, pois com essa ação o agente fica recebendo uma série de pequenas recompensas e não observa a maior recompensa que está no último estado.

Algoritmos implementados no Ambiente:

- *Q-Learning*
- *SARSA*
- *Q-Learning* λ (*eligibility traces*)
- *SARSA* λ (*eligibility traces*)
- *Q-Learning* λ (*replacing*)
- *SARSA* λ (*replacing*)

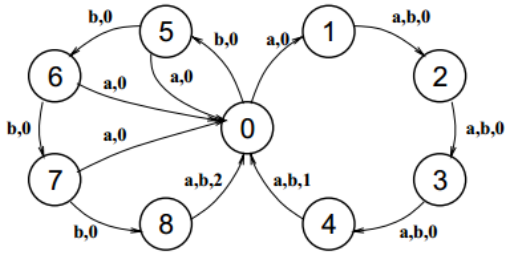

Figura 5.2: Ambiente Loop Domain

Na Figura 5.2, pode ser observado o ambiente descrito como *loop domain* [Dearden, Friedman e Russell 1998]. Esse ambiente consiste em oito estados divididos em dois *loops*, cada um com quatro estados retornando para o estado zero e duas ações *"a"* e *"b"*. O método de aprendizagem nesse ambiente pode demorar bastante para atingir a convergência, pois o mesmo pode ficar escolhendo as ações *"a"*, visualizadas no *loop* da esquerda na Figura 5.2 e demorar a encontrar a maior recompensa que se encontra no estado *"8"*. A política ótima para esse ambiente é o agente sempre escolher a ação *"b"*.

Algoritmos implementados no Ambiente:

- *Q-Learning*
- *SARSA*
- *Q-Learning* λ (*eligibility traces*)
- *SARSA* λ (*eligibility traces*)

- *Q-Learning* λ (*replacing*)
- *SARSA* λ (*replacing*)

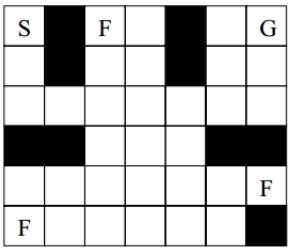

Figura 5.3: Ambiente Maze Domain

Na Figura 5.3, é observado o ambiente *Maze* [Dearden, Friedman e Russell 1998]. Ele é representado por uma matriz 7x6 e o agente pode realizar quatro ações: cima, baixo, esquerda e direita. O objetivo do agente nesse ambiente é recolher todas as bandeiras antes de encontrar o objetivo final. O *"S"* representa o estado inicial, *"F"* representa onde está localizada as bandeiras e *"G"* o estado final ou o objetivo a ser alcançando pelo agente. A recompensa recebida pelo agente é baseada no número de bandeiras que o mesmo conseguiu pegar antes de chegar no estado representado pelo objetivo final. Se o agente se mover em direção a parede o estado não é alterado.

Algoritmos implementados no Ambiente:

- *Q-Learning*
- *SARSA*
- *Q-Learning* λ (*eligibility traces*)
- *SARSA* λ (*eligibility traces*)
- *Q-Learning* λ (*replacing*)
- *SARSA* λ (*replacing*)
- *Dyna-Q*

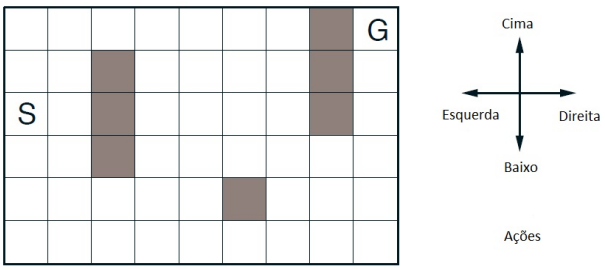

Figura 5.4: Ambiente Dyna Maze

Na Figura 5.4 é possível observar o ambiente *Dyna Maze* [Sutton e Barto 1998]. Ele é representado por uma matriz 9x6 e o agente pode realizar quatro ações: cima, baixo, esquerda e direita. O *"S"* representa o estado inicial, os quadrados mais escuros representam os obstáculos e *"G"* representa o estado final ou o objetivo a ser alcançando pelo agente. A recompensa fornecida a cada estado da execução do sistema é zero, no entanto, se o agente alcançar o estado *"G"* ele recebe *"+1"*.

Algoritmos implementados no Ambiente:

- *Q-Learning*
- *SARSA*
- *Q-Learning* λ (*eligibility traces*)
- *SARSA* λ (*eligibility traces*)
- *Q-Learning* λ (*replacing*)
- *SARSA* λ (*replacing*)
- *Dyna-Q*

Esses ambientes foram escolhidos pelas diferentes complexidades e desafios que aplicam ao método de aprendizagem por reforço e por serem utilizados no meio acadêmico para realizar testes inerentes a essa problemática [Sutton e Barto 1998], [Russell e Norvig 2009] [Dearden, Friedman e Russell 1998].

5.2 Ferramentas e Tecnologias

As ferramentas e tecnologias utilizadas no processo de desenvolvimento deste trabalho, podem ser visualizadas na lista abaixo:

- Os ambientes descritos anteriormente e a biblioteca foram implementados na linguagem *C/C++*, pois a mesma possui um bom desempenho por combinar características de linguagens de baixo e alto níveis.
- O ambiente de programação utilizado foi o Visual Studio 2010.

5.3 Resultados Experimentais

Esta seção possui o objetivo de apresentar os resultados obtidos na implementação[1] dos mecanismos de aprendizagem e ambientes previamente apresentados. A partir dos dados e gráficos obtidos nos testes, será possível visualizar o desempenho e comparação de cada um deles nos ambientes.

Para efetuar a comparação entre os algoritmos de aprendizagem por reforço, foi executada um conjunto de estratégias já existentes e usadas na literatura, visando a obtenção dos dados do ambiente e com isso, a realização da análise estatística. Abaixo encontra-se listadas essas estratégias.

- Cada ambiente foi executado um número máximo de vezes, sem mudança nos parâmetros dos algoritmos de aprendizagem. Essa estratégia tem por objetivo obter a média desse conjunto de execuções.
- Cada experimento foi limitado a um número fixo de episódios, visando obter os dados dentro de um limite fechado e não em qualquer limite.

Já para verificar o desempenho dos algoritmos de AR, foram utilizadas duas métricas que também são bastante usadas na literatura, são elas [Russell e Norvig 2009] [Sutton e Barto 1998]:

[1] Os códigos relacionados a cada mecanismos e ambientes podem ser visualizados no github:https://github.com/thiagorm/Environments

- Verificar em cada episódio da execução do mecanismo de aprendizagem, quantas ações o agente realizou para chegar no objetivo.

- Somar a cada passo da execução do sistema a recompensa obtida pelo agente, com o objetivo de obter o acumulo total.

A primeira opção de verificação de desempenho será utilizada nos ambientes de tarefas episódicas, onde a cada episódio é somado a quantidade de ações que o agente realizou para chegar no objetivo. A tendência dos valores dessa soma é baixar a cada episódio, pois quanto mais vezes o agente encontra o objetivo, mais ele conhece o caminho.

A segunda opção de verificação de aprendizagem será utilizada em ambientes de tarefas contínuas, onde não existe um estado final como objetivo para o agente, consequentemente não existe episódio. Geralmente nesses ambientes a verificação é realizada através de qual mecanismo alcança a política ótima primeiro ou do acumulo de recompensa. [Poole e Mackworth 2010].

O melhor método de avaliação a ser escolhido é de acordo com o funcionamento do ambiente, pois se o ambiente possui episódios e segurança de convergência para uma política ótima, a avaliação por ações deve ser a mais segura. É importante ressaltar que se a avaliação dos mecanismos for realizada através do método de averiguar quem atingiu a política ótima primeiro, ela pode esta errada se o número de episódios ainda não estiver terminado, pois o sistema irá continuar executando e o agente ainda pode ficar obtendo recompensas menores e não convergir no final [Poole e Mackworth 2010] [Dearden, Friedman e Russell 1998].

5.3.1 Análise das ações com $\varepsilon-$*greedy* e *SoftMax*

O método utilizado no trabalho para realizar os testes será o ε-greedy, por sua grande utilização pelos autores na literatura [Russell e Norvig 2009] [Sutton e Barto 1998]. No entanto, nesta seção será apresentado uma comparação entre o ε-*greedy* e *softmax* utilizando como ambiente o *Dyna Maze*. Essa análise, possui o objetivo de mostrar seus desempenhos, para que a avaliação entre cada algoritmo possa ser realizada. Na figura 5.5 pode ser observado, a comparação nos mecanismos de aprendizagem *Q-Learning* e *SARSA* respectivamente.

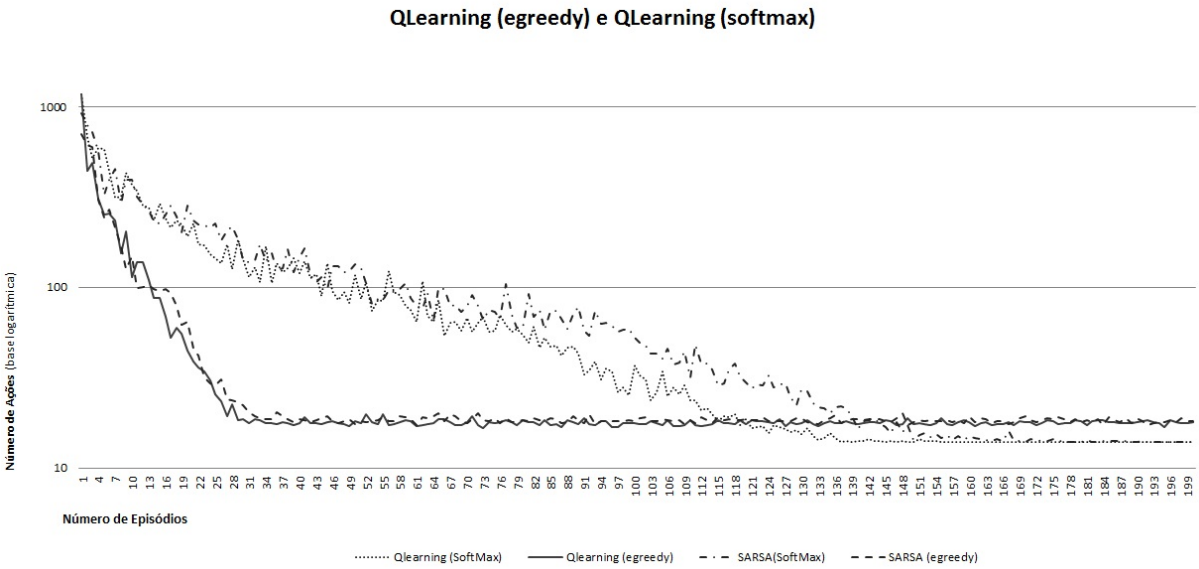

Figura 5.5: Gráfico análise mecanismos *Q-Learning* e *SARSA* com ε*-greedy* e *Softmax*

Visualizando a Figura 5.5 é possível concluir que a eficiência de um algoritmo de aprendizagem por reforço no ambiente *Dyna Maze* pode ser definida a partir do método de exploração que está implementado no mecanismo. Com isso, o algoritmo de aprendizagem que esteja implementado com o método ε*-greedy* no ambiente *Dyna Maze* executará menos ações por episódio e assim será mais eficiente.

Já o método de exploração *softmax* demora mais para atingir a convergência. No entanto, ao convergir estabiliza-se em uma solução ótima, sem uma grande variação nas ações. Como pode ser visualizado no quadrado vermelho da Figura 5.6.

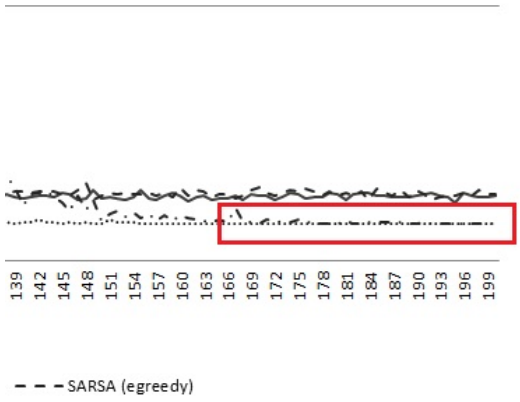

Figura 5.6: Variação de ações do mecanismo *Softmax*

5.3.2 Análise no ambiente *Dyna Maze*

O ambiente *Dyna maze* foi proposto por *Sutton* [Sutton e Barto 1998], com a finalidade de realizar testes de desempenho no algoritmo *Dyna-Q*. Ele será utilizado neste trabalho por fornecer características que testem o desempenho de um mecanismo de aprendizagem por reforço e por ter sido utilizado por um autor conhecido no tema para aferir testes com esses algoritmos.

A estratégia utilizada para a elaboração dos testes foi executar o ambiente "trinta vezes", cada uma com "mil e setecentos episódios". De cada episódio, foi extraído o número de ações realizadas pelo agente para alcançar o objetivo. Depois, foi tirada a média entre cada uma das ações das trinta amostras.

A escolha na quantidade de episódios a serem executados no ambiente, foi realizada de acordo com número necessário que o agente precisou para alcançar a política ótima. O objetivo nesse ambiente é o agente sair de um ponto ou estado inicial e conseguir chegar no estado final.

Parâmetros utilizados no *Dyna Maze*, são os mesmos referenciados por *Sutton* [Sutton e Barto 1998]:

- α (taxa de aprendizagem) **0.1**
- γ (taxa de desconto) **0.95**
- ϵ (taxa de exploração) **0.1**
- λ (fator de rastreamento) **0.01**

Nas seções abaixo serão apresentados os resultados obtidos dos testes no ambiente *Dyna-Q*.

Q-Learning vs SARSA

Os primeiros algoritmos a serem comparados é o *Q-Learning* e o *SARSA*. O primeiro atualiza sua matriz de estado-ações, levando em consideração a maior ação do próximo estado e o segundo atualiza sua função estado-ações, escolhendo as ações de acordo com a política do método de exploração.

Na Figura 5.7 pode ser visualizado o gráfico de comparação entre os mecanismos *Q-Learning* e *SARSA* no Ambiente *Dyna Maze*:

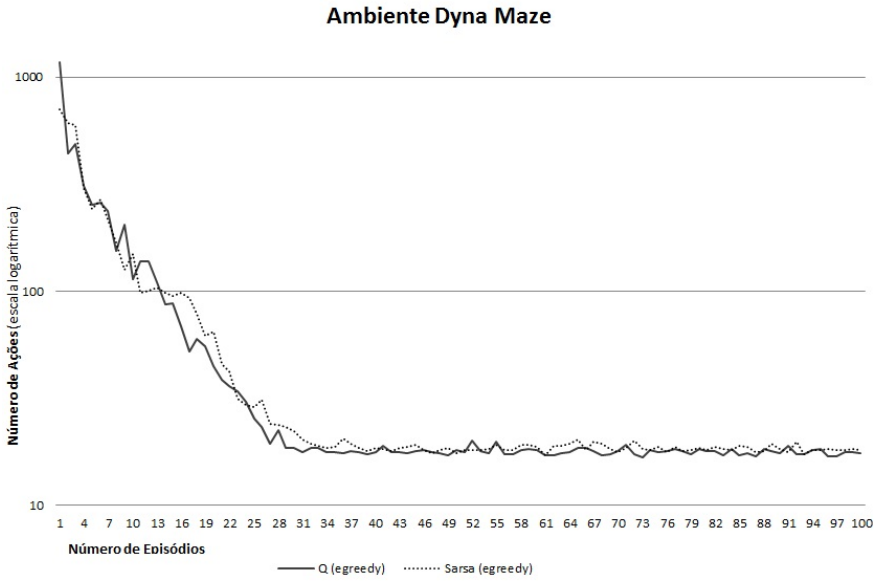

Figura 5.7: Gráfico de análise do *Q-Learning* e *SARSA* com *egreedy*, Ambiente *Dyna Maze*

É possível notar observando a Figura 5.7 a similaridade nos dados entre os mecanismos. No entanto, com um pouco mais de atenção pode ser visto que o *Q-Learning* algumas vezes, mesmo que em uma escala pequena, obtêm uma vantagem em relação ao método *SARSA*. Com isso, é possível concluir que em alguns episódios o método *Q-Learning* utilizou menos ações para chegar ao objetivo.

Eligibility Traces λ

A técnica de *Eligibility Traces* foi implementada no *Q-Learning* e *SARSA* com o objetivo de melhorar seus desempenhos a partir da marcação de determinados eventos (a passagem do agente por determinados estados-ações). É importante ressaltar que esse conceito gera uma carga de processamento considerável, pois a cada episódio os estados marcados são atualizados pelo mecanismo. Nesta seção será possível observar as análises e comparações realizadas entre essa técnica e seus respectivos algoritmos bases.

Nas Figuras 5.8 e 5.9 podem ser visualizados os gráficos de comparação para os algoritmos *Q-Learning* λ e *SARSA* λ no ambiente *Dyna Maze*:

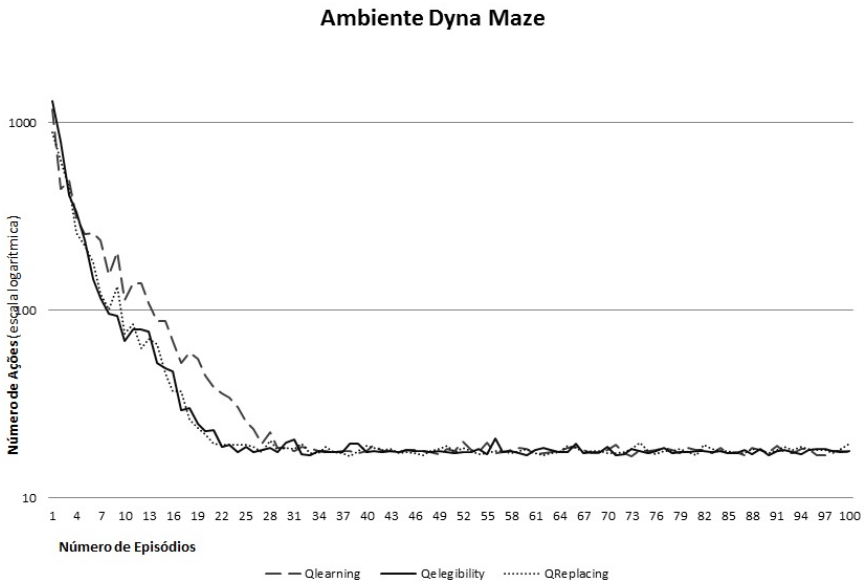

Figura 5.8: Gráfico da análise do *Q-Learning*, *eligibility traces* e (*replacing*), ambiente *Dyna Maze*

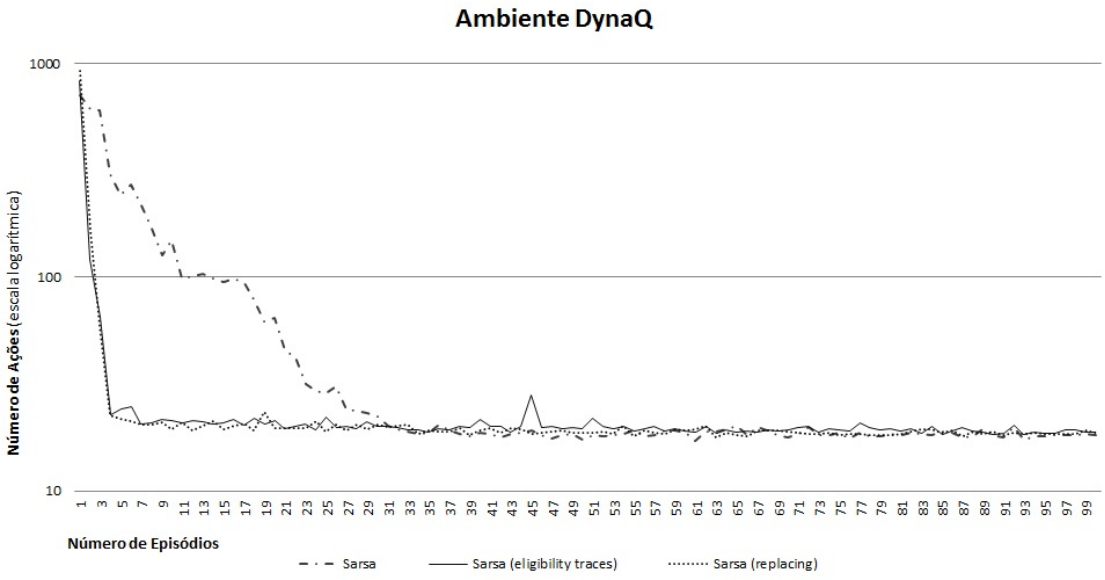

Figura 5.9: Gráfico da análise do *SARSA*, *eligibility traces* e (*replacing*), ambiente *Dyna Maze*

Inicialmente é possível observar que os mecanismos *Q-Learning* e *SARSA* implementados com a técnica de *eligibility traces* possui maior eficiência do que o mecanismos originais, necessitando de menos ações para atingir o objetivo. Também é de se notar, que depois dos algoritmos atingirem a convergência, eles passam a realizar a mesma média de número de ações por episódio.

Assim observa-se que a técnica de *eligibility traces* de marcação de eventos, realmente aumenta a eficiência do algoritmo, fazendo o agente conhecer o caminho até o objetivo mais rápido a cada episódio.

Dyna-Q

O mecanismo *Dyna-Q* propõe uma nova forma de solucionar o problema de AR. A partir de um modelo do mundo idêntico ao ambiente do agente, esse mecanismo simula através de uma variável de controle, passos hipotéticos a partir dos estados previamente visitados por ele. Elaborando assim, uma forma de planejamento para o agente.

Na Figura 5.10 pode ser observado o gráfico de comparação com a variável de controle *n* do mecanismo *Dyna-Q* no ambiente *Dyna Maze*:

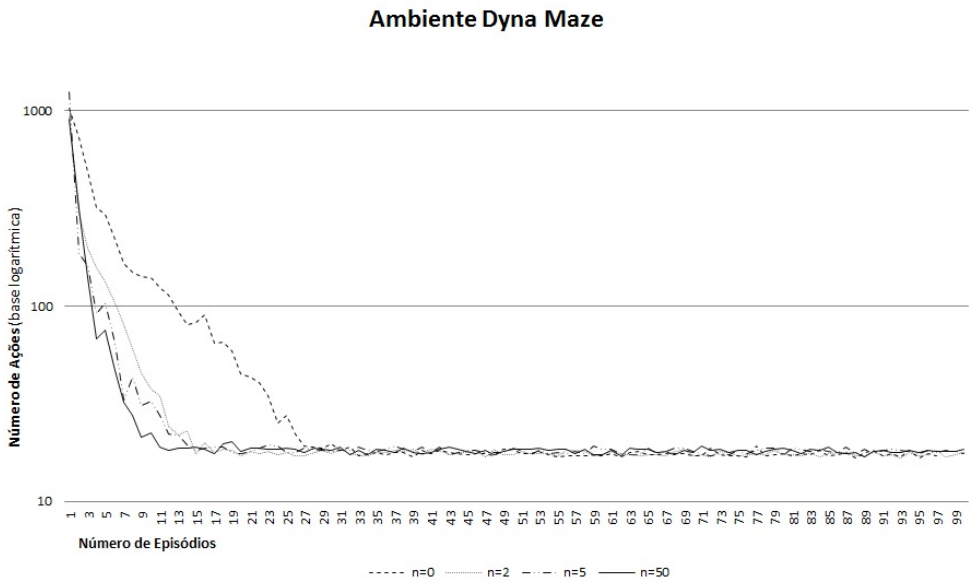

Figura 5.10: Gráfico da análise do *DynaQ* no ambiente **Dyna Maze**

É possível visualizar na Figura 5.10 o gráfico com a análise desse mecanismo. Com a variável de controle *n* igual a "zero", o mecanismo segue o mesmo desempenho do *Q-Learning* original, o que já era previsto. No entanto, a partir do momento em que *n* é incrementado, o algoritmo vai ganhando um desempenho melhor, ou seja, seu planejamento permite que o agente realize menos ações por episódio para chegar ao estado final.

Conclusão da Análise no Ambiente *Dyna Maze*

Depois da análise dos mecanismos no ambiente *Dyna Maze*, é possível observar que o *SARSA (eligibility traces)*, *SARSA (replacing)* e *Dyna-Q*, alcançaram os melhores resultados. Então, para verificar qual desses mecanismo é o mais eficiente no ambiente *Dyna Maze, o SARSA (eligibility traces) e o Dyna-Q*, com a variável de controle *n* igual a "cinquenta", foram comparados. Essa avaliação pode ser observada no gráfico da figura 5.11:

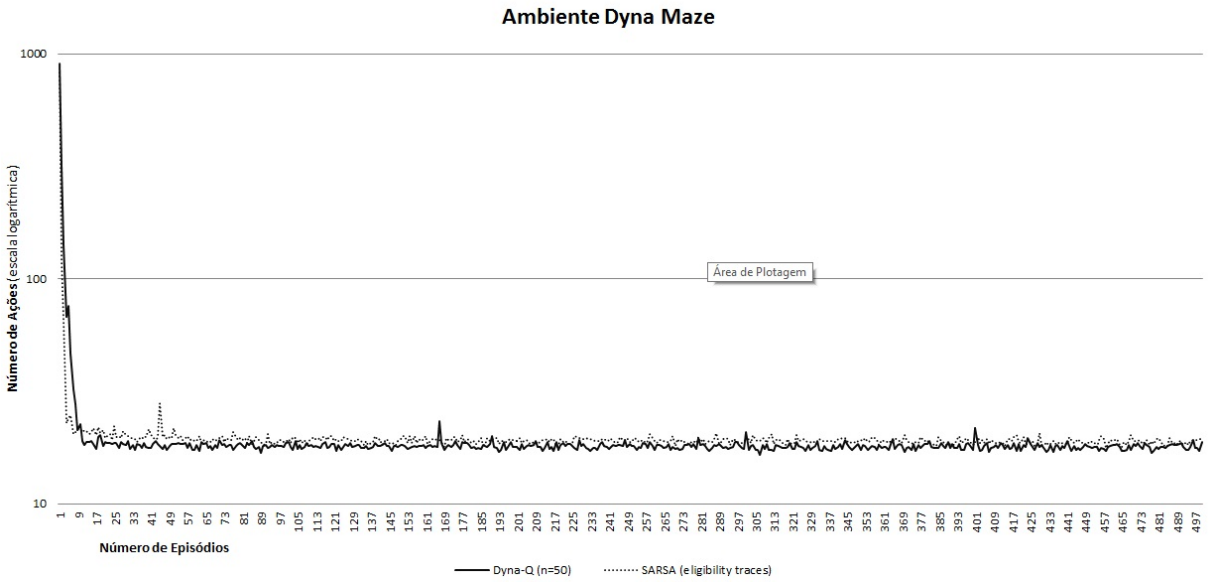

Figura 5.11: Comparação dos resultados entre o *SARSA (eligibility traces)* e o *Dyna-Q*, Ambiente *Dyna Maze*

Observando o gráfico da Figura 5.11, pode-se concluir que o mecanismo de aprendizagem por reforço que alcançou uma convergência mais rápida, conseguindo com menos ações por episódio conhecer o objetivo final do ambiente *Dyna Maze* foi o *SARSA (eligibility traces*), através dos registros temporários de marcações de eventos (visita em determinado estado-ação).

Assim, é possível concluir que em ambientes do tipo *tilemap*, onde os obstáculos e objetivos a serem alcançados são estáticos e a métrica utilizada, é verificar a quantidade de passos por episódio que o agente realiza para chegar ao objetivo, a variação do algoritmo *SARSA* o *SARSA* λ pode obter melhor desempenho que os outros algoritmos analisados neste trabalho. Pois, suas características de marcação de eventos e atualização da função Q a partir do método de exploração implementado, lhe fornece maior vantagem em encontrar o estado objetivo mais rápido.

5.3.3 Análise no Ambiente *Chain Domain*

O Ambiente *Chain Domain* foi proposto por *Dearden* [Dearden, Friedman e Russell 1998], para análise e comparação de alguns algoritmos de AR. Neste ambiente, os algoritmos serão comparados de acordo com o acumulo de recompensa obtido a cada passo da execução do

sistema. Possuindo o melhor desempenho o mecanismo que acumulou mais recompensa.

A estratégia utilizada para a elaboração dos testes foi executar o ambiente "trinta vezes", cada uma com "três mil episódios". De cada passo executado pelo sistema foi extraído a soma das recompensas obtidas até aquele momento. Depois, foi tirada a média entre as recompensas de cada passo das trinta amostras.

Os Parâmetros utilizados nos mecanismos para serem avaliados no ambiente *Chain Domain*, foram escolhidos de acordo com a avaliação feita a partir da tabela na Figura 5.12 e os gráficos da Figura 5.13. O algoritmo utilizado para efetuar essa avaliação foi o *Q-Learning*.

		Parâmetros	
Experimento	ε-greedy	γ	α
A	0,2	0,1	0,1
B	0,2	0,1	0,9
C	0,2	0,9	0,1
D	0,2	0,9	0,9
E	1	0,1	0,1
F	1	0,1	0,9
G	1	0,9	0,1
H	1	0,9	0,9

Figura 5.12: Tabela com a variações dos parâmetros dos mecanismos de aprendizagem

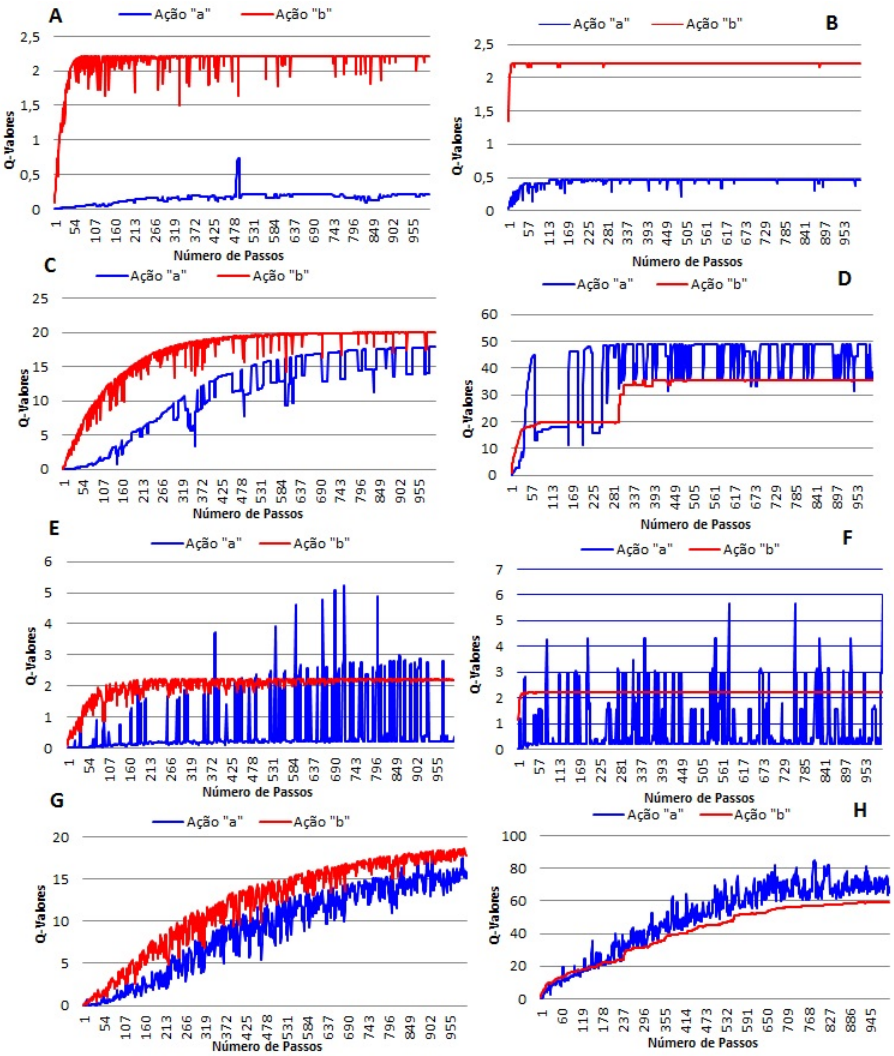

Figura 5.13: Experimento com a variação dos parâmetros dos mecanismos no ambiente *Chain Domain*

A tabela da Figura 5.12 exibe os parâmetros para cada um dos gráficos expostos na Figura 5.13, de acordo com as letras na coluna “experimento” da tabela.

Cada experimento foi executado “oito vezes” e a cada passo foi salvo o acumulo da função estado-ação Q(s,a) das ações do ambiente. Essa estratégia têm o objetivo de obter a aproximação dos Q-Valores entre as ações, já que a política ótima desse ambiente é de sempre executar as ações *a*.

Comparando as curvas nos experimentos A e E, onde os valores de ε-greedy são respectivamente 0,2 e 1,0, percebe-se que quanto maior o valor desse parâmetro mais pertubação a curva sofre, aumentando assim a exploração do ambiente.

Quando o fator de desconto γ é baixo, experimentos B e F, nota-se uma grande importância com as recompensas a curto prazo, ou seja uma maior valorização das ações *b*. O uso de γ 0,9 aproxima os Q-Valores das ações *a* e *b*, atribuindo mais importância as recompensas a longo prazo, experimentos C, D, G e H.

Com os valores da taxa de aprendizagem α baixos, os Q-Valores tendem a convergir de maneira mais instáveis. Atribuindo taxas menores para esse parâmetro, a instabilidade tende a diminuir.

O experimento que mais aproximou as duas ações foi o H. Nesse caso, a curva dos Q-Valores da ação *a* ultrapassou de forma mais estável a curva dos Q-Valores da ação *b*, mostrando assim uma melhor convergência e alcançando a política ótima do ambiente, já que os valores das ações *a*, são mais altos. [Crook e Hayes 2003]

Observando essa análise os parâmetros escolhidos para serem utilizados no *Chain Domain* foram:

- α (taxa de aprendizagem) **0.8**
- γ (taxa de desconto) **0.99**
- ϵ (taxa de exploração) **0.8**
- λ (fator de rastreamento) **0.1**

Abaixo será exibida as análises e comparações realizadas no ambiente *Chain Domain*.

Q-Learning vs SARSA

Na Figura 5.14 pode ser visualizado o gráfico de comparação dos mecanismo *Q-Learning* e *SARSA* no ambiente *Chain Domain*:

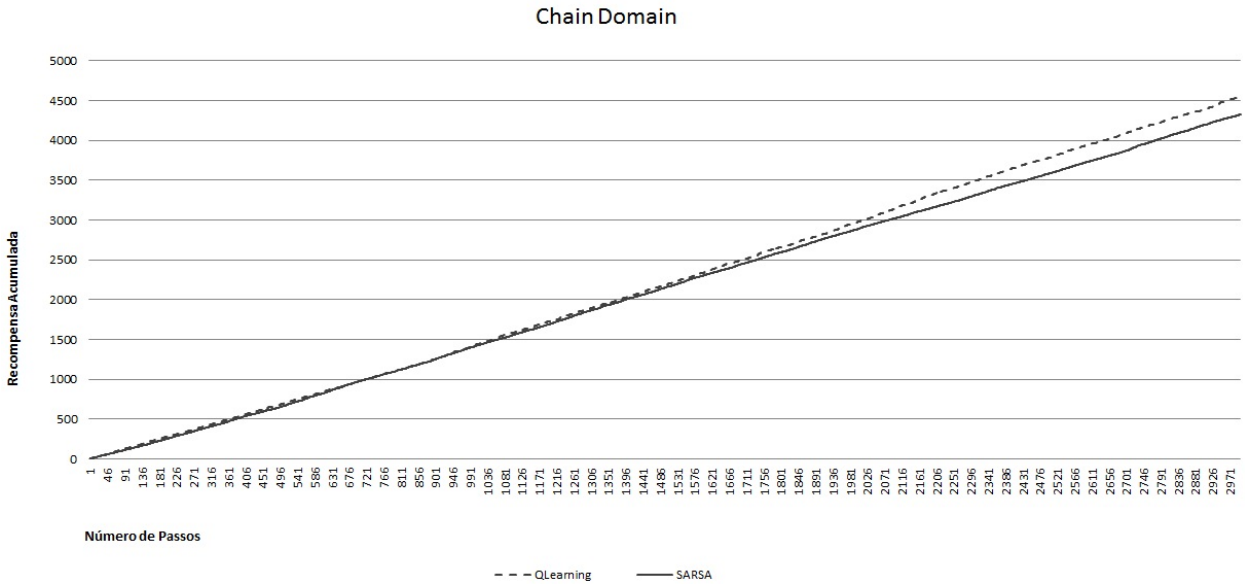

Figura 5.14: Gráfico análise *Q-Learning* e *SARSA* ambiente *Chain Domain*

É percebido na Figura 5.14 que o mecanismo *Q-Learning* obtêm uma vantagem na quantidade de recompensas obtidas por passos na execução do ambiente *Chain Domain*, ou seja, ele acumula mais recompensa que o mecanismo *SARSA*, conhecendo assim cada vez mais o ambiente e possuindo maiores chances de atingir a política ótima.

Esse resultado pode ser atribuído a característica da função de atualização do Q-Learning usar como referência o maior valor da ação do estado, isso combinado com um fator de desconto e ε-*greedy* alto, permite ao agente acumular as recompensas dos valores de "dois" e depois convergir em direção a de valor "dez". Já a chance do mecanismo *SARSA* atualizar sua função estado-ação com os valores das maiores recompensas é baixa, pois sua função *Q(s,a)* depende parâmetro ε-*greedy* do método de exploração.

Eligibility Tracesλ

Nas Figura 5.15 e 5.16 podem ser observados o gráfico de comparação dos mecanismos *Q-Learning* λ e *SARSA* λ no ambiente *Chain Domain*:

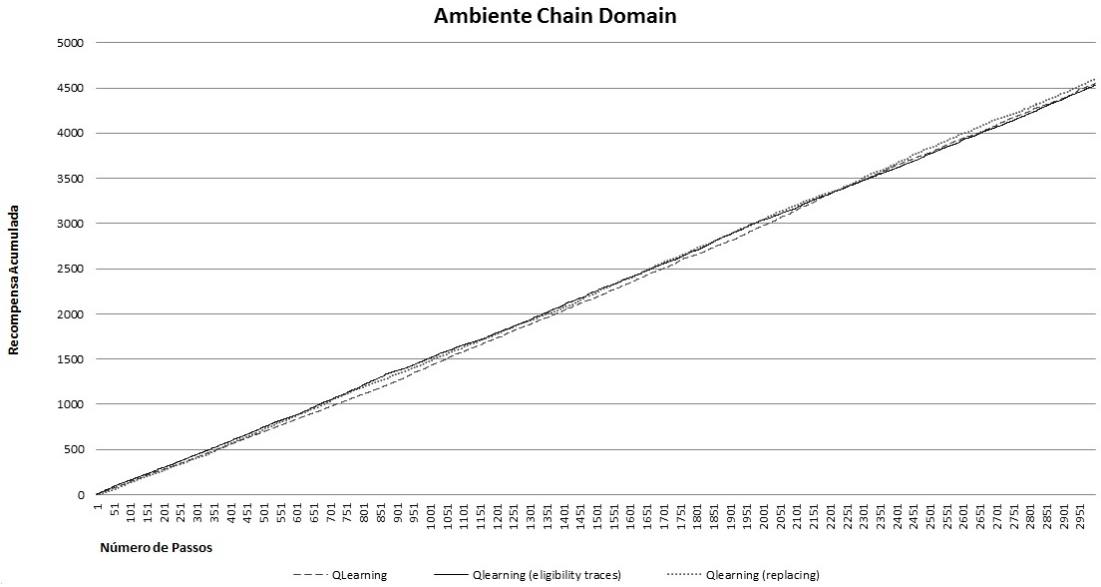

Figura 5.15: Gráfico análise *Q-Learning* λ ambiente *Chain Domain*

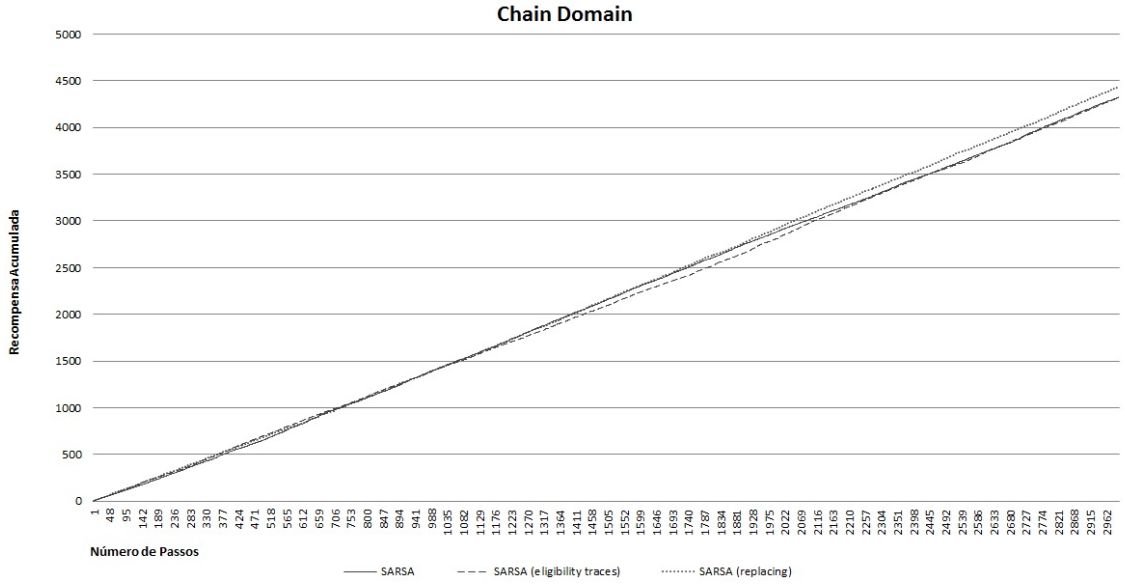

Figura 5.16: Gráfico análise *SARSA* λ ambiente *Chain Domain*

De acordo com os gráficos das Figuras 5.15 e 5.16, pode-se perceber a similaridade entre os resultados dos mecanismos. No entanto, observando mais atentamente é possível notar que em alguns momentos, os mecanismos *eligibility traces* e *replacing* possuem uma pequena vantagem entre o original *Q-learning*.

Já no do gráfico do *SARSA* o mecanismo *replacing* chega a obter uma vantagem na sua variação de acumulo de recompensa.

Conclusão da Análise no Ambiente *Chain Domain*

Na avaliação dos mecanismos no ambiente *Chain Domain*, é possível observar que os algoritmos *Q-Learning (replacing)* e *SARSA (replacing)*, mesmo com uma vantagem pequena, alcançaram bons resultados em relação aos outros. Então, para verificar qual desses mecanismo é o mais eficiente no ambiente *Chain Domain*, foi realizada uma comparação entre eles. Essa avaliação pode ser observada no gráfico da figura 5.17:

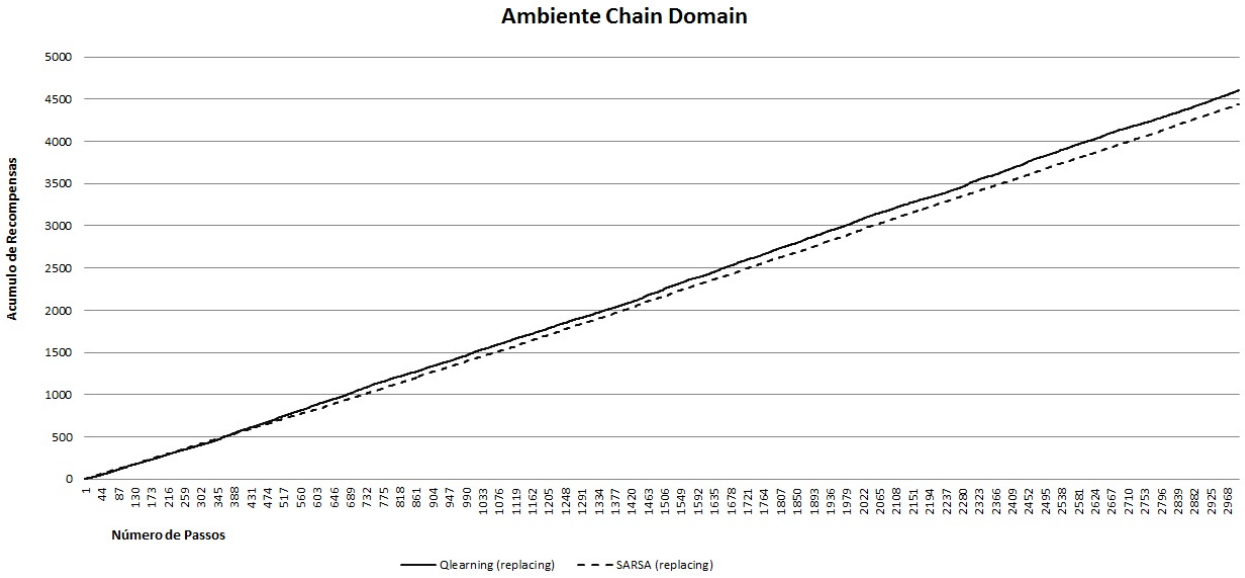

Figura 5.17: Comparação entre os Algoritmos *QLearning (replacing)* e *SARSA(replacing)*, Ambiente *Chain Domain*

De acordo com gráfico da Figura 5.17, pode-se concluir que o mecanismo de aprendizagem por reforço que alcançou uma convergência mais rápida, conseguindo obter uma quantidade maior de recompensas por passos na execução do Ambiente *Chain Domain*, foi o *QLearning (replacing*). Através dos registros temporários de marcações de eventos (visita em determinado estado-ação) e sua característica marcação unitária e não incremental.

5.3.4 Análise no Ambiente *Loop Domain*

O Ambiente *Loop Domain* também foi utilizado por *Dearden* [Dearden, Friedman e Russell 1998], em seu artigo. O objetivo nesse ambiente é atingir a política ótima de sempre realizar as ações *b*. O desempenho dos algoritmos será medido de acordo com o acumulo de recompensa a cada passo da execução do sistema.

A estratégia utilizada para a elaboração dos testes foi executar o ambiente "trinta vezes", cada uma com "três mil episódios". De cada passo foi extraído a soma da recompensa até aquele momento. Depois, foi tirada a média entre cada recompensa das "trinta" amostras.

Os Parâmetros utilizados nos mecanismos para serem avaliados no ambiente *Loop Domain*, também foram escolhidos de acordo com a avaliação feita a partir da tabela na Figura 5.12 e os gráficos da Figura 5.18. O algoritmo utilizado para efetuar essa avaliação foi o

Q-Learning.

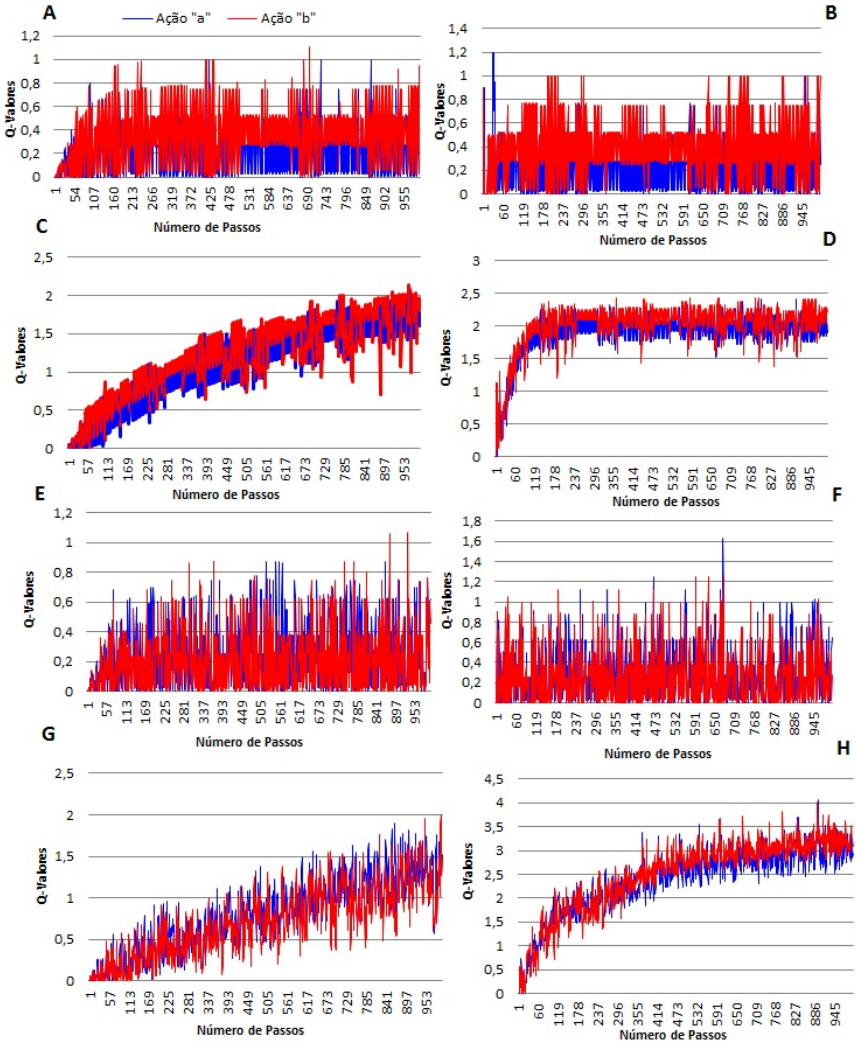

Figura 5.18: Experimento com a variação dos parâmetros dos mecanismos no ambiente *Loop Domain*

Cada experimento também foi executado "oito vezes" e a cada passo foi salvo o acumulo da função estado-ação *Q(s,a)* das ações do ambiente. Essa estratégia têm o objetivo de obter a aproximação entre os Q-Valores, pois a política ótima para esse ambiente é sempre executar as ações *b*.

Avaliando os gráficos da Figura 5.18 é possível perceber que as mesmas situações ocorridas no ambiente *Chain Domain* se repetem no *Loop Domain*. Quanto maior o valor de ε-greedy, mais exploração o agente irá realizar no ambiente e consequentemente os Q-Valores serão bem mais distribuídos. Atribuindo valores altos ao parâmetro γ, mais próximo será a

curva de convergência entre os Q-valores de *a* e *b*. Com uma taxa de aprendizagem baixa maior será a instabilidade das curvas e menor será velocidade de aprendizagem.

No entanto é possível perceber uma certa similaridade entre a convergência do gráfico D e H. Os Q-Valores parecem convergir por iguais nas duas ações. Então, como fato de esclarecimento foi pego os Q-Valores das ações *b* do gráfico D e comparado com os Q-Valores das ações *b* do gráfico H. Já que esses são os únicos em que as curvas de convergência são estáveis e os valores das ações *b* parecem mais altos que as ações *a*.

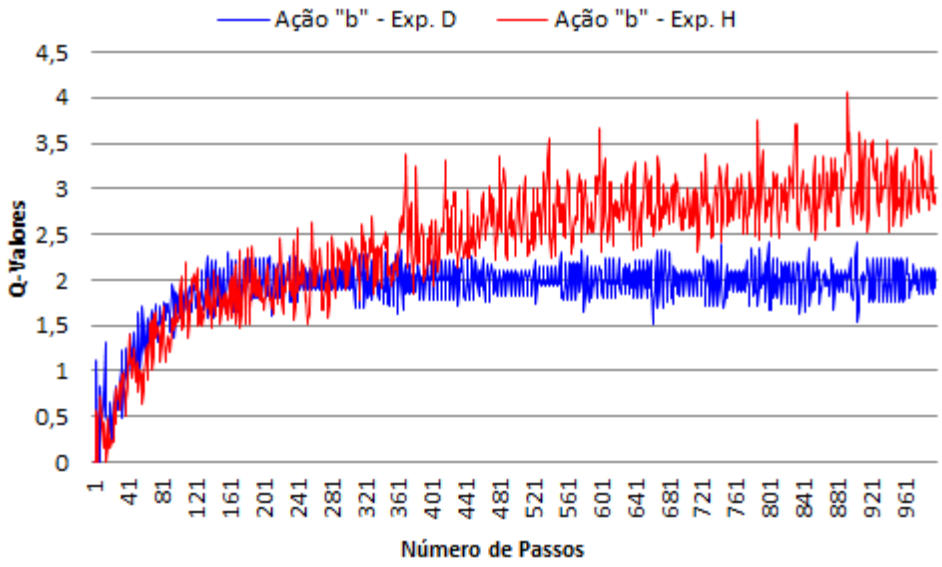

Figura 5.19: Experimento com os experimentos D e H no ambiente *Loop Domain*

De acordo com a Figura 5.19 fica claro que os Q-Valores das ações *b* do gráfico H são maiores que as do gráfico D. Assim, confirmando a melhor convergência dos Q-Valores das ações para o gráfico H.

De acordo com a análise os parâmetros utilizados no *Loop Domain* foram:

- α (taxa de aprendizagem) **1**
- γ (taxa de desconto) **0.99**
- ϵ (taxa de exploração) **0.8**
- λ (fator de rastreamento) **0.1**

Q-Learning vs SARSA

É Possível visualizar na Figura 5.20 o gráfico com o resultado da análise entre os algoritmos *Q-Learning* e *SARSA* no ambiente *Loop Domain*:

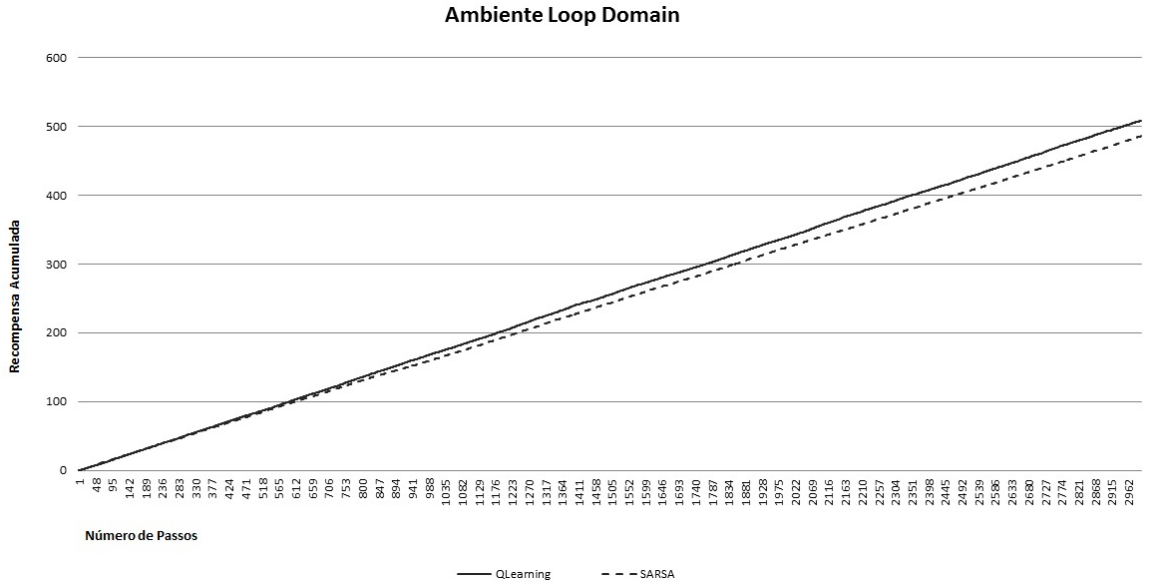

Figura 5.20: Gráfico análise *Q-Learning* e *SARSA* Ambiente *Loop Domain*

Observando o gráfico é possível notar a vantagem da reta do algoritmo *Q-Learning* em relação a do *SARSA*. Com isso, é possível afirmar que a recompensa acumulada pelo mecanismo *Q-Learning* foi maior no ambiente *Loop Domain*. Esse resultado também pode ser relacionado com o modo como os mecanismos atualizam sua função ação-valor, sendo o *Q-Learning* pelo valor da ação mais alta do estado e o *SARSA* de acordo com o parâmetro do método de exploração. Assim, como no *Chain Domain*.

Eligibility Traces λ

É Possível visualizar nas Figuras 5.21 e 5.22 os gráficos com os resultados das análises entre os algoritmos *Q-Learning* λ e *SARSA* λ para o ambiente *Loop Domain*:

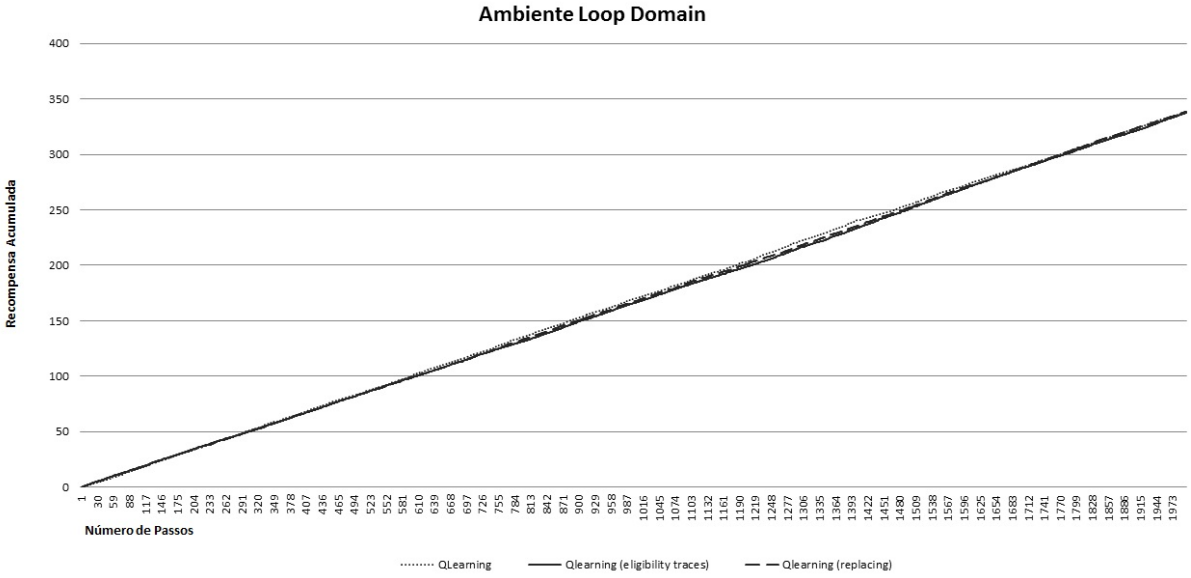

Figura 5.21: Gráfico análise *Q-Learning* λ ambiente *Loop Domain*

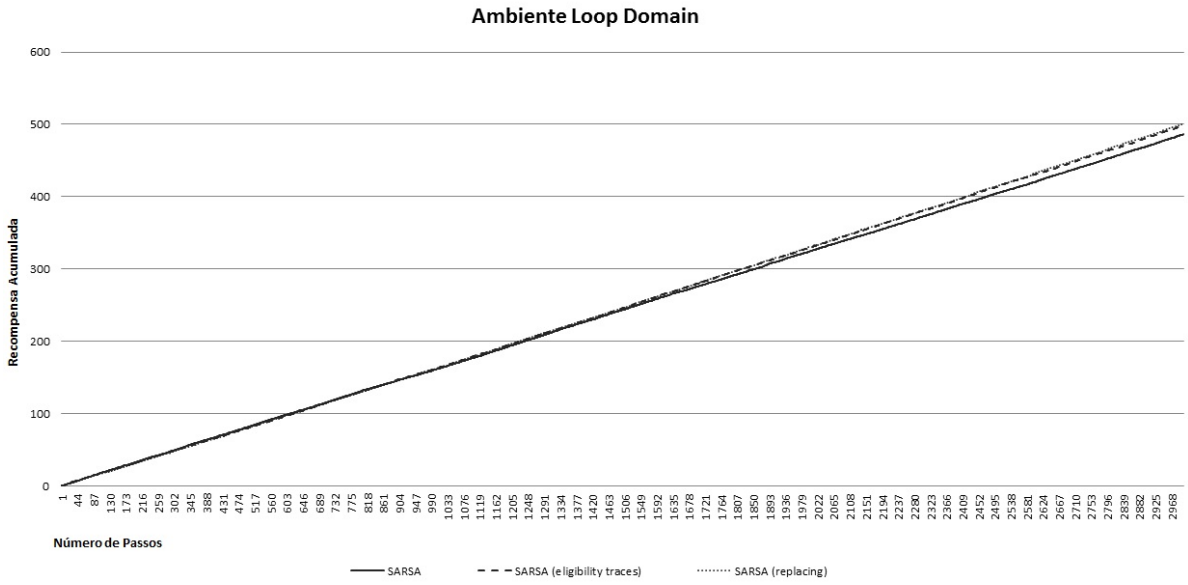

Figura 5.22: Gráfico análise *SARSA* λ ambiente *Loop Domain*

Observando os resultados obtidos é claramente visualizado a partir das retas do gráfico 5.21 que as recompensas acumuladas obtidas a cada passo da execução do sistema são muito parecidas. Em alguns pontos do gráfico, o mecanismo *Q-Learning* parece estar obtendo certa vantagem, no entanto podemos concluir que a recompensa acumulada neste ambiente, por esses algoritmos são muito similares.

No *SARSA* já é possível notar uma vantagem mais acentuada no acumulo de recompensa dos mecanismos *eligibility traces* e *replacing*.

Conclusão da Análise no Ambiente *Loop Domain*

Na avaliação dos mecanismos do ambiente *Loop Domain*, é possível observar que mesmo adquirindo uma vantagem pequena, os algoritmos *Q-Learning* e *SARSA (replacing)* alcançaram bons resultados em relação aos outros. Com o objetivo de verificar qual desses mecanismos é o mais eficiente, foi realizada uma comparação entre eles. Essa avaliação pode ser observada no gráfico da figura 5.23:

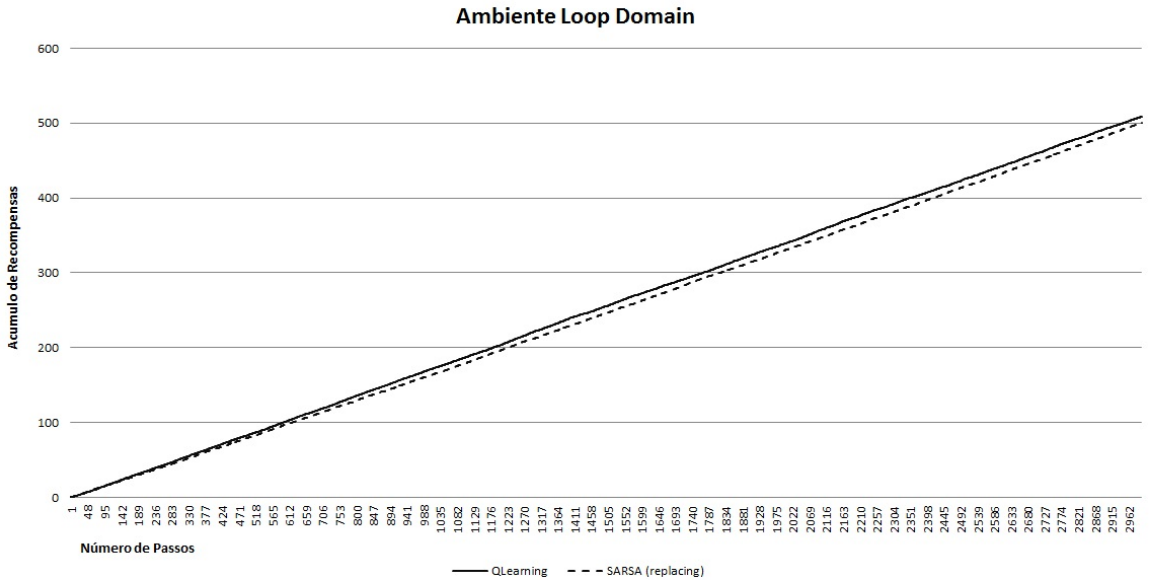

Figura 5.23: Comparação entre os Algoritmos *QLearning* e *SARSA(replacing)*, Ambiente *Loop Domain*

É possível visualizar no gráfico da Figura 5.23, que o mecanismo de aprendizagem por reforço mais eficiente foi o *QLearning*, alcançando uma convergência mais rápida e conseguindo obter uma quantidade maior de recompensas por passos na execução do ambiente *Loop Domain*, através de sua característica de atualizar sua função estado-ação *Q(s,a)*, com o valor da maior ação do próximo estado.

5.3.5 Análise no Ambiente *Maze Domain*

O Ambiente *Maze Domain* também foi utilizado por *Dearden* [Dearden, Friedman e Russell 1998]. O objetivo neste ambiente é tentar coletar todas as bandeiras antes de chegar no objetivo ou estado final. A recompensa será de acordo com o número de bandeiras coletadas, sendo o máximo de "três".

A estratégia utilizada para a elaboração dos testes foi executar o ambiente "dez vezes", cada uma com "vinte mil episódios". De cada episódio, foi extraído o número de bandeiras obtidas pelo agente antes de alcançar o objetivo. Depois, foi tirada a média entre cada um dos valores das "dez amostras".

Os Parâmetros utilizados nos mecanismos para serem avaliados no ambiente *Maze Domain*, também foram escolhidos de acordo com a avaliação da tabela da Figura 5.12 e o

gráfico da Figura 5.24. O algoritmo utilizado para efetuar essa avaliação foi o *Q-Learning*.

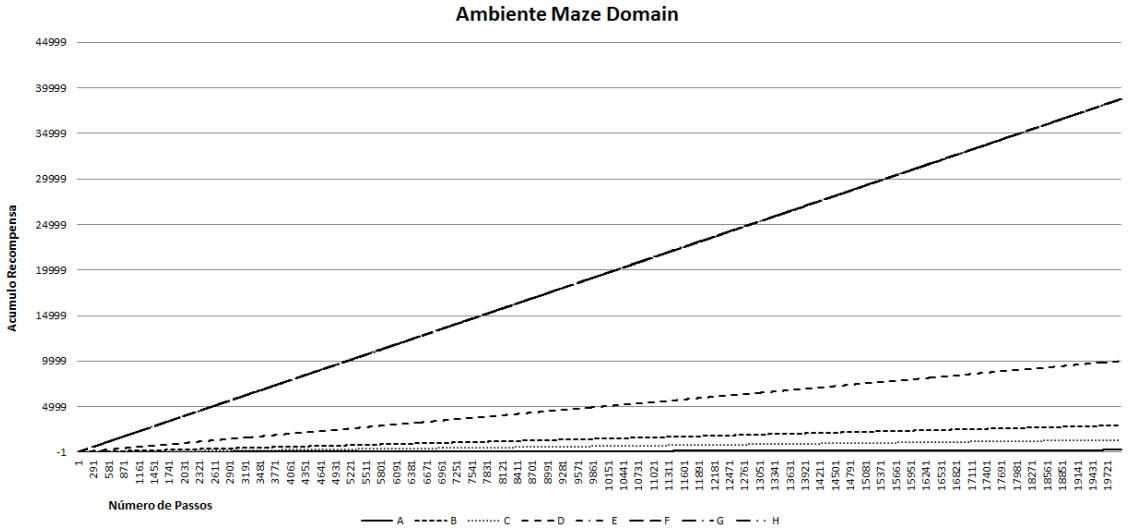

Figura 5.24: Experimento com a variação dos parâmetros dos mecanismos no ambiente *Maze Domain*

Para esse experimento, o ambiente também foi executado "oito vezes" e a cada passo foi salvo o acumulo do total de bandeiras recolhidas pelo agente, já que o objetivo nesse ambiente é saber qual mecanismo recolhe mais bandeira antes de alcançar o objetivo.

De acordo com o gráfico da Figura 5.24 as retas que convergem mais rápido, são as que possuem os valores de ε-*greedy* mais altos, independente dos valores da taxa de aprendizagem α e fator de desconto γ. É percebido nas retas de todos os experimentos que foram executadas com o valor de ε-*greedy* igual a "um", a convergência por igual.

De acordo com a análise os parâmetros utilizados no *Maze Domain* foram:

- α (taxa de aprendizagem) **0.8**
- γ (taxa de desconto) **0.8**
- ϵ (taxa de exploração) **0.9**
- λ (fator de rastreamento) **0.1**

Q-Learning vs SARSA

Na Figura 5.27 pode ser visualizado o gráfico de comparação entre o mecanismo *Q-Learning* e *SARSA* no ambiente *Maze Domain*:

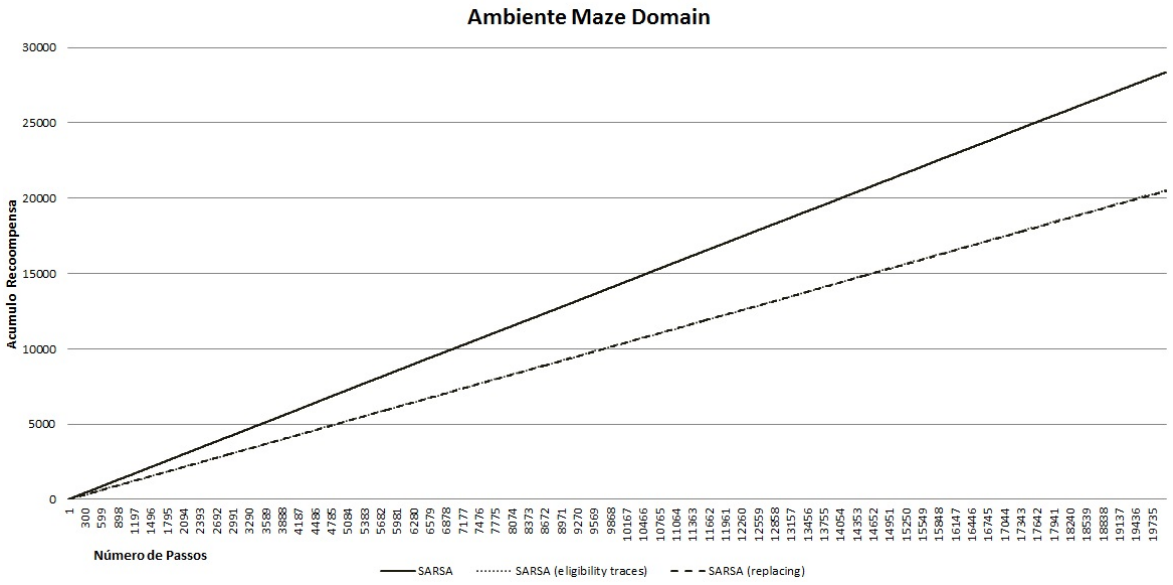

Figura 5.25: Gráfico análise *Q-Learning e SARSA* ambiente *Maze Domain*

É possível observar que no ambiente *Maze Domain* o algoritmo *SARSA* mostrou uma convergência bem mais rápida que o *Q-Learning*, ou seja, o agente recolheu mais bandeiras por episódio. Nesse caso a vantagem do mecanismo *SARSA* ocorreu pela sua característica de atualizar a função estado-ação *Q(s,a)*, de acordo com o método de exploração, diferente do *Q-Learning* que atualiza de acordo com o valor da maior ação do estado. Assim, possuindo uma maior probabilidade de explorar os valores dos estados, pois sua função *Q(s,a)* é atualizada de acordo com o ε-*greedy*, que nesse caso é 0.9.

Depois da convergência é possível notar um ambiente quase ou todo explorado pelo agente.

Eligibility Traces λ

Nas Figuras 5.26 e 5.27 podem ser visualizados os gráficos de comparação entre os mecanismos *Q-Learning* λ e *SARSA* λ no ambiente *Maze Domain*:

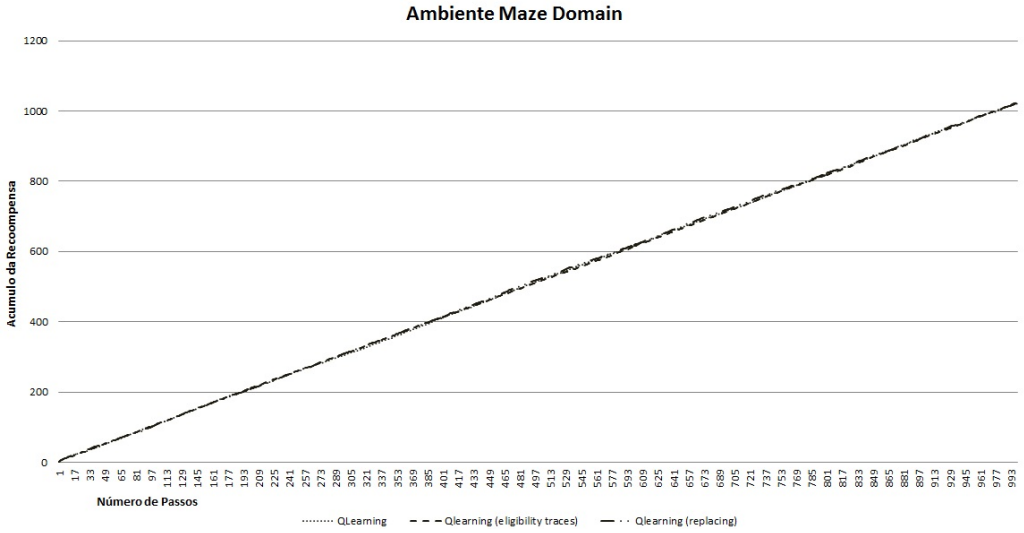

Figura 5.26: Gráfico análise *Q-Learning*, *eligibility traces* e (*replacing*) no ambiente *Maze Domain*

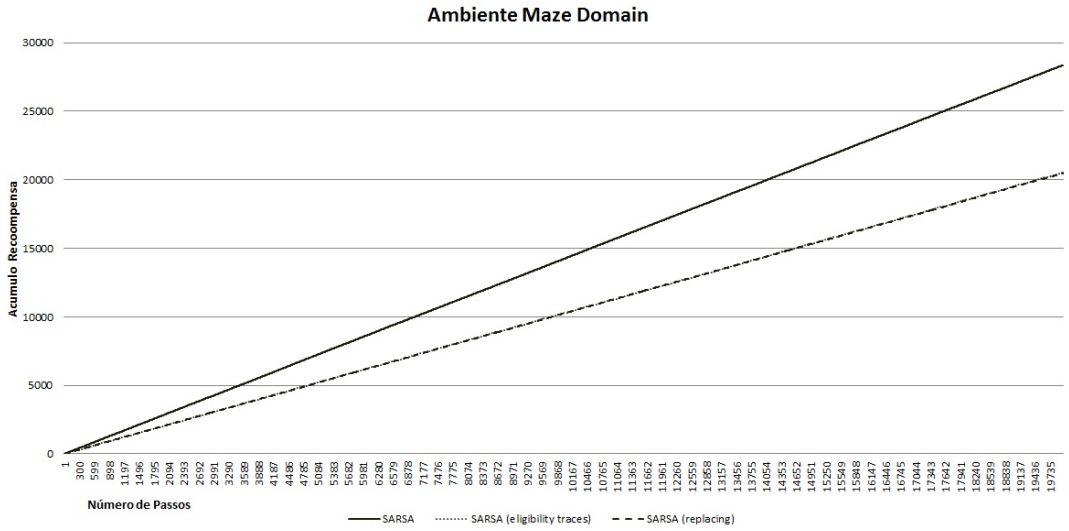

Figura 5.27: Gráfico análise *SARSA*, *eligibility traces* e (*replacing*) no ambiente *Maze Domain*

Observando os gráficos é possível notar que as retas de convergência dos mecanismos *Q-Learning*, *eligibility traces* e *replacing* são iguais. Com isso, pode-se concluir que os métodos recolhem as bandeiras do ambiente de maneira similar a cada episódio.

Já na Figura 5.27, com o gráfico de análise do mecanismo SARSA e suas variações, as curvas similares são as do *eligibility traces* e *replacing*, possuindo o algoritmo *SARSA* uma maior eficiência que os outros dois nesse ambiente. Isso acontece, pelo fato da marcação de eventos proporcionada pelos métodos de *eligibility traces*, fornecerem maior eficiência aos mecanismos, fazendo eles explorarem menos o ambiente e seguirem em direção ao estado final.

Dyna-Q

Na Figura 5.28 pode ser visualizado o gráfico de comparação entre as variações da variável de controle *n* do algoritmo *Dyna-Q* no ambiente *Maze Domain*.

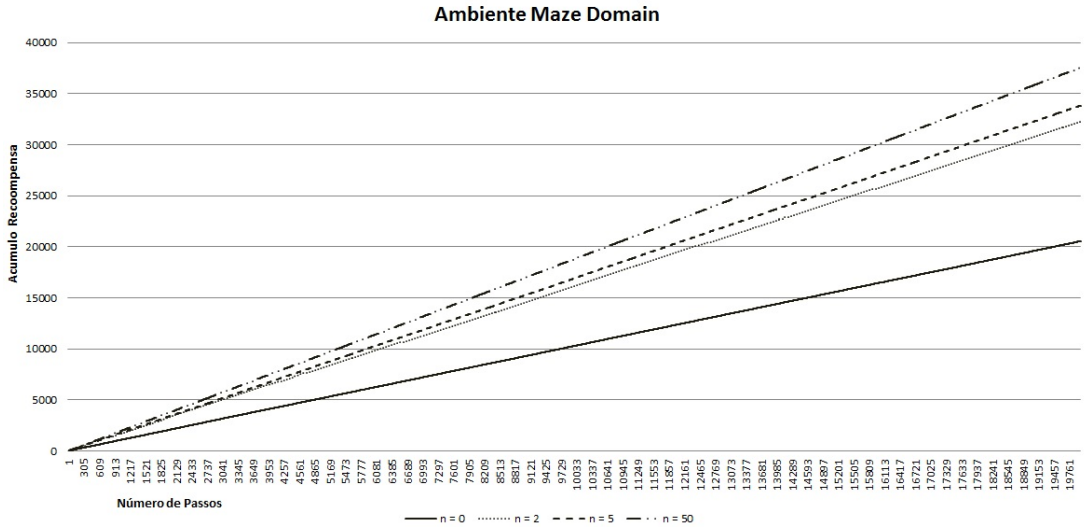

Figura 5.28: Gráfico análise *Dyna-Q* no ambiente *Maze Domain*

De acordo com a Figura 5.28, quando maior a variável de controle *n*, mais eficiente o mecanismo se comporta. A cada incremento na variável de controle, mais o agente recolhe bandeiras por episódio no ambiente.

Conclusão da Análise no Ambiente *Maze Domain*

Depois da análise dos mecanismos no ambiente *Maze Domain*, é possível observar que os algoritmos *SARSA* e *Dyna-Q*, com a variável de controle *n* igual a "50", alcançaram bons resultados em relação aos outros. Então, com o objetivo de verificar qual desses mecanismos é o mais eficiente, foi realizada uma comparação entre eles. Essa avaliação pode ser observada no gráfico da figura 5.29:

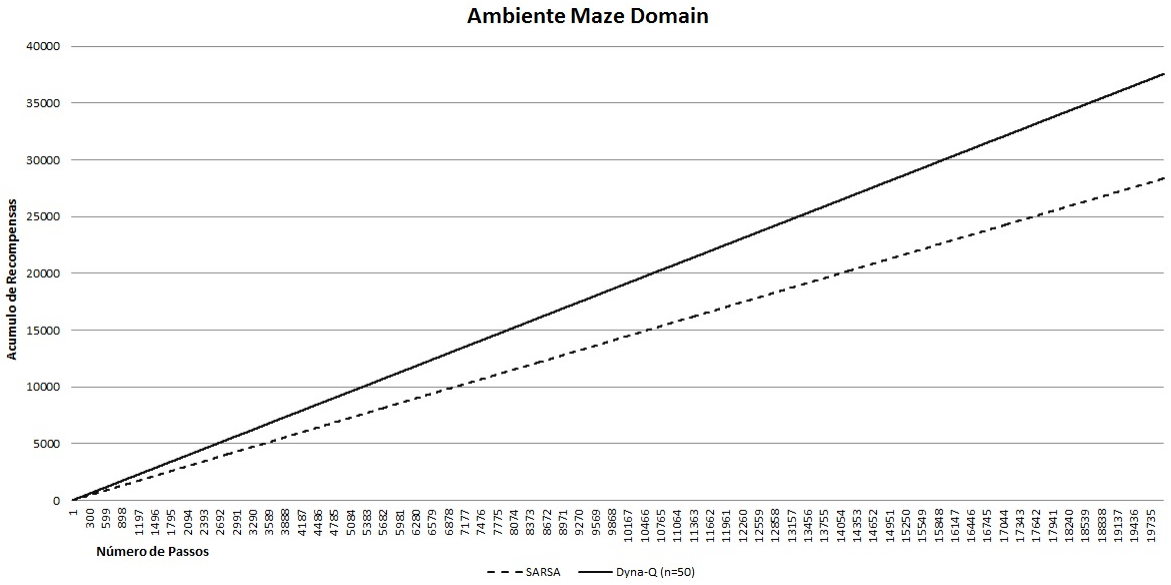

Figura 5.29: Comparação entre os Algoritmos *SARSA* e *Dyna-Q*, Ambiente *Maze Domain*

De acordo com o gráfico da Figura 5.29, o mecanismo de aprendizagem por reforço mais eficiente foi o *Dyna-Q*, alcançando uma convergência mais rápida e conseguindo obter uma quantidade maior de bandeiras a cada passo antes de alcançar o objetivo no ambiente *Maze Domain*. Isso ocorreu por causa de sua arquitetura. Conseguindo planejar as ações futuras do agente a partir de um modelo do mundo.

Depois da análise dos melhores algoritmos no ambiente *Maze Domain*, pode-se concluir que em um ambiente *tilemap*, com obstáculos e objetivos estáticos e com a métrica de verificar a quantidade total de recompensa (adquiridas a partir de *flags* no ambiente) por episódio, o algoritmo *Dyna-Q* pode obter melhor desempenho que todos os algoritmos analisados. Pois, sua característica de planejar as ações em uma cópia do ambiente, lhe da vantagem de verificar onde as *flags* estão no ambiente principal.

5.4 Conclusão

A biblioteca *AILibrary-RL* possui o objetivo de ajudar no desenvolvimento de novas aplicações que solucionem o problema de aprendizagem por reforço. A partir da criação de novos agentes, ambientes ou utilizando os já existentes para criação de novos experimentos.

Cada ambiente foi escolhido com a proposta de desafiar o mecanismo de aprendizagem a maximizar o reforço e assim alcançar o objetivo.

Nas análises não se pode deixar de notar a importância do algoritmo *Q-Learning*. Que é encontrado implementado na forma de várias variantes e ainda possui sua função de atualização *Q(s,a)* como base de tantos outros algoritmos.

Foi possível observar nas análises dos ambiente *Chain Domain* e *Loop Domain* que os mecanismo de *eligibility traces* e *replacing* conseguiram alcançar quase ou os mesmos desempenhos dos originais *Q-Learning* e *SARSA*. Já nos outros ambientes foi possível observar uma maior variação nos valores obtidos pelos mecanismos.

Também Notou-se que a evolução dos algoritmos trouxe novas técnicas na forma de solucionar os problemas encontrados. Desde a marcação sobre determinado evento (passagem por algum par de estado-ação) com objetivo de privilegia-lo nas escolhas futuras e a criação de novas arquiteturas que forneceram ao agente um tipo de planejamento no ambiente. No entanto, muitas vezes foi notado através das comparações que essas variações em alguns tipos ambientes não melhoraram tanto o mecanismo original.

Capítulo 6

Trabalhos Relacionados

Esse capítulo irá apresentar os resultados de um levantamento realizado na literatura de alguns trabalhos que analisam algoritmos de aprendizagem por reforço, para que os mesmos sejam comparados com a análise realizada neste trabalho. Também foi feito um levantamento de bibliotecas já existentes na literatura, para que elas sejam comparadas com a biblioteca desenvolvida *AILibrary-RL*.

6.1 Análise de Algoritmos de Aprendizagem por Reforço

Depois da análise dos algoritmos de aprendizagem realizada nesta dissertação, foi feito um levantamento bibliográfico com o objetivo de encontrar trabalhos similares na literatara que pudessem fornecer novos conceitos ao objetivo apresentado. Após essa pesquisa, foram escolhidos diversos trabalhos e verificado quais agentes, ambientes e métricas eles utilizaram. Sendo estes posteriormente avaliados e justapostos com a análise realizada nesta dissertação.

Alguns dos trabalhos encontrados podem ser observados com mais detalhes na lista abaixo:

- **Análise Funcional Comparativa de Algoritmos de Aprendizagem por Reforço:** Este é um trabalho de dissertação, onde alguns mecanismos de aprendizagem por reforço foram selecionados, com o objetivo de gerar estatísticas funcionais que permitissem a análise de aspectos como eficiência e eficácia dos algoritmos em condições específicas. Para gerar os dados de testes, os algoritmos foram implementados em uma plataforma modular (PSA), desenvolvida pelo autor, que permitiu a simulação

dos agentes inteligentes em dois tipos de ambientes episódicos e do tipo *tilemap* ou *grid*, onde o objetivo do agente é sair de um ponto A e alcançar um ponto B. A métrica usada foi observar o número de passos por episódio que o agente realizou para alcançar o objetivo. Os algoritmos utilizados foram:

- *Q-Learning*
- *SARSA*
- *Q-Learning* λ *(eligibility traces)*
- *SARSA* λ *(eligibility traces)*
- *Q-Learning* λ *(replacing traces)*
- *SARSA* λ *(replacing traces)*
- *Dyna-Q*
- *Dyna-Q* (com varrimento priorizado)

Todos os algoritmos pertencem ao método de Diferença Temporal. [Pessoa 2011]

- ***Simulation-Based Evaluations of Reinforcement Learning Algorithms for Autonomous Mobile Robot Path Planning:***

  Este trabalho é um artigo publicado no ITCS & IRoA 2011, sendo duas conferências internacionais, a primeira ITCS de Tecnologia da Informação Convergência e Serviços e a segunda IRoA de Robôs Inteligentes, automação, facilidades de telecomunicações e aplicações.

  O objetivo deste trabalho é comparar cinco algoritmos fundamentais de aprendizagem por reforço, sendo eles: *Q-Learning*, *SARSA*, *Q-Learning* λ, *SARSA* λ e o *Dyna-Q*. Avaliando qual dos cinco algoritmos é o mais eficiente para resolver o problema de melhor caminho para um robô. O ambiente testado é episódico do tipo *Tilemap* ou *Grid*. A métrica utilizada foi observar o número de passos por episódio que o agente realizou para alcançar o objetivo. [Viet, Kyaw e Chung 2011]

- ***Performance Comparison of Two Reinforcement Learning Algorithms for Small Mobile Robots:***

Este trabalho é um artigo publicado em 2009 no *International Journal of Control and Automation*. Seu objetivo é comparar a eficiência de convergência entre o algoritmo *Q-Learning* e uma de suas variações o *Relational Q-Learning*. O ambiente utilizado foi um ambiente contínuo e a métrica utilizada foi verificar a recompensa acumulada por passos da execução do agente. [Neruda e Slusný March, 2009]

- ***Reinforcement Learning Benchmarks and Bake-offs II:***

  Neste trabalho encontram-se um conjunto de artigos publicados na conferência internacional NIPS (Neural Information Processing Systems) de 2005. Nestes artigos observa-se a análise dos mais variados algoritmos de aprendizagem e nos mais variados tipos de ambientes. A maioria desses ambientes são os mais clássicos da literatura, como o problema do *Cart pole*, do *Mountain car*, do *Puddle world*, do *Blackjack*, do *Sensor Network* e do *Taxi*. Alguns deles são episódicos e outros são contínuos. As métricas utilizadas nos artigos para verificação da eficiência são, verificar a recompensa acumulada por passos da execução do agente, observar o número de passos por episódio que o agente realizou para alcançar o objetivo, verificar a média da recompensa acumulada por passos da execução do agente. [Dutech et al. 2005]

Observando os trabalhos analisados, é possível notar que a maioria deles são recentes e ainda utilizam os algoritmos clássicos para realizar seus experimentos e análises.

Também nota-se que a maioria dos trabalhos realizam experimentos com os algoritmos, no entanto, utilizam como base apenas um ambiente e uma métrica de avaliação, ou seja, avaliam os algoritmos em um ambiente episódico com a métrica de verificar a quantidade de ações que o agente leva para alcançar o estado final ou em um ambiente contínuo e verificam o total de recompensa acumulada a cada passo do agente.

Como pôde ser visualizado no decorrer deste trabalho, as análises dos algoritmos foram realizadas em "quatro" ambientes diferentes, cada um com seu problema referente ao aprendizado. "Dois" ambientes são contínuos e "dois" ambientes são episódicos, ou seja, duas métricas foram utilizadas para avaliar os mecanismos de aprendizagem no ambiente e verificar qual deles alcançam o melhor desempenho.

Os parâmetros dos mecanismos também foram avaliados em alguns ambientes. O objetivo dessa análise, foi observar qual variação dos parâmetros ficava mais próxima de levar

o mecanismo de aprendizagem à convergência, assim excluindo a possibilidade de testar parâmetros sem relevância.

6.2 Biblioteca

A construção de bibliotecas voltadas para inteligência artificial, com ênfase em aprendizagem, vem aumentando nos últimos anos. Este fato motivou um levantamento bibliográfico com o propósito de encontrar objetos similares que pudessem incorporar novos conceitos à proposta apresentada neste trabalho. Após esta revisão, elencaram-se diversos trabalhos correlacionados, sendo estes posteriormente avaliados e justapostos com a *AILibrary-RL*. Depois foram incorporadas diversas características a *AILibrary-RL*, a fim de beneficiar esta proposta.

Na lista abaixo poderá ser visualizado algumas dessas bibliotecas:

- **Pybrain:** A *Pybrain* é uma biblioteca modular de aprendizado de máquina, que possui o objetivo de facilitar o uso dos algoritmos de Redes neurais e aprendizagem por reforço e avaliá-los através de comparação de algoritmos pré-definidos no ambiente. *PyBrain* é uma biblioteca para uso na linguagem *python*, restringindo um pouco o seu uso para outras plataformas. [Schaul et al. 2010].

- **SkyAi:** A *SkyAi* é uma biblioteca de aprendizagem por reforço direcionada para o aprendizado de comportamentos para robôs. Seu principal objetivo é fazer os robôs aprenderem como seus usuários, de forma bem abstraída. Essa biblioteca possui uma bom tempo de execução pois foi construída em c++. [Yamaguchi 2011].

- **RL-Glue:** A *RL-Glue* é uma interface destinada ao desenvolvimento de soluções para o problema da aprendizagem por reforço. Ela fornece uma interface padrão, onde é permitido que seus usuários criem agentes, ambientes e experimentos e depois reutilizem seus códigos, assim, facilitando o desenvolvimento e reutilização. Essa interface já possui versões para várias linguagens de programação, sendo elas, C/C++, Java, *Lisp*, *Matlab*, *Python* e *Go Language*, facilitando ainda mais a reutilização de seus códigos entre os desenvolvedores [Tanner e White 2009].

- **RL-Library:** A *RL-Library* é uma biblioteca de aprendizagem por reforço. No entanto, ela é construída a partir da interface fornecida pela biblioteca *RL-Glue*, ela já possui vários ambientes, agentes e experimentos pré-definidos. [Tanner 2007].

- **RL-ToolBox:** A *RL-ToolBox* é uma biblioteca *open-source* para o desenvolvimento de todo tipo de algoritmo de aprendizagem por reforço. Ela possui uma interface que pode ser utilizada para a implementação de qualquer tipo de tarefa de aprendizagem por reforço, mas também possui uma grande variedade de algoritmos implementados. [Neumann 2005]

- **RL-ToolKit:** A *RL-ToolKit* é uma biblioteca desenvolvida por *Sutton*, um autor muito importante na área de aprendizagem por reforço. Ela tem como objetivo, fornecer uma variedade de algoritmos de aprendizagem por reforço e ambientes para auxiliar no ensino e aprendizagem deste tema. [Sutton 2011]

Com o objetivo de verificar a quantidade de algoritmos implementados em cada uma das bibliotecas a tabela 6.1 foi criada.

A Tabela 6.1 abaixo, exibe os algoritmos implementados em cada uma das bibliotecas:

Bibliotecas/ Algoritmos	**AILibrary-RL**	**PyBrain**	**SkyAI**	RL-**Library**	RL-**Toolbox**	RL-**ToolKit**
Q-Learning	X	X	X		X	X
Q-Learning λ/ (eligibility traces)	X	X	X		X	
Q-Learning λ/ (replacing traces)	X	X	X		X	
SARSA	X	X	X	X	X	X
SARSA λ/ (eligibility traces)	X	X	X		X	
SARSA λ/ (replacing traces)	X	X	X		X	
Dyna-Q	X				X	

Tabela 6.1: Tabela elencando os algoritmos implementados nas bibliotecas.

Observado a Tabela 6.1, é possível verificar que a biblioteca *RL-Glue* não está elencada, pois ela não possui algoritmos nativos implementados, ela se comporta mais como uma interface para facilitar a implementação de novos experimentos. Também é possível observar que a grande maioria das bibliotecas possuem os algoritmos clássicos, como o *Q-learning* e suas variações e o *SARSA* e suas variações. No entanto apenas a *RL-Toolbox*, das bibliotecas analisadas, possuem o algoritmo *Dyna-Q* e sua variação com varrimento priorizado implementado.

A partir da Tabela 6.1 e das descrições dos algoritmos é possível visualizar que as bibliotecas podem possuir três objetivos. O primeiro é reunir uma quantidade de agentes e ambientes para que possam ser reutilizados em um outro experimento, a segunda é ser uma interface que facilita, modulariza e reutiliza códigos para criação e desenvolvimento de novas soluções de aprendizagem por reforço e a terceira é juntar os dois conceitos.

Pode-se visualizar melhor essa descrição tendo como exemplo a biblioteca *RL-Library*. Ela possui vários agentes, ambientes e experimentos implementados, mas utiliza a interface e biblioteca *RL-Glue* para implementá-los.

O objetivo da *AILibrary-RL* é juntar esses dois conceitos. Como exemplo, temos a *PyBrain*, *SKyAI* e a *RL-Toolbox* que utilizam a mesma ideia.

Com as duas ideias juntas, é possível no momento do desenvolvimento de uma nova solução, reutilizar o código do agente ou ambiente já implementado. No entanto, se o agente ou ambiente ainda não existir, é possível implementá-lo de uma maneira mais rápida através da interface que modulariza as respectivas responsabilidades entre cada ponto da solução da aprendizagem por reforço, um exemplo é a divisão de responsabilidades entre agente, ambiente e experimento, da biblioteca *AILibrary-RL*, onde cada módulo gerencia seus métodos.

Capítulo 7

Conclusão

Este trabalho apresentou um estudo de alguns dos mais representativos algoritmos referentes a exploração e aprendizagem, relacionados ao problema de aprendizagem por reforço. Visando a análise e avaliação de seus comportamentos diante de vários tipos de ambientes e seus diferentes objetivos. Com os códigos da implementação dos algoritmos e ambientes usados para efetuar a análise, foi construída a biblioteca AILibrary-RL.

O estudo de alguns trabalhos relacionados à biblioteca dessa dissertação também foi realizado, com a finalidade de observar o que está presente na literatura e elencar algumas vantagens e desvantagens. Nesse estudo, foi observado a força que o tema da aprendizagem por reforço possui ainda nos dias de atuais.

A biblioteca *AILibrary-RL* foi desenvolvida com a finalidade de ajudar no desenvolvimento de novas soluções para problema de aprendizado por reforço. A partir dos mecanismos de aprendizagem, os métodos de exploração e os ambientes estudados, ela foi construída. Visando exatamente agilizar a criação de novos experimentos, a modularização e a reutilização de código.

Nas análises e comparações, pôde ser observado o desempenho dos algoritmos nos ambientes propostos. Cada ambiente possui um objetivo diferente para o mecanismo alcançar, assim, avaliando de formas diferentes os algoritmos. Uma das avaliações, foi pelo número de ações por episódio que o agente realizava para alcançar o objetivo. Outra pelo acumulo de recompensa a cada passo de execução do sistema e por fim a quantidade de bandeiras recolhidas antes de alcançar o estado final. Com essas análises, foi possível visualizar a eficiência de cada algoritmo e qual é o melhor em cada ambiente.

Os parâmetros usados nos algoritmos também foram analisados, com o objetivo de observar quais seriam os melhores valores para o algoritmo atingir a convergência. Depois da avaliação, os testes e análises estatísticas foram realizados de acordo com os melhores parâmetros.

Na avaliação feita no ambiente *Dyna Maze*, foi possível observar como as variações do *eligibility traces* λ melhoraram o desempenho do agente, através da característica da marcação de eventos (visita a determinado estado-ação). Também foi possível visualizar o desempenho da arquitetura *Dyna-Q*, que adquiriu um dos melhores resultados para encontrar a política ótima.

Já nas avaliações observando o acumulo de recompensa, pôde ser notado como os mecanismos se equipararam uns aos outros. Possuindo maior divergência nos resultados os algoritmos bases *Q-Learning* e *SARSA*.

Pode-se concluir que dependendo do objetivo a ser realizado no ambiente escolhido, é importante analisar se as variações dos algoritmos bases realmente irão melhorar o desempenho, pois como foi observado, quanto mais evolução ocorre nos algoritmos, maior é o gasto de processamento computacional.

Desde 1989, quando *Watkins* apresentou o mecanismo *Q-Learning* para solucionar o problema de aprendizagem por reforço, o estudo desse tema vem crescendo, tanto na complexidade dos problemas a serem solucionados, como na criação de novos métodos e mecanismos para solucionar esse problema. Nesse estudo, foi observado a força que esse tema ainda possui nos dias de hoje, com a criação e aperfeiçoamento de mecanismos de aprendizagem e métodos de exploração, como também no desenvolvimento de ambientes e bibliotecas para ajudar no estudo e implementação de novas soluções.

7.1 Trabalhos Futuros

Os métodos e mecanismos de aprendizagem e exploração estudados e implementados são uma parte importante do AR, entretanto, não são todos. Como trabalho futuro será interessante a pesquisa e implementação de novos algoritmos e ambientes que desafiem ainda mais a solução desses problemas.

Na lista abaixo encontra-se alguns algoritmos que foram pesquisados, mas não imple-

mentados:

- **Métodos Hierárquicos**: A ideia dos métodos hierárquicos é decompor um problema em uma série de sub-tarefas e solucionar cada uma separadamente, agilizando e hierarquizando a solução do problema proposto. Os valores da função estado-ação Q(s,a) nesse tipo de problema não são iguais a toda a solução, mas individualizadas por cada tarefa.

  De acordo com [Dietterich 1998], os métodos hierarquísticos podem apresentar as seguintes vantagens:

  - **Exploração Mais Eficaz:** através da evolução do algoritmo diante aos altos níveis de abstração;
  - **Aprendizagem mais Rápida:** com a quebra do problema principal em sub-tarefas menores;
  - **Atualização mais Rápida:** através das políticas anteriormente aprendidas, nas sub-tarefas já exploradas.

  A partir dos estudos realizados, foram encontrados três mecanismos referentes aos métodos hierárquicos, são eles: O *MaxQ* e suas variações *MaxQQ* e o *HSMQ*.

- ***Dyna-H***: A estratégia *Dyna-H*, foi proposta por um grupo de autores espanhóis [Santos et al. 2011]. Sua ideia é associar na construção do planejamento *Dyna-Q*, uma determinada heurística, com o objetivo de selecionar os pares estado-ação, através de uma função genérica que não dependa de nenhum modelo explícito do ambiente.

Assim como o estudo desses algoritmos, a análise e comparação deles com os existentes seriam muito interessantes. Contribuindo assim na reutilização de código para novos soluções.

Uma análise que também pode ser muito interessante é o teste dos algoritmos em ambientes dinâmicos, onde as características e recursos do ambiente mudam constantemente. Nesses casos, os mecanismos teriam que ser adaptados para uma aprendizagem *on-line*, onde suas funções *Q(s,a)* necessitariam ser revisadas constantemente com os valores das mudanças dos estados do ambiente.

7.2 Considerações Finais

Neste trabalho foi possível conferir como está o estado da arte de alguns métodos e algoritmos referentes ao estudo da aprendizagem por reforço. Também foi realizada uma comparação e análise estatística, visando a avaliação dessas soluções em diferentes tipos de ambientes. Com o código implementado para as avaliações foi desenvolvida uma biblioteca para o estudo e implementação de novos experimentos. Assim, motivando e de alguma forma contribuindo para evolução desse tema.

Bibliografia

[Coppin 2010]COPPIN, B. *Inteligência Artificial*. [S.l.]: Genio, 2010.

[Crook e Hayes 2003]CROOK, P. A.; HAYES, G. *Learning in a State of Confusion: Perceptual Aliasing in Grid World Navigation*. [S.l.]: Institute of Perception, Action and Behaviour - School of Informatics, University of Edinburgh, 2003.

[Dearden, Friedman e Russell 1998]DEARDEN, R.; FRIEDMAN, N.; RUSSELL, S. Bayesian q-learning. In: . [S.l.]: AAAI Press, 1998. p. 8.

[Dietterich 1998]DIETTERICH, T. G. *The MAXQ Method for Hierarchical Reinforcement Learning*. [S.l.]: Department of Computer Science - Oregon State University, 1998.

[Dutech et al. 2005]DUTECH, A. et al. *Reinforcement Learning Benchmarks and Bake-offs II*. [S.l.]: A workshop at NIPS conference, 2005.

[Fernandes 2003]FERNANDES, A. M. R. *Inteligência artificial: Noções Gerais*. [S.l.]: Visual Books, 2003.

[Klopf 1972]KLOPF, A. H. *Brain Function and Adaptive Systems - A Heterostatic Theory*. [S.l.]: AIR FORCE CAMBRIDGE RESEARCH LAORATORIES, 1972.

[Moore e Atkeson 1993]MOORE, A. W.; ATKESON, C. G. *Prioritized Sweeping: Reinforcement Learning with Less Data and Less Real Time*. [S.l.]: Cambridge : MIT Artificial Intelligence Laboratory, 1993.

[Neruda e Slusný March, 2009]NERUDA, R.; SLUSNý, S. *Performance Comparison of Two Reinforcement Learning Algorithms for Small Mobile Robots*. [S.l.]: International Journal of Control and Automation Vol. 2, No. 1, March, 2009.

[Neumann 2005]NEUMANN, G. *The Reinforcement Learning Toolbox, Reinforcement Learning for Optimal Control Tasks*. [S.l.]: Institut fur Grundlagen der Informationsverarbeitung (IGI), 2005.

[Peng e Williams 1992]PENG, J.; WILLIAMS, J. R. *Efficient Search Control in Dyna*. [S.l.]: Massachusetts : College of Computer Science, Northeastern University, 1992.

[Pessoa 2011]PESSOA, J. M. D. *Análise Funcional Comparativa de Algoritmos de Aprendizagem por Reforço*. [S.l.]: INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA, 2011.

[Pessoa 2011]PESSOA, J. M. D. *Análise Funcional Comparativa de Algoritmos de Aprendizagem por Reforço*. [S.l.]: INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA, 2011.

[Poole e Mackworth 2010]POOLE, D.; MACKWORTH, A. *ARTIFICIAL INTELLIGENCE FOUNDATIONS OF COMPUTATIONAL AGENTS*. 2010. `http://artint.info/html/ArtInt.html`.

[Puterman 2005]PUTERMAN, M. L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. [S.l.]: John Wiley & Sons, Inc., 2005.

[Rehm et al. 2009]REHM, F. et al. *Aplicações da Inteligência Artificial em Jogos*. 2009. `Https://disciplinas.dcc.ufba.br/pub/mata64/semestrecorrente/artigo-jogos.pdf`.

[Rummery e Niranjan 1994]RUMMERY, G. A.; NIRANJAN, M. *ON-LINE Q-LEARNING USING CONNECTIONIST SYSTEMS*. [S.l.]: Cambridge University Engineering Department, 1994.

[Russell e Norvig 2009]RUSSELL, S.; NORVIG, P. *Inteligência Artificial*. [S.l.]: Elsevier Editor, 2009.

[Santos et al. 2011]SANTOS, M. et al. *Dyna-H: a heuristic planning reinforcement learning algorithm applied to role-playing game strategy decision systems*. [S.l.]: Computer Architectures and Automation - Complutense University of Madrid, Spain., 2011.

[Schaul et al. 2010]SCHAUL, T. et al. Pybrain. *Journal of Machine Learning Research*, 2010.

[SUTTON 1990]SUTTON, R. S. *Integrated Architectures for Learning, Planning, and Reacting based on Approximating Dynamic Programming*. [S.l.]: GTE Laboratories Incorporated, 1990.

[Sutton 2011]SUTTON, R. S. *Reinforcement Learning Toolkit*. [S.l.]: University of Alberta, 2011.

[Sutton e Barto 1998]SUTTON, R. S.; BARTO, A. G. *Reinforcement Learning: An Introduction*. [S.l.]: The MIT Press, 1998.

[SUTTON e SINGH 1996]SUTTON, R. S.; SINGH, S. P. *Reinforcement Learning with Replacing Eligibility Traces*. [S.l.]: Cambridge University Engineering Department, 1996.

[Szepesvári 2010]SZEPESVáRI, C. *Algorithms for Reinforcement Learning*. [S.l.]: Morgan & Calypool, 2010.

[Tanner 2007]TANNER, B. *RL-Library*. 2007. `https://code.google.com/p/rl-library/`.

[Tanner e White 2009]TANNER, B.; WHITE, A. RL-Glue : Language-independent software for reinforcement-learning experiments. *Journal of Machine Learning Research*, v. 10, p. 2133–2136, September 2009.

[Tokic 2010]TOKIC, M. *Adaptive epsilon-greedy exploration in reinforcement learning based on value differences*. [S.l.]: Weingarten : Germany, 2010.

[Viet, Kyaw e Chung 2011]VIET, H. H.; KYAW, P. H.; CHUNG, T. *Simulation-Based Evaluations of Reinforcement Learning Algorithms for Autonomous Mobile Robot Path Planning*. [S.l.]: IT Convergence and Services, Lecture Notes in Electrical Engineering Volume 107, pp 467-476, 2011.

[Watkins 1989]WATKINS, C. J. C. H. *Learning from Delayed Rewards*. [S.l.]: Cambridge University, 1989.

[Yamaguchi 2011]YAMAGUCHI, A. *SkyAI*. 2011. `http://skyai.org/wiki/?SkyAI`.