UNIVERSIDADE FEDERAL DO PARANÁ

VINÍCYUS ARAÚJO BRASIL

Q-SVR: APRENDIZADO POR REFORÇO VIA APROXIMAÇÃO DA FUNÇÃO AÇÃO-VALOR POR MÁQUINAS DE VETORES SUPORTE

CURITIBA

2024

VINÍCYUS ARAÚJO BRASIL

Q-SVR: APRENDIZADO POR REFORÇO VIA APROXIMAÇÃO DA FUNÇÃO AÇÃO-VALOR POR MÁQUINAS DE VETORES SUPORTE

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Métodos Numéricos em Engenharia do Programa de Pós-Graduação em Métodos Numéricos em Engenharia do Setor de Ciências Exatas da Universidade Federal do Paraná.

Orientador: Profº Drº Anderson Luiz Ara Souza

CURITIBA

2024

DADOS INTERNACIONAIS DE CATALOGAÇÃO NA PUBLICAÇÃO (CIP)
UNIVERSIDADE FEDERAL DO PARANÁ
SISTEMA DE BIBLIOTECAS – BIBLIOTECA DE CIÊNCIA E TECNOLOGIA

Brasil, Vinícyus Araújo
Q-SVR: aprendizado por reforço via aproximação da função ação-valor por máquinas de vetores suporte / Vinícyus Araújo Brasil. – Curitiba, 2024.
1 recurso on-line : PDF.

Dissertação (Mestrado) - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Métodos Numéricos em Engenharia.

Orientador: Anderson Luiz Ara Souza

1. Algorítmos computacionais. 2. Aprendizado do computador. I. Universidade Federal do Paraná. II. Programa de Pós-Graduação em Métodos Numéricos em Engenharia. III. Souza, Anderson Luiz Ara. IV . Título.

Bibliotecário: Leticia Priscila Azevedo de Sousa CRB-9/2029

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação MÉTODOS NUMÉRICOS EM ENGENHARIA da Universidade Federal do Paraná foram convocados para realizar a arguição da Dissertação de Mestrado de **VINÍCYUS ARAÚJO BRASIL** intitulada: **Q-SVR: APRENDIZADO POR REFORÇO VIA APROXIMAÇÃO DA FUNÇÃO AÇÃO-VALOR POR MÁQUINAS DE VETORES SUPORTE**, sob orientação do Prof. Dr. ANDERSON LUIZ ARA SOUZA, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.

A outorga do título de mestre está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

Curitiba, 26 de Setembro de 2024.

Assinatura Eletrônica
17/10/2024 15:32:06.0
ANDERSON LUIZ ARA SOUZA
Presidente da Banca Examinadora

Assinatura Eletrônica
21/10/2024 10:15:59.0
MARCELO RODRIGO PORTELA FERREIRA
Avaliador Externo (UNIVERSIDADE FEDERAL DA PARAÍBA - UFPB)

Assinatura Eletrônica
31/10/2024 15:43:11.0
EVELIN HERINGER MANOEL KRULIKOVSKI
Avaliador Externo (DEPARTAMENTO DE MATEMáTICA - UFPR)

RESUMO

A busca por novos algoritmos que consigam resolver o problema do aprendizado por reforço, no qual um agente explora um ambiente com estados buscando tomar ações que maximizem sua recompensa, tem crescido nos últimos tempos, principalmente ao combinar com outras técnicas de aproximação de funções já estabelecidas em outras áreas do aprendizado de máquina. Apesar de alguns algoritmos já terem se consolidado da literatura, ainda existem problemas notáveis para explorar, como a dificuldade em lidar com ambientes com um alto número de ações e estados do Q-Learning e o problema da não-convexidade do algoritmo Deep Q-Network. Uma técnica pouco explorada para o problema do aprendizado por reforço é a Máquina de Vetores Suporte, técnica utilizada tanto para classificação e regressão que tem uma grande capacidade de generalização. Essa tem a característica desejável em algoritmos de otimização que é a convexidade no seu problema de otimização. O objetivo deste trabalho é propor um algoritmo de aprendizado por reforço utilizando Máquina de Vetores Suporte, o qual é chamado de Q-SVR. O estudo se limitou à ambientes de problemas de aprendizado por reforço que têm um número discreto e pequeno de estados. Testes foram realizados e mostraram a viabilidade do algoritmo proposto, assim como sua superioridade em relação aos algoritmos *Q-Learning* e *Deep Q-Network* em algumas métricas específicas. Uma aplicação no jogo de cartas Blackjack também foi apresentada. Os testes também apontaram possibilidades de melhorias do algoritmo proposto, como o alto tempo computacional e a dificuldade de lidar com problemas com muitos estados.

Palavras-chaves: Máquinas de vetores suporte. Aprendizado por reforço. Algorit mos de aproximação de função.

ABSTRACT

The search for new algorithms that can solve the reinforcement learning problem, in which an agent explores an environment with states seeking to take actions that maximize its reward, has grown in recent times, especially when combined with other function approximation techniques already established in other areas of machine learning. Although algorithms have already been consolidated in the literature, there are still notable problems to explore, such as the difficulty in dealing with environments with a high number of actions and states of Q-Learning and the problem of non-convexity of the Deep Q-Network algorithm. A little-explored technique for the reinforcement learning problem is the Support Vector Machine, a technique used for both classification and regression that has a great generalization capacity. This has a desirable characteristic in optimization algorithms, which is convexity in its optimization problem. The objective of this work is to propose a reinforcement learning algorithm using Support Vector Machine, which is called Q-SVR. The study was limited to reinforcement learning problem environments that have a small and discrete number of states. Tests were performed and showed the viability of the proposed algorithm, as well as its superiority over the *Q-Learning* and *Deep Q-Network* algorithms in some specific metrics. An application in the card game Blackjack was also presented. The tests also pointed out possibilities for improvements in the proposed algorithm, such as the high computational time and the difficulty of dealing with problems with many states.

Key-words: Support Vector Machines. Reinforcement Learning. Function approximation algorithms.

LISTA DE ILUSTRAÇÕES

FIGURA 1 – A interação agente-ambiente em AR. 17
FIGURA 2 – A política ϵ-gulosa com $\epsilon_0 = 0,99$ e $\epsilon_d = 0,99$. 20

FIGURA 3 – A função de custo ε-insensitiva e o papel das variáveis de folga. . 24
FIGURA 4 – Divisão das amostras em conjuntos 29

FIGURA 5 – Caminhada aleatória. 38
FIGURA 6 – Número médio de passos no ambiente caminhada aleatória com 7 estados. 40
FIGURA 7 – Número médio de passos no ambiente caminhada aleatória com 15 estados. 40
FIGURA 8 – Cadeia com quatro estados 40
FIGURA 9 – Valores estimados do Q-SVR que geram a política ótima desde o primeiro episódio 42
FIGURA 10 – Recompensa média na cadeia com quatro estados. 43
FIGURA 11 – Representação do grafo utilizado no problema do caminho mínimo 44
FIGURA 12 – Recompensa média por episódio do algoritmo Q-SVR no ambiente do caminho mínimo. 45
FIGURA 13 – Recompensa média por episódio do algoritmo DQN no ambiente do caminho mínimo. 45
FIGURA 14 – Recompensa média por episódio do algoritmo *Q-Learning* no ambiente do caminho mínimo. 46

FIGURA 15 – Estratégia de referência para as mãos *hard* (à esquerda) e para as mãos soft (à direita). 50
FIGURA 16 – Média móvel de 500 episódios da recompensa recebida. 51
FIGURA 17 – Estratégia gerada pelo Q-SVR para as mãos *hard* (à esquerda) e para as mãos *soft* (à direita). 51
FIGURA 18 – Estado-valor das mãos hard (à esquerda) e das mãos soft (à direita) gerado pelo Q-SVR. 52
FIGURA 19 – Comparação entre os números dos cenários possíveis. 52
FIGURA 20 – Proporção do número de vitórias das duas estratégias. 53

FIGURA 21 – Número de passos dos testes do algoritmo DQN no ambiente caminhada aleatória com 7 estados 59
FIGURA 22 – Número de passos dos testes do algoritmo DQN no ambiente caminhada aleatória com 11 estados 60

FIGURA 23 – Número de passos dos testes do algoritmo DQN no ambiente caminhada aleatória com 15 estados 61
FIGURA 24 – Número de passos dos testes do algoritmo Q-Learning no ambiente caminhada aleatória com 7 estados 62
FIGURA 25 – Número de passos dos testes do algoritmo Q-Learning no ambiente caminhada aleatória com 11 estados 63
FIGURA 26 – Número de passos dos testes do algoritmo Q-Learning no ambiente caminhada aleatória com 15 estados 64
FIGURA 27 – Número de passos dos testes do algoritmo Q-SVR no ambiente caminhada aleatória com 7 estados 65
FIGURA 28 – Número de passos dos testes do algoritmo Q-SVR no ambiente caminhada aleatória com 11 estados 66
FIGURA 29 – Número de passos dos testes do algoritmo Q-SVR no ambiente caminhada aleatória com 15 estados 67
FIGURA 30 – Recompensa em cada teste por episódio do algoritmo DQN no ambiente do caminho mínimo 68
FIGURA 31 – Recompensa em cada teste por episódio do algoritmo Q-Learning no ambiente do caminho mínimo 69
FIGURA 32 – Recompensa em cada teste por episódio do algoritmo Q-SVR no ambiente do caminho mínimo 70

LISTA DE TABELAS

TABELA 1 – Algumas funções kernel e suas formas funcionais. 26

TABELA 2 – Porcentagem dos testes que convergiram para a política ótima. . 39
TABELA 3 – Mediana dos números de episódios até a estabilização na política ótima. 39
TABELA 4 – Máximo do números de episódios até a estabilização na política ótima. 39
TABELA 5 – Função de recompensa do ambiente da caminhada em cadeia de 4 estados 41
TABELA 6 – Porcentagem de testes que convergiram para a política ótima por algoritmo na cadeia com quatro estados. 41
TABELA 7 – Medianas dos números de episódios até a política ser a política ótima na cadeia com quatro estados. 42
TABELA 8 – Porcentagem de testes que convergiram para a caminhos por algoritmo. 44
TABELA 9 – Medianas dos números de episódios até a estabilização na política 46
TABELA 10 – Tempo de execução para uma semente aleatória. 47

TABELA 11 – Valores numéricos dos parâmetros. 49
TABELA 12 – Quantidade média de jogos vencidos. 52

LISTA DE ABREVIATURAS E DE SIGLAS

AOSVR *Accurate Online Support Vector Regression*

AR Aprendizado por reforço

DQN *Deep Q-Network*

PDM Processo de decisão markoviano

SVM *Support Vector Machines*

SVR *Support Vector Regression*

LISTA DE SÍMBOLOS

$\mathcal{S}$	Conjunto de estados
$\mathcal{A}$	Conjunto de ações
$\mathcal{T}$	Dinâmica de transição
$\mathcal{R}$	Função de recompensa
s_t	Estado no tempo t
a_t	Ação no tempo t
r_t	Recompensa recebida em t
γ	Taxa de desconto
π	Política
S_t	Variável aleatória do estado
R_t	Variável aleatória da recompensa
A_t	Variável aleatória da ação
$\mathbb{E}$	Esperança matemática
v	Função estado-valor
q	Função ação-valor
$\check{Q}$	Função que estima a função ação-valor
ϵ	Probabilidade do agente tomar uma ação de exploração
ε	Tolerância da função ε-insensitiva

SUMÁRIO

1 INTRODUÇÃO 11

2 APRENDIZADO POR REFORÇO 16

2.1 PROCESSO DE DECISÃO MARKOVIANO 16

2.2 FUNÇÕES VALOR 17

2.3 EXPLORAÇÃO E APROVEITAMENTO 19

2.4 ALGORITMOS TABULARES E DE APROXIMAÇÃO DE FUNÇÃO 20

3 MÁQUINAS DE VETORES SUPORTE 23

3.1 REGRESSÃO USANDO MÁQUINAS DE VETORES SUPORTE 23

3.2 APRENDIZADO INCREMENTAL 28

3.2.1 Adicionando uma nova amostra 28

3.2.2 Algoritmo do AOSVR 32

3.3 PROPOSTA DO Q-SVR 35

4 EXPERIMENTOS 37

4.1 CONFIGURAÇÕES INICIAIS 37

4.2 AMBIENTES TESTADOS 38

4.2.1 Caminhada aleatória 38

4.2.2 Caminhada em cadeia de 4 estados 40

4.2.3 Caminho mínimo em um grafo ponderado 43

5 APLICAÇÃO NO JOGO DE CARTAS BLACKJACK 48

5.1 REGRAS BÁSICAS DO BLACKJACK 48

5.2 MODELAGEM COMO UM PROBLEMA DE APRENDIZADO POR REFORÇO 49

5.2.1 Espaço de estados e ações 49

5.2.2 Função de recompensa 49

5.3 RESULTADOS 49

6 COMENTÁRIOS FINAIS 54

REFERÊNCIAS 56

ANEXOS **58**

ANEXO A **NÚMERO DE PASSOS EM CADA UM DOS TESTES NO AMBIENTE CAMINHADA ALEATÓRIA 59**

ANEXO B **RECOMPENSA EM CADA UM DOS TESTES NO AMBIENTE DO CAMINHO MÍNIMO 68**

1 INTRODUÇÃO

O aprendizado por reforço (AR) é a área do aprendizado de máquina que visa criar agentes autônomos capazes de tomarem decisões em um ambiente para maximizarem a recompensa recebida (SUTTON; BARTO, 2020). Tais agentes aprendem à interagir no ambiente interagindo com o mesmo, o qual dá recompensas para o agente que usam tais recompensas para tomar suas futuras ações.

Tal paradigma tem uma vasta lista de aplicações. Dentre todas as possíveis aplicações, destacam-se três na literatura de aprendizado por reforço: jogos eletrônicos, finanças e aplicações médicas. No que tange ao primeiro, o mesmo tem resultados notáveis com agente capazes de derrotar melhores jogadores de Go (SILVER et al., 2016) e a de jogar Atari 2600 e, em jogos específicos, conseguir resultados melhores que humanos jogando (MNIH et al., 2013). Há aplicações no ramo de finanças também, onde agentes aprendem a negociar ativos buscando os maiores lucros nas operações (execução ótima) e o de otimização de portfolio (HAMBLY; XU; YANG, 2023). Por fim, dentre as aplicações médicas há agentes resolvendo problemas de agendamento e agentes em sistemas de diagnósticos automatizados (YU et al., 2021).

O problema pode também ser entendido como o controle ótimo de um sistema dinâmico estocástico, onde o agente aprende uma política para controlar o sistema com base em interações com o mesmo. Matematicamente, o sistema pode ser modelado como um processo de decisão markoviano (PDM), sendo que o mesmo consiste em um conjunto de conjuntos e funções. Os conjuntos que o compõem são os conjunto de estados, sendo esses todas as possíveis configurações do sistema, e o conjunto de ações, sendo essas as mudanças possíveis no estado do sistema. Uma das funções que consistem o PDM é a função de probabilidade de transição, também conhecida como dinâmica de transição, que dita a probabilidade do agente parar em um certo estado dado que estava em outro estado e tomou determinada ação. A outra função que constitui um PDM é a função de recompensa, que associa ao agente um escalar ao tomar uma certa ação em um certo estado (GEIST; PIETQUIN, 2013).

O agente deve criar uma política, ou seja, um mapeamento de cada estado para uma distribuição de probabilidade para as ações, para poder escolher as ações. Para quantificar a qualidade de estar em um certo estado, as funções estado-valor e ação-valor são utilizadas. Elas associam cada estado como a recompensa esperada onde o agente começa naquele estado e segue uma determinada política. O agente busca o máximo de recompensa esperada, sendo essa alcançada com a política ótima, ou seja, aquela que, dentre todas as políticas, maximiza o retorno acumulado esperado. Caso o modelo esteja disponível, é possível calcular a política ótima com a equação de

Bellman e com programação dinâmica (GEIST; PIETQUIN, 2013).

O aprendizado por reforço busca estimar a política ótima quando a dinâmica de transição não está disponível, utilizando apenas interações com o ambiente. Quando o problema tem uma quantidade moderada de estados e ações, as funções valor, ou seja, a função ação-valor a função estado-valor, podem ser representadas como uma tabela. Os algoritmos tabulares, como são conhecidos, tem como representante notável o *Q-Learning* (WATKINS; DAYAN, 1992), algoritmo que estima a função ação-valor. Entretanto, quando o número de estados torna-se demasiadamente grande ou trata-se de um espaço contínuo de estados, a aproximação de função deve ser introduzida para aproximar as funções valor (LONG JIHAO; HAN, 2023).

A vantagem de se usar a aproximação de função para aproximação de valor é não ser preciso representar explicitamente o par ação-valor em uma tabela. Assim, é possível que o agente generalize tais pares de acordo com a experiências coletadas. Diversas formas de utilizar aproximação de função são conhecidas na literatura como, por exemplo, a aproximação linear, que generaliza os valores da função ação-valor como uma combinação linear parametrizada por um conjunto de pesos. Métodos que combinam algoritmos tabulares e aproximação de função também foram explorados na literatura (MELO; RIBEIRO, 2007).

Outro algoritmo notável nesta classe de algoritmos é o *Deep Q-Learning* (DQN), no qual uma rede neural artificial como aproximador de função é utilizado para tomar as decisões. Introduzido por Mnih et al. (2013), o mesmo une a abordagem do Q-Learning com redes neurais, especificamente as redes neurais convolucionais. No artigo, os testes foram realizados em um emulador da plataforma Atari 2600, onde as imagens do jogo são utilizadas como entrada para a rede, que retorna uma das ações possíveis dentro de um conjunto. Os resultados obtidos em alguns experimentos foram similares aos de humanos jogando, o que caracterizou um grande avanço na área (SANTOS, 2020).

Apesar dos bons resultados, algoritmos de redes neurais tem um problema conhecido. A alta complexidade topológica do espaço de parâmetros faz com que o processo de otimização dos parâmetros tenha problemas, sendo o principal problema conhecido é a obtenção dos mínimos locais. Quando aplicado a problemas de AR com pequeno número de espaços, estes problemas afloram, como concluiu An et al. (2018).

Um modelo de aproximação de função que não sofre problemas de mínimos locais é a Máquina de Vetores Suporte (SVM - *Support Vector Machine*). Introduzido por Vapnik (1995), a formulação inicial resolvia o problema da classificação de uma nova amostra em um de dois grupos. Tal modelo foi adaptado algum tempo depois para resolver a tarefa de regressão, modelo conhecido como *Support Vector Regressor* (SVR). A característica do seu problema de otimização ser um problema quadrático

e utilizar a função kernel, que gera uma matriz hessiana semidefinida positiva, gera um problema convexo, o que implica que o seu minimizador local é um minimizador global. Tais características culminam em boas propriedades de generalização como demonstram testes realizados (MARTIN, 2002), principalmente em problemas onde há poucas amostras disponíveis.

No contexto da literatura de AR, o SVM foi ainda pouco explorado. Como o algoritmo para resolver o problema de otimização foi inicialmente criado para trabalhar com um conjunto de várias amostras de uma vez, foi necessário o desenvolvimento de algoritmos que pudessem calcular a atualização dos parâmetros dada uma nova amostra. Logo, somente após a introdução de formas de se calcular de forma incremental que se tornou possível a utilização do SVM para o AR, uma vez que o agente pode se adaptar ao ambiente enquanto o explora (LEE et al., 2009).

O primeiro algoritmo para calcular o SVR de forma *online* foi o de Martin (2002). O trabalho é uma extensão do trabalho de Cauwenberghs e Poggio (2001) que trabalharam em uma forma incremental para o SVM para classificação. O método de Martin conseguia os mesmos resultados que métodos exatos de programação quadrática, além de ser mais rápido para a época e ter a característica de ser incremental, o que abriu portas para o uso em séries temporais e AR.

Baseado no trabalho de Martin, Lee et al. (2009) foram capazes de utilizar o método para resolver o problema do AR. Utilizando-se do algoritmo SVR de Martin, o algoritmo utiliza o erro da Diferença Temporal como variável alvo do SVR para conseguir avaliar as ações do estado em que o agente se encontra. O método proposto conseguiu superar o *Q-learning* e o SARSA no balanceamento do carrinho com vara e massa e no ambiente da caminhada em cadeia de quatro estados.

An et al. (2018) utilizaram uma metodologia diferente para resolver o problema, uma SVM de forma incremental para classificação. Os autores apresentam o algoritmo denominado SVM-A2C, onde o agente utiliza o SVM para selecionar as ações via classificação. Há também o uso da estrutura do *Actor-Critic* para melhorar ainda mais as ações tomadas, onde o *Actor* toma as decisões e o *Critic* atualiza os parâmetros relacionando os dados do estado atual com a ação que tem o maior valor de vantagem.

Uma alternativa ao trabalho de Martin na tarefa do aprendizado incremental, Ma, Theiler e Perkins (2003) apresentaram o AOSVR (*Accurate Online Support Vector Regression*), algoritmo capaz de chegar nos mesmos resultados de métodos que resolvem o problema utilizando programação quadrática porém em menos tempo computacional, segundo os autores. Observa-se então uma oportunidade de pesquisa, empregando a capacidade do AOSVR para resolver o problema do AR.

Portanto, apesar da escassez de trabalhos unindo SVM e AR, há um enorme potencial de desenvolvimento de algoritmos, uma vez que as diversas áreas aplicáveis exigem uma variedade de algoritmos. O objetivo deste trabalho é, então, utilizar o AOSVR para resolver o problema do aprendizado por reforço. Para tanto, foi desenvolvido um novo algoritmo, denominado Q-SVR, que adapta o AOSVR para ser utilizado em ambientes de AR. O algoritmo foi implementado computacionalmente e comparado com outros modelos já conceituados na literatura.

Dada as diversas implicações teóricas e práticas, a presente dissertação limita-se à testar apenas em ambientes com espaços discretos e com um baixo número de estados. Tal decisão é tomada uma vez que simplifica a análise dos resultados, uma vez que a mesma em ambientes contínuos costuma ser mais complexa.

O presente trabalho está organizado da seguinte forma: no Capítulo 2, é apresentado a formalização matemática do aprendizado por reforço. Em seguida, no Capítulo 3, é formalizado o modelo de Máquina de Vetores Suporte, assim como o algoritmo AOSVR para fazer o aprendizado incremental. No Capítulo 4 é apresentado o algoritmo do Q-SVR, assim como a configuração e resultados de experimentos realizados. O Capítulo 5 é dedicado à apresentar uma aplicação do algoritmo no jogo de cartas Blackjack. Por fim, o Capítulo 6 apresenta os comentários finais, assim como indicações de caminhos para futuros trabalhos.

2 APRENDIZADO POR REFORÇO

Este capítulo é dedicado à apresentar os conceitos básicos para que o problema do aprendizado por reforço possa ser formalizado. A seguir, são apresentadas características que servem para diferenciar os algoritmos para a resolução do problema. Por fim, alguns algoritmos básicos são apresentados.

2.1 PROCESSO DE DECISÃO MARKOVIANO

Formalmente, podemos definir o problema do aprendizado por reforço como um processo de decisão markoviano (PDM), que consiste na tupla $\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}$, onde

- $\mathcal{S}$ é o conjunto de estados, onde denota-se um estado $s \in \mathcal{S}$;
- $\mathcal{A}$ é o conjunto de ações, onde denota-se uma ação $a \in \mathcal{A}$;
- $\mathcal{T}$ é a dinâmica de transição, um mapeamento da ação a_t e estado s_t para um outro estado s_{t+1}, denotado por $\mathcal{T}(s_{t+1}, |s_t, a_t)$;
- $\mathcal{R}$ é a função de recompensa, onde $\mathcal{R}(s_t, a_t, s_{t+1}) \in \mathbb{R}$.

O agente toma algumas ações disponíveis dentro do conjunto $\mathcal{A}$, o que o muda do estado s_t para o s_{t+1} e lhe concede a recompensa r_t. O objetivo do agente é tomar ações que maximizem o retorno, sendo esse a soma das recompensas apartir daquela ação naquele estado. No caso dos PDMs finitos, os quais acabam após T episódios, pode-se escrever o retorno como:

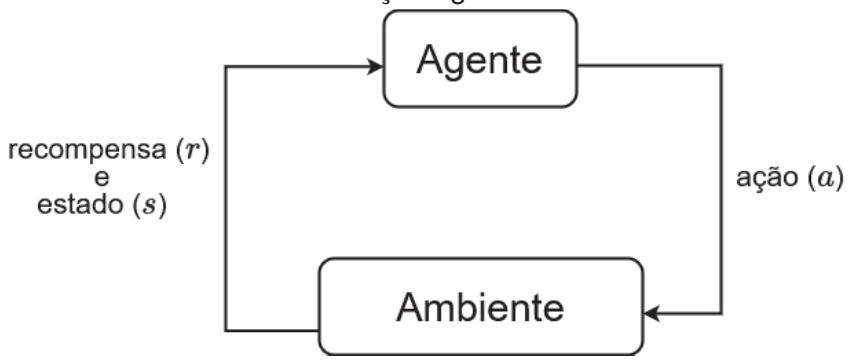
$$G_t = \sum_{t=0}^{T-1} \gamma^t r_{t+1},$$

sendo $\gamma \in (0, 1]$ é o fator de desconto. No caso de γ mais próximo de 1, as ações em tempos mais futuros tem mais peso relativo ao caso onde γ é menor.

Para tomar as decisões, o agente utiliza uma política $\pi(a_t|s_t)$ sendo essa definida como um mapeamento de estados para distribuições de probabilidades de ações. A Figura 1 ilustra a interação agente-ambiente.

Este conjunto de ações funciona como uma regra para as decisões do agente naquele estado específico. Como é possível haver diversos conjuntos de ações, há um conjunto dessas que maximiza o retorno esperado, ou seja, a soma do produto de cada probabilidade de saída da experiência pelo seu respectivo valor, representado

FIGURA 1 – A interação agente-ambiente em AR.



FONTE: (LEE et al., 2009).

seu valor médio se for repetida muitas vezes. Tais políticas são chamadas de políticas ótimas, denotadas por $\pi^*(a_t|s_t)$, e escritas como

$$\pi^*(a_t|s_t) = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}\Big[G_t\Big|\pi\Big].$$

É necessário apontar que uma hipótese que é assumida, quando se modela os problemas de aprendizado por reforço com processos de decisão markoviano, é a propriedade de Markov (SUTTON; BARTO, 2020). A mesma assume que o estado e ação anteriores à atual contém toda a informação relevante a respeito das interações agente-ambiente passadas. Logo, pode-se assumir que todos os valores possíveis de s_t e r_t estejam definidos em uma distribuição de probabilidade discreta dependente apenas de s_{t-1} e a_{t-1} e que pode se pode escrever tal distribuição como

$$p(s_{t+1}, r_{t+1}|s_t, a_t) = Pr(\{S_{t+1} = s_{t+1}, R_t = r_{t+1}|S_t = s_t, A_t = a_t\})$$

onde S_t, R_t e A_t representam as variáveis aleatórias na distribuição.

2.2 FUNÇÕES VALOR

A partir dos conceitos apresentados na seção anterior, é possível criar funções que estimam a qualidade, em termos de retorno esperado, de tomar certas ações em certo estado. Em termos apenas do estado atual, ou seja, da qualidade de se estar naquele estado, podemos definir a função estado-valor $v_\pi(s_t)$ para uma certa política π e um estado s_t,

$$v_\pi(s_t) = \mathbb{E}_\pi\Big[G_t|S_t = s_t\Big], \forall s \in \mathcal{S}.$$

A equação de Bellman apresenta a relação recursiva de $v_\pi(s_t)$ com $v_\pi(s_{t+1})$ (SANTOS, 2020),

$$v_\pi(s_t) = \sum_a \pi(a|s_t) \sum_{s_{t+1}, r_{t+1}} p(s_{t+1}, r_{t+1}|s_t, a)[r_{t+1} + \gamma v_\pi(s_{t+1})], \quad \forall s \in \mathcal{S}.$$

Com ela, pode-se definir então a política ótima utilizando a função estado-valor,

$$v_*(s_t) = \max_a \sum_{s_{t+1}, r_{t+1}} p(s_{t+1}, r_{t+1}|s_t, a)[r_{t+1} + \gamma v_*(s_{t+1})].$$

Definindo também a função ação-valor $q_\pi(s_t, a_t)$, cuja interpretação é a de qualidade, em termos de retorno acumulado, de tomar a ação a no estado s_t seguindo uma política π,

$$q_\pi(s_t, a_t) = \mathbb{E}_\pi\Big[G_t|S = s_t, A_t = a_t\Big], \forall s_t \in \mathcal{S} \quad e \quad \forall a_t \in \mathcal{A}.$$

De forma similar à função estado-valor, pode-se também definir a política ótima em função da função ação-valor:

$$q_*(s_t, a_t) = \max_a \sum_{s_{t+1}, r_{t+1}} p(s_{t+1}, r_{t+1}|s_t, a)\Big[r + \gamma \max_a q_*(s_{t+1}, a)\Big].$$

2.3 EXPLORAÇÃO E APROVEITAMENTO

Um dos problemas que surgem durante a interação do agente com o ambiente é balancear a quantidade de ações que ele toma para explorar o ambiente (que podem trazer maiores recompensas) com a quantidade de ações que toma para aproveitar as recompensas que já conhece. O dilema é conhecido como dilema da exploração e aproveitamento (do inglês *exploration and exploitation*) e várias maneiras de lidar com ele foram propostas na literatura. Uma delas é a política ϵ-gulosa para tomar as decisões. Nela, o agente toma uma ação aleatória com probabilidade ϵ (visando explorar o ambiente) e a ação com o maior valor esperado com probabilidade $1 - \epsilon$ (visando se aproveitar do conhecimento que tem no ambiente). Logo, a escolha da ação pelo agente é feita da seguinte forma:

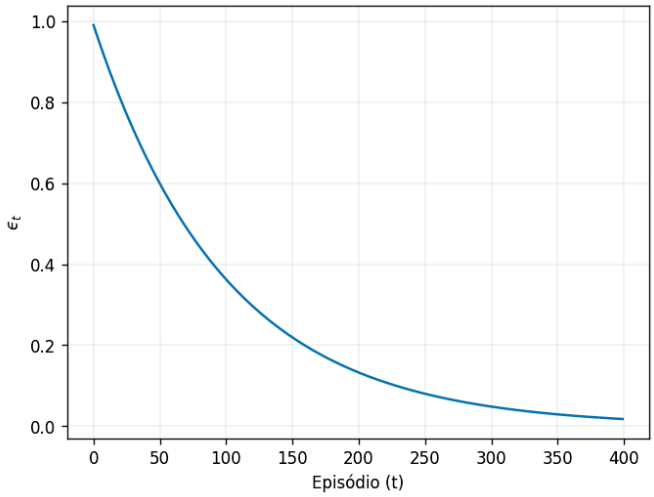
$$\text{Ação no estado} \quad s = \begin{cases} \text{Exploração com probabilidade} \quad \epsilon. \\ \text{Aproveitamento com probabilidade} \ 1 - \epsilon. \end{cases}$$

Para garantir uma exploração inicial do ambiente e depois um aproveitamento, o valor de ϵ deve variar com o tempo. Definindo ϵ_0 como um valor inicial e ϵ_d como o valor de decaimento, o valor de ϵ_t em um certo episódio t é dado por

$$\epsilon_t = \epsilon_0 \epsilon_d^t.$$

A Figura 2 exemplifica a equação com valores arbitrários. Definindo ϵ_t dessa forma permite que nos episódios iniciais haja mais exploração por parte do agente e que, conforme o número de episódios aumenta, mais o agente se aproveita do conhecimento do ambiente.

FIGURA 2 – A política ϵ-gulosa com $\epsilon_0 = 0,99$ e $\epsilon_d = 0,99$.



FONTE: O autor (2024).

2.4 ALGORITMOS TABULARES E DE APROXIMAÇÃO DE FUNÇÃO

Uma vez formalizado o problema de aprendizado por reforço, há algoritmos que visam buscar uma solução para o mesmo. Estes podem ser classificados por tentarem aproximar a função $\mathcal{T}(\cdot)$ e derivar um controlador dela, sendo estes chamados de algoritmos baseados em modelos. Caso não os tentem, ou seja, tentem aprender a controlar sem aproximar o modelo, são chamados de algoritmos livres de modelos (KAELBLING; LITTMAN; MOORE, 1996).

Outra forma que podem ser classificados pela forma que a política é utilizada. No caso de algoritmos dentro da política (*On-policy*), as ações são tomadas de acordo com uma política π, ao mesmo tempo que essa política é atualizada por meio de suas funções valor. No caso de algoritmos fora da política (*Off-policy*), a política utilizada para a tomada de ações se diferencia da política aplicada para o aprendizado (SANTOS, 2020). Dentre os diversos algoritmos existentes, dois foram escolhidos para apresentação no presente estudo dada sua importância histórica e contemporânea, respectivamente.

Um algoritmo clássico *Off-policy* é o *Q-Learning*, proposto por Watkins e Dayan (1992), que utiliza a função ação-valor para a seleção de ações. Conforme interage com o ambiente, o algoritmo atualiza sua estimativa para cada par (s, a) em uma tabela chamada de *Lookup Table* até convergir para uma solução, como o módulo da diferença entre dois valores estimados das funções valor seja sempre menor que um certo valor, ou parar em um critério de parada, como o número máximo de episódios. O Algoritmo 1 ilustra o funcionamento do mesmo. Esse é um exemplo clássico de um algoritmo tabular, onde os valores aproximados da função Q são guardados em um tabela, sendo cada tupla uma linha e uma coluna.

Algoritmo 1 *Q-Learning*

Entrada: $\alpha, \epsilon \geq 0$
  inicializar $Q(s, a) \quad \forall s \in \mathcal{S}$ e $a \in \mathcal{A}$ arbitrariamente
  para episódio **faça**
    Inicialize s_t
    enquanto s_t não for terminal **faça**
      Escolha a_t de s_t de acordo com a política derivada de Q (por ex. ϵ-gulosa)
      Execute ação a_t e observe r_{t+1} e s_{t+1}
      $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$
    fim enquanto
  fim para

Além dos métodos tabulares, há um conjunto de algoritmos que utilizam funções para representar a função Q ao invés de tabelas. Tais algoritmos são conhecidos como aproximadores de função, o que permite ao agente de aprendizagem generalizar a partir de estados que ele visitou até estados que não visitou (NORVIG; RUSSELL, 2014). Há diversas formas de algoritmos aproximadores de função, como os que utilizam funções lineares, por exemplo. Um algoritmo notável nesta classe de algoritmos é o *Deep Q-Network* (DQN), o qual usa uma rede neural artificial como aproximador de função para tomar as decisões. A mesma é parametrizada com um conjunto de pesos θ que vai sendo atualizado durante o processo de treinamento.

A atualização dos pesos é realizada a partir do cálculo do gradiente da equação abaixo. Ao invés de calcular analiticamente o gradiente da função, utiliza-se o método do gradiente descendente estocástico, atualizando a cada passo de tempo. A técnica de repetição de experiências é utilizada e tem demonstrado importantes resultados para realizar o treinamento, uma vez o método do gradiente descendente estocástico tende a ser instável (LINS, 2020). A técnica consiste em armazenar o histórico $e_t = (s_t, a_t, r_t, s_{t+1})$ a cada passo t em uma estrutura de dados $D = \{e_1, \ldots, e_t\}$. Durante o treinamento, são retiradas aleatoriamente com a mesma probabilidade pequenos lotes da experiência $(s_t, a_t, r_t, s_{t+1}) \sim U(D)$ onde $U(D)$ representa uma distribuição uniforme. Com as amostras de experiência, na i-ésima iteração a atualização dos pesos θ_i da rede é feita em relação à função de custo $L_i(\theta_i)$ e dos pesos θ_i^- da iteração e dada por:

$$L_i(\theta_i) = E_{(s_i, a_i, r_{i+1}, s_{i+1}) \sim (D)} \left[\left(r + \gamma \max_{a_{i+1}} Q(s_{i+1}, a_{i+1}; \theta_i^-) - Q(s_i, a_i; \theta_i) \right)^2 \right].$$

O algoritmo da DQN, como apresentado por Lins (2020), é descrito no Algoritmo 2. Nele, θ_i representa os parâmetros da rede na iteração i e θ_i^- representa os parâmetros fixados por uma constante C calculadados na iteração passada, usados para computar o valor alvo $\check{Q}$ na iteração i.

Algoritmo 2 *Deep Q-Network* com repetição por experiência

Inicializar memória de repetição D com a capacidade N
Inicializar a função $Q(s,a) \quad \forall s \in \mathcal{S}$ e $a \in \mathcal{A}$ com pesos θ aleatórios
Inicializar a função $\check{Q}(s,a) \quad \forall s \in \mathcal{S}$ e $a \in \mathcal{A}$ com pesos θ^- aleatórios
para episódio até episódio máximo **faça**
  Inicialize s_t e realize o pré processamento $\varphi(s_t)$
  enquanto s_t não for terminal **faça**
    Escolha a_t de s_t de acordo com a política derivada de $Q(s_t, a_t; \theta)$ (por ex. ϵ-gulosa)
    Execute ação a_t e observe r_{t+1} e s_{t+1}
    Armazene $(\varphi(s_t), a_t, r_{t+1}, \varphi(s_{t+1}))$ em D
    Retire ($\varphi(s_i)$, a_i, r_i, $\varphi(s_{i+1})$) aleatoriamente de D
    $$y_i \rightarrow \begin{cases} r_i & \text{, se o episódio terminar em } i+1 \\ r_i + \gamma \max_{a_{i+1}} \check{Q}(\varphi(s_{i+1}), a_{i+1}; \theta) & \text{, caso contrário} \end{cases}$$
    Calcule o gradiente de $(y_i - Q(\varphi(s_{i+1}), a_{i+1}; \theta))^2$ em relação aos pesos θ
    A cada passo reinicie $\check{Q} = Q$
  fim enquanto
fim para

3 MÁQUINAS DE VETORES SUPORTE

Neste capítulo será apresentado o modelo de Máquinas de Vetores Suporte (*Support Vector Machines* - SVM) visando a tarefa de regressão, chamado de SVR (*Support Vector Regression*). Também será apresentado o seu algoritmo de aprendizado incremental chamado de AOSVR (*Accurate Online Support Vector Regression*). Por fim, a proposta de uso do SVR para o problema do aprendizado por reforço é descrita.

3.1 REGRESSÃO USANDO MÁQUINAS DE VETORES SUPORTE

Como apresentam Parrella (2007) e Ma, Theiler e Perkins (2003), dado o conjunto de dados $U = \{(\mathbf{x}_i, y_i), i = 1, \dots N\}$ onde $\mathbf{x}_i \in \mathbb{R}^p$ e $y_i \in \mathbb{R}$, pode-se construir a seguinte função linear no espaço das características $\mathcal{F}$:

$$f(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b \tag{3.1}$$

sendo $\mathbf{w}$ um vetor em $\mathcal{F}$ e $\phi(\mathbf{x})$ mapeia o vetor $\mathbf{x}$ para um vetor em $\mathcal{F}$. O vetor $\mathbf{w}$ e o escalar b são o minimizadores do seguinte problema de otimização:

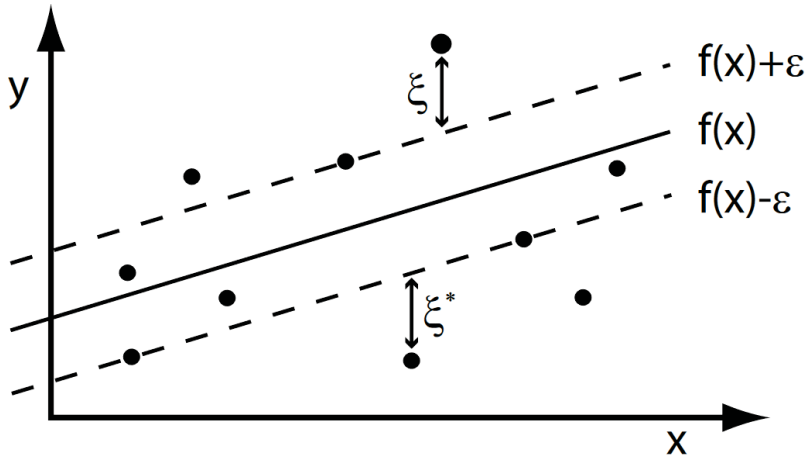
$$\begin{aligned} \min_{\mathbf{w},b} \quad & P = \frac{1}{2}\mathbf{w}^T \cdot \mathbf{w} \\ \text{s.a} \quad & y_i - (\mathbf{w}^T \phi(\mathbf{x}_i) + b) \leq \varepsilon \\ & (\mathbf{w}^T \phi(\mathbf{x}_i) + b) - y_i \leq \varepsilon \end{aligned}$$

sendo essa a chamada função de custo ε-insensitiva e ilustrada na Figura 3. Nessa função, o hiperparâmetro ε define o tamanho da margem de erro aceita. Para um aprofundamento no estudo dos hiperparâmetros, Krulikovski (2017) clareia tais conceitos. Adicionando as variáveis de folga ξ_i e ξ_i^*

$$\begin{aligned} \min_{\mathbf{w},b} \quad & P = \frac{1}{2}\mathbf{w}^T \cdot \mathbf{w} + C \sum_{i=1}^{N} (\xi_i + \xi_i^*) \\ \text{s.a} \quad & y_i - (\mathbf{w}^T \phi(\mathbf{x}_i) + b) \leq \varepsilon + \xi_i, \\ & (\mathbf{w}^T \phi(\mathbf{x}_i) + b) - y_i \leq \varepsilon + \xi_i^*, \\ & \xi_i, \xi_i^* \geq 0, i = 1, \dots, N. \end{aligned}$$

Na função, o valor C é um hiperparâmetro que controla o grau de penalização para erros fora da margem ε. O critério de otimização penaliza os valores preditos pela função $f(\mathbf{x})$ que tem um diferença maior que ε de y_i. As variáveis ξ_i, ξ_i^* são variáveis de folga que correspondem à diferença entre y_i e $f(\mathbf{x}) + \varepsilon$ e y_i e $f(\mathbf{x}) - \varepsilon$, respectivamente, como mostra a Figura 3.

FIGURA 3 – A função de custo ε-insensitiva e o papel das variáveis de folga.



FONTE: Ma, Theiler e Perkins (2003).

Introduzindo os multiplicadores de Lagrange α, α_i, η e η^* para penalizar as restrições, ou seja, tirar como restrições e colocar na função objetivo, podemos escrever a Lagrangiana como

$$\begin{aligned}\mathcal{L}_P(\mathbf{w}, b, \varepsilon) = \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}(\xi_i + \xi_i^*) - \sum_{i=1}^{N}(\eta_i\xi_i + \eta_i^*\xi_i^*) \\ & - \sum_{i=1}^{N}\alpha_i(\varepsilon + \xi_i + y_i - \mathbf{w}^T\phi(\mathbf{x}_i) - b) \\ & - \sum_{i=1}^{N}\alpha_i^*(\varepsilon + \xi_i^* + y_i - \mathbf{w}^T\phi(\mathbf{x}_i) - b) \\ \text{s.a}\end{aligned}$$

onde $\alpha_i, \alpha_i^*, \eta_i, \eta_i^*, \geq 0$. As condições de primeira ordem são

$$\begin{aligned}
\frac{\partial \mathcal{L}_P}{\partial b} &= -\sum_{i=1}^{N}(\alpha_i - \alpha_i^*) = 0, \\
\frac{\partial \mathcal{L}_P}{\partial \mathbf{w}} &= \mathbf{w} - \sum_{i=1}^{N}\phi(\mathbf{x}_i)(\alpha_i - \alpha_i^*) = 0 \implies \mathbf{w} = \sum_{i=1}^{N}\phi(\mathbf{x}_i)(\alpha_i - \alpha_i^*), \\
\frac{\partial \mathcal{L}_P}{\partial \xi_i} &= C - \eta_i - \alpha_i = 0 \implies \eta_i = C - \alpha_i, \alpha_i \in [0, C], \\
\frac{\partial \mathcal{L}_P}{\partial \xi_i^*} &= C - \eta_i^* - \alpha_i^* = 0 \implies \eta_i^* = C - \alpha_i^*, \alpha_i^* \in [0, C], \\
\frac{\partial \mathcal{L}_P}{\partial \eta_i} &= \sum_{i=1}^{N}\xi_i = 0, \\
\frac{\partial \mathcal{L}_P}{\partial \eta_i^*} &= \sum_{i=1}^{N}\xi_i^* = 0.
\end{aligned}$$

Substituindo as novas definições de $\mathbf{w}, \eta$ e η^* na Lagrangiana, tem-se que (PARRELLA, 2007):

$$\begin{aligned}\mathcal{L}_P(\mathbf{w}, b, \varepsilon) = &-\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_j)(\alpha_i-\alpha_i^*)(\alpha_j-\alpha_j^*)\\ &+\sum_{i=1}^{N}y_i(\alpha_i-\alpha_i^*)\\ &-\sum_{i=1}^{N}\varepsilon(\alpha_i-\alpha_i^*)\\ &-C\sum_{i=1}^{N}(\xi_i+\xi_i^*).\end{aligned}$$

Utilizando as condições de primeira ordem da Lagrangeana para ξ e ξ^*

$$\sum_{i=1}^{N}\xi_i = 0, \quad \sum_{i=1}^{N}\xi_i^* = 0$$

pode-se remover o parâmetro C, fazendo com que o problema de otimização dual seja:

$$\begin{aligned}\min_{\alpha,\alpha^*} \quad & D = \frac{1}{2}\sum_{i=1}^{N}\sum_{i=1}^{N}\mathcal{Q}_{ij}(\alpha_i-\alpha_i^*)(\alpha_j-\alpha_j^*)\\ &+\varepsilon\sum_{i=1}^{N}(\alpha_i-\alpha_i^*)\sum_{i=1}^{N}+y_i(\alpha_i-\alpha_i^*)\\ \text{s.a} \quad & 0\le\alpha_i,\alpha_i^*\le C, i=1,\ldots,N,\\ & \sum_{i=1}^{N}(\alpha_i-\alpha_i^*)=0,\end{aligned} \tag{3.2}$$

sendo $\mathcal{Q}_{ij} = \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j) = k(\mathbf{x}_i, \mathbf{x}_j)$ sendo $k(\mathbf{x}_i, \mathbf{x}_j)$ a função kernel. Dada uma função $\phi : \mathbb{R}^n \to \mathbb{R}^m$, as funções kernel são definidas como o produto interno entre o resultado de se aplicar o mapeamento em dois vetores $\mathbf{x}_1$ e $\mathbf{x}_2$ em F, espaço das características:

$$k(\mathbf{x}_1, \mathbf{x}_2) = \phi(\mathbf{x}_1)\cdot\phi(\mathbf{x}_2).$$

Como mostra Hamel(2009), o truque do kernel diz que se escolhe-se a função kernel apropriada não é necessário calcular $\phi(\cdot)$. A Tabela 1 mostra algumas das funções kernel comumente utilizadas com o SVM.

Introduzindo os multiplicadores de Lagrange $\delta_i, \delta_i^*, u_i, u_i^*$ e ζ, a Lagrangeana do

TABELA 1 – Algumas funções kernel e suas formas funcionais.

Nome	$k(\mathbf{x}_1, \mathbf{x}_2)$	parâmetro
Linear	$\mathbf{x}_1^T \cdot \mathbf{x}_2$	sem parâmetro
Polinomial	$(\mathbf{x}_1^T \cdot \mathbf{x}_2)^d$	$d \geq 2$ e $d \in \mathbb{Z}$
RBF	$e^{-d\|\|\mathbf{x}_1 - \mathbf{x}_2\|\|^2}$	$d > 0$ e $d \in \mathbb{R}$
RBF Gaussiano	$e^{-\frac{\|\|\mathbf{x}_1 - \mathbf{x}_2\|\|^2}{2d^2}}$	$d > 0$ e $d \in \mathbb{R}$

FONTE: Adaptado de Hamel (2009).

Equação (3.2) é dada por

$$\begin{aligned} L_D =& \frac{1}{2}\sum_{i=1}^{N}(\alpha_i - \alpha_i^*)\mathcal{Q}_{ij}(\alpha_j - \alpha_j^*) + \varepsilon\sum_{i=1}^{N}(\alpha_i + \alpha_i^*) - \sum_{i=1}^{N} y_i(\alpha_i - \alpha_i^*) \\ &+ \sum_{i=1}^{N}(\delta_i\alpha_i - \delta_i^*\alpha_i^*) + \sum_{i=1}^{N}(u_i(\alpha_i - C) - u_i^*(\alpha_i^* - C)) \\ &+ \zeta\sum_{i=1}^{N}(\alpha_i - \alpha_i^*) \end{aligned}$$

As condições de primeira ordem dá função são

$$\begin{aligned} \frac{\partial L_D}{\partial \alpha_i} &= \sum_{j=1}^{N}(\alpha_j - \alpha_j^*)\mathcal{Q}_{ij}(\alpha_i - \alpha_i^*) + \varepsilon - y_i + \zeta - \delta_i + u_i = 0 \\ \frac{\partial L_D}{\partial \alpha_i^*} &= -\sum_{j=1}^{N}(\alpha_j - \alpha_j^*)\mathcal{Q}_{ij}(\alpha_i - \alpha_i^*) + \varepsilon - y_i + \zeta - \delta_i^* + u_i^* = 0 \\ \frac{\partial L_D}{\partial \zeta} &= \sum_{j=1}^{N}(\alpha_j - \alpha_j^*) = 0 \end{aligned}$$

As condições para um ponto ser ótimo são (pelo Teorema de Kuhn-Tucker):

$$\begin{cases} \alpha_i \in [0, C], & \alpha_i(\sum_{j=1}^{N}\mathcal{Q}_{ij}(\alpha_i - \alpha_i^*) + \zeta - y_i + \varepsilon - \delta_i + u_i) = 0 \\ \alpha_i^* \in [0, C], & \alpha_i^*(\sum_{j=1}^{N}\mathcal{Q}_{ij}(\alpha_i - \alpha_i^*) + \zeta - y_i + \varepsilon - \delta_i^* + u_i^*) = 0 \\ \delta_i \geq 0, & \delta_i\alpha_i = 0 \\ \delta_i^* \geq 0, & \delta_i^*\alpha_i^* = 0 \\ u_i \geq 0, & u_i(\alpha_i - C) = 0 \\ u_i^* \geq 0, & u_i^*(\alpha_i^* - C) = 0 \end{cases}$$

A função estimada pode ser escrita como

$$f(\mathbf{x}_i) = \sum_{i=1}^{N}\mathcal{Q}_{ij}(\alpha - \alpha^*) + \zeta$$

onde ζ passrá a ser denomindo b na equação 3.1 para melhor compreensão. A função margem $h(\mathbf{x}_i)$ é dada por

$$h(\mathbf{x}_i) = f(\mathbf{x}_i) - y_i.$$

Substituindo nas condições ótimas :

$$\begin{cases} \alpha_i \in [0, C], & \alpha_i(\sum_{j=1}^{N} h(\mathbf{x}_i) + \zeta - y_i + \varepsilon - \delta_i + u_i) = 0, \\ \alpha_i^* \in [0, C], & \alpha_i^*(\sum_{j=1}^{N} h(\mathbf{x}_i) + \zeta - y_i + \varepsilon - \delta_i^* + u_i^*) = 0, \\ \delta_i \geq 0, & \delta_i \alpha_i = 0, \\ \delta_i^* \geq 0, & \delta_i^* \alpha_i^* = 0, \\ u_i \geq 0, & u_i(\alpha_i - C) = 0, \\ u_i^* \geq 0, & u_i^*(\alpha_i^* - C) = 0. \end{cases}$$

Pode-se relacionar $h(\mathbf{x})$ e ε com uma mudança em α. Quando $\alpha_i^* = 0$ ou $\alpha_i = 0$:

$$\begin{aligned} \alpha_i = 0 &\implies \alpha_i^* > 0, \delta_i \geq 0, u_i = 0 \\ &\implies h(\mathbf{x}_i) + \varepsilon - \delta_i = 0 \\ &\implies h(\mathbf{x}_i) = -\varepsilon + \delta_i \\ &\implies h(\mathbf{x}_i) \in [-\varepsilon, 0], \end{aligned}$$

$$\begin{aligned} \alpha_i^* = 0 &\implies \alpha_i > 0, \delta_i^* \geq 0, u_i^* = 0 \\ &\implies h(\mathbf{x}_i) + \varepsilon - \delta_i^* = 0 \\ &\implies h(\mathbf{x}_i) = -\varepsilon + \delta_i^* \\ &\implies h(\mathbf{x}_i) \in [0, \varepsilon]. \end{aligned}$$

Já quando $\alpha_i^* \in (0, C)$ ou $\alpha_i \in (0, C)$:

$$\begin{aligned} 0 < \alpha_i < C &\implies \alpha_i^* = 0, \delta_i = 0, u_i = 0 \\ &\implies h(\mathbf{x}_i) + \varepsilon = 0 \\ &\implies h(\mathbf{x}_i) = -\varepsilon, \end{aligned}$$

$$\begin{aligned} 0 < \alpha_i^* < C &\implies \alpha_i = 0, \delta_i^* = 0, u_i^* = 0 \\ &\implies h(\mathbf{x}_i) + \varepsilon = 0 \\ &\implies h(\mathbf{x}_i) = +\varepsilon. \end{aligned}$$

E quando $\alpha_i = C$ ou $\alpha_i^* = C$

$$\begin{aligned} \alpha_i = C &\implies \alpha_i^* = 0, \delta_i = 0, u_i \geq 0 \\ &\implies h(\mathbf{x}_i) + \varepsilon + u_i = 0 \\ &\implies h(\mathbf{x}_i) = -\varepsilon - u_i \\ &\implies h(\mathbf{x}_i) \leq -\varepsilon, \end{aligned}$$

$$\begin{aligned} \alpha_i^* = C &\implies \alpha_i = 0, \delta_i^* = 0, u_i^* \geq 0 \\ &\implies -h(\mathbf{x}_i) + \varepsilon + u_i = 0 \\ &\implies h(\mathbf{x}_i) = \varepsilon + u_i^* \\ &\implies h(\mathbf{x}_i) \geq \varepsilon. \end{aligned}$$

Definindo o coeficiente θ_i como

$$\theta_i = \alpha_i - \alpha_i^*$$

para todo $\mathbf{x}_i$, é possível então classificá-lo em função da função margem:

$$\begin{cases} h(\mathbf{x}_i) \geq \varepsilon, & \theta_i = -C, \\ h(\mathbf{x}_i) = \varepsilon, & -C < \theta_i < 0, \\ -\varepsilon \leq h(\mathbf{x}_i) \leq \varepsilon, & \theta_i = 0, \\ h(\mathbf{x}_i) = \varepsilon, & 0 < \theta_i < C, \\ h(\mathbf{x}_i) \leq \varepsilon, & \theta_i = C. \end{cases} \tag{3.3}$$

Essas inequações formam um sistema de condições e o algoritmo para incrementar o SVR se baseia nos conjuntos formados por elas.

3.2 APRENDIZADO INCREMENTAL

Há diversas formas de fazer o aprendizado incremental. Aqui é a apresentada por Ma, Theiler e Perkins (2003), cujo nome é AOSVR (*Accurate On-line Support Vector Regression*). Esse método, junto com as correções do mesmo feito por Parrella (2007), servirá de base para o Q-SVR, modelo proposto desta dissertação.

3.2.1 Adicionando uma nova amostra

O objetivo do algoritmo incremental, ao adicionar uma nova amostra $\mathbf{x}_c$ ao conjunto de amostras, é o de calcular θ_c e as diferenças $\Delta\theta_i, \forall i \in U$, além de calcular a diferença Δb. Para tanto, é útil separar inicialmente as amostras em diferentes conjuntos que terão diferentes tratamentos.

De acordo com o resultado do valor calculado da Equação (3.3), os vetores podem ser classificados em 3 conjuntos (PARRELLA, 2007):

- Conjunto de vetores suporte da margem:

$$S = \{i | (\theta_i \in [0, +C] \cap h(\mathbf{x}_i) = -\varepsilon) \cup (\theta_i \in [-C, 0] \cup h(\mathbf{x}_i) = +\varepsilon)\}$$

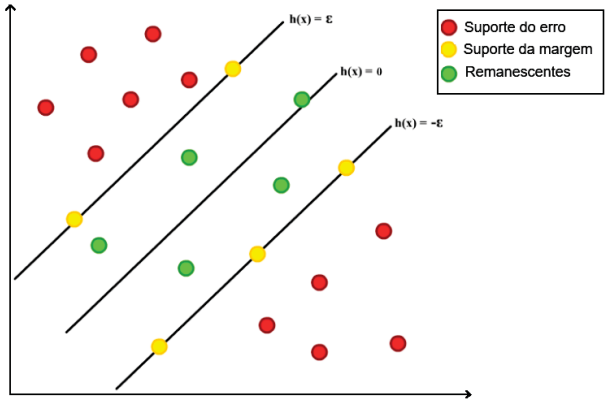
- Conjunto de vetores suporte do erro:

$$E = \{i | (\theta_i = -C \cap h(\mathbf{x}_i) \geq +\varepsilon) \cup (\theta_i = +C \cap h(\mathbf{x}_i) \leq -\varepsilon)\}$$

- Conjunto dos vetores remanescentes:

$$R = \{i | \theta_i = 0 \cap |h(\mathbf{x}_i)| \leq -\varepsilon\}$$

FIGURA 4 – Divisão das amostras em conjuntos



FONTE: adaptado de (PARRELLA, 2007).

A Figura 4 ilustra tais conjuntos, onde os pontos vermelhas representam as amostras que pertencem ao conjunto E, as amarelas ao conjunto S e as verdes ao conjunto R.

Supondo a introdução de uma nova amostra $\mathbf{x}_c$ em U, o algoritmo AOSVR prevê a Equação (3.2) deve continuar a mesma, logo

$$\theta_c + \sum_{i=1}^{N} \theta_i = 0$$

com θ_c inicialmente de valor zero. A nova função margem que será utilizada no novo dado terá a forma de

$$h(\mathbf{x}_i) = \sum_{j=1}^{N} \mathcal{Q}_{ij}\theta_j^{'} + Q_{ic}\theta_c^{'} + b^{'} - y_i.$$

As variações ao adicionar a nova amostra são dadas por

$$\Delta\theta_i = \theta_i^{'} - \theta_i \quad \text{e} \quad \Delta b = b^{'} - b.$$

A função margem tem a seguinte variação:

$$\Delta h(\mathbf{x}_i) = \mathcal{Q}_{ic}\Delta\theta_c + \sum_{i=1}^{N} \mathcal{Q}_{ij}\Delta\theta_j + \Delta b.$$

Como tem-se que a soma dos pesos antes e depois da adição da nova amostra tem que ser zero, tem se que

$$\sum_{i=1}^{N} \theta_i = 0 \quad \text{e} \quad \theta_c + \sum_{i=1}^{N} \theta_i^N = 0.$$

Surge, então, que

$$\Delta\theta_c + \Delta\sum_{i=1}^{N} \theta_i^N = 0 \implies \sum_{i}^{N} \Delta\theta_j = -\Delta\theta_c.$$

Define-se o conjunto de vetores suporte de elementos

$$S = \{s_1, s_2, \ldots, s_{l_s}\}.$$

Como, pelas propriedades dos conjuntos 3.3, apenas vetores do conjunto de vetores suporte da margem podem alterar θ_j, tem-se que

$$\sum_{j \in S} \mathcal{Q}_{ij} \Delta\theta_j + \Delta b = -\mathcal{Q}_{ic} \Delta\theta_c, \quad \text{onde } i \in S$$

$$\sum_{j \in S} \Delta\theta_j = -\Delta\theta_c$$

Reescrevendo as equações de forma matricial:

$$\begin{bmatrix} 0 & 1 & \cdots & 1 \\ 1 & \mathcal{Q}_{s_1 s_1} & \cdots & \mathcal{Q}_{s_1 s_{l_s}} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \mathcal{Q}_{s_{l_s} s_1} & \cdots & \mathcal{Q}_{s_{l_s} s_{l_s}} \end{bmatrix} \begin{bmatrix} \Delta b \\ \Delta\theta s_1 \\ \vdots \\ \Delta\theta s_{l_s} \end{bmatrix} = - \begin{bmatrix} 1 \\ \mathcal{Q}_{s_1 c} \\ \vdots \\ \mathcal{Q}_{s_{l_s} c} \end{bmatrix} \Delta\theta_c$$

Tambem pode-se escrever as variações de θ e b:

$$\begin{bmatrix} \Delta b \\ \Delta\theta s_1 \\ \vdots \\ \Delta\theta s_{l_s} \end{bmatrix} = \begin{bmatrix} 0 & 1 & \cdots & 1 \\ 1 & \mathcal{Q}_{s_1 s_1} & \cdots & \mathcal{Q}_{s_1 s_{l_s}} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \mathcal{Q}_{s_{l_s} s_1} & \cdots & \mathcal{Q}_{s_{l_s} s_{l_s}} \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ \mathcal{Q}_{s_1 c} \\ \vdots \\ \mathcal{Q}_{s_{l_s} c} \end{bmatrix} \Delta\theta_c$$

Definindo a matriz $\mathbf{R}$ como:

$$\mathbf{R} = \begin{bmatrix} 0 & 1 & \cdots & 1 \\ 1 & \mathcal{Q}_{s_1 s_1} & \cdots & \mathcal{Q}_{s_1 s_{l_s}} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \mathcal{Q}_{s_{l_s} s_1} & \cdots & \mathcal{Q}_{s_{l_s} s_{l_s}} \end{bmatrix}^{-1}$$

o vetor β pode ser definido em função de $\mathbf{R}$:

$$\beta = \begin{bmatrix} \beta_b \\ \beta_{s_1} \\ \vdots \\ \beta_{s_{l_s}} \end{bmatrix} = -\mathbf{R} \begin{bmatrix} 1 \\ \mathcal{Q}_{s_1 c} \\ \vdots \\ \mathcal{Q}_{s_{l_s} c} \end{bmatrix} \tag{3.4}$$

Para os vetores em S, a h não varia dado que é sempre $|\varepsilon|$. A variação dos valores β e θ é dada por

$$\begin{bmatrix} \Delta b \\ \Delta\theta_{s_1} \\ \vdots \\ \Delta\theta_{s_{l_s}} \end{bmatrix} = \beta \Delta\theta_c = \begin{bmatrix} \beta_b \\ \beta_{s_1} \\ \vdots \\ \beta_{s_{l_s}} \end{bmatrix} \Delta\theta_c. \tag{3.5}$$

Para os vetores de E e de R, o h muda. Definindo o conjunto dos vetores não-suporte:

$$N = E \cup R = \{n_1, n_2, \ldots, n_{l_n}\}.$$

A variação na forma matricial de h para estas amostras é dada por:

$$\begin{bmatrix} \Delta h(\mathbf{x}_{n_1}) \\ \vdots \\ \Delta h(\mathbf{x}_{n_{l_n}}) \end{bmatrix} = \begin{bmatrix} \Delta \mathcal{Q}_{n_1 c} \\ \vdots \\ \Delta \mathcal{Q}_{n_{l_n} c} \end{bmatrix} \Delta\theta_c + \begin{bmatrix} 0 & 1 & \cdots & 1 \\ 1 & \mathcal{Q}_{n_1 s_1} & \cdots & \mathcal{Q}_{n_1 s_{l_s}} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \mathcal{Q}_{n_{l_n} s_1} & \cdots & \mathcal{Q}_{n_{l_n} s_{l_s}} \end{bmatrix} \begin{bmatrix} \Delta b \\ \Delta\theta_{s_1} \\ \vdots \\ \Delta\theta_{s_{l_s}} \end{bmatrix}.$$

Substituindo as variações de θ em b

$$\begin{bmatrix} \Delta h(\mathbf{x}_{n_1}) \\ \vdots \\ \Delta h(\mathbf{x}_{n_{l_n}}) \end{bmatrix} = \begin{bmatrix} \Delta \mathcal{Q}_{n_1 c} \\ \vdots \\ \Delta \mathcal{Q}_{n_{l_n} c} \end{bmatrix} \Delta\theta_c + \begin{bmatrix} 0 & 1 & \cdots & 1 \\ 1 & \mathcal{Q}_{n_1 s_1} & \cdots & \mathcal{Q}_{n_1 s_{l_s}} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \mathcal{Q}_{n_{l_n} s_1} & \cdots & \mathcal{Q}_{n_{l_n} s_{l_s}} \end{bmatrix} \beta\Delta\theta_c.$$

Sendo γ é definida como

$$\gamma = \begin{bmatrix} \Delta \mathcal{Q}_{n_1 c} \\ \vdots \\ \Delta \mathcal{Q}_{n_{l_n} c} \end{bmatrix} + \begin{bmatrix} 0 & 1 & \cdots & 1 \\ 1 & \mathcal{Q}_{n_1 s_1} & \cdots & \mathcal{Q}_{n_1 s_{l_s}} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \mathcal{Q}_{n_{l_n} s_1} & \cdots & \mathcal{Q}_{n_{l_n} s_{l_s}} \end{bmatrix} \beta, \tag{3.6}$$

então, a variação h fica:

$$\begin{bmatrix} \Delta h(\mathbf{x}_{n_1}) \\ \vdots \\ \Delta h(\mathbf{x}_{n_{l_n}}) \end{bmatrix} = \gamma\Delta\theta_c \tag{3.7}$$

Dessa forma, $\Delta\theta_i$ e Δb podem ser calculados usando β e $\Delta(\mathbf{x}_i)$ podem ser calculados usando γ. Caso o conjunto de suporte esteja vazio, os valores são atualizados da seguinte forma:

$$\Delta b = \Delta\theta_c,$$

$$\Delta h(\mathbf{x}_i) = \Delta b, i = 1, \ldots, N.$$

Parrella (2007) mostra um método eficiente para atualizar a matriz $\mathbf{R}$ ao adicionar uma amostra em S. A matriz $\mathbf{R}$ foi definida como

$$\mathbf{R} = \begin{bmatrix} 0 & 1 & \cdots & 1 \\ 1 & \mathcal{Q}_{s_1 s_1} & \cdots & \mathcal{Q}_{s_1 s_{l_s}} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \mathcal{Q}_{s_{l_s} s_1} & \cdots & \mathcal{Q}_{s_{l_s} s_{l_s}} \end{bmatrix}^{-1}.$$

A primeira amostra deve ser adicionada à matriz da seguinte forma:

$$\mathbf{R} = \begin{bmatrix} -\mathcal{Q}_{11} & 1 \\ 1 & 0 \end{bmatrix}$$

No caso das demais amostras, a forma mais eficiente de atualizar a matriz é com a seguinte equação:

$$\mathbf{R}_{\text{atualizada}} = \begin{bmatrix} & & 0 \\ & \mathbf{R}_{\text{antiga}} & \vdots \\ & & 0 \\ 0 & 0 \quad \dots & 0 \end{bmatrix} + \frac{1}{\gamma_i} \begin{bmatrix} \beta \\ 1 \end{bmatrix} \begin{bmatrix} \beta^T & 1 \end{bmatrix}. \tag{3.8}$$

Se a nova amostra é adicionada ao conjunto S, γ_i é definida como:

$$\gamma_i = Q_{CC} + \begin{bmatrix} 1 & Q_{Cs_1} & \dots & Q_{Cs_{l_s}} \end{bmatrix} \beta.$$

Já no caso onde a nova amostra é movida do conjunto E ou R para S, β e γ_i devem ser recalculadas como:

$$\beta = -\mathbf{R} \begin{bmatrix} 1 \\ Q_{is_1} \\ \vdots \\ Q_{is_{l_s}} \end{bmatrix},$$

$$\gamma_i = Q_{ii} + \begin{bmatrix} 1 & Q_{is_1} & \dots & Q_{is_{l_s}} \end{bmatrix} \beta.$$

Caso a k-ésima amostra seja removida do conjunto S, a linha e a coluna são removidas da matriz, fazendo com que a nova matriz $\mathbf{R}$ possa ser atualizada por

$$\mathbf{R}_{\text{atualizada}} = \mathbf{R}_{\mathbf{I},\mathbf{I}} - \frac{\mathbf{R}_{\mathbf{I},k}\mathbf{R}_{\mathbf{I},k}}{\mathbf{R}_{k,k}} \tag{3.9}$$

onde $\mathbf{I} = \begin{bmatrix} 1 & \cdots & k-2 & k-1 & k+1 & k+2 & \cdots & s_{l_s} \end{bmatrix}$ é uma vetor de valores que indicam a coluna ou linha a serem utilizadas, $\mathbf{R}_{\mathbf{I},\mathbf{I}}$ é a matriz com apenas as colunas do vetor $\mathbf{I}$ e $\mathbf{R}_{\mathbf{I},k}$ é um vetor coluna com as linhas $\mathbf{I}$ e a coluna k da matriz $\mathbf{R}$.

3.2.2 Algoritmo do AOSVR

O Algoritmo 3 mostra o procedimento para incrementar uma nova amostra. Para fins de facilitar a exposição do algoritmo, o mesmo foi dividido em três partes, onde a primeira define a situação mais simples, o segunda calcula e determina uma dos casos possíveis e, por fim, a terceira toma uma decisão com base no resultado da segunda parte.

A Parte 1 do Algoritmo 3 checa se $\mathbf{x}_c$ pode ser incluída no conjunto R, uma vez que esse é o caso onde há o menor esforço computacional. Para tanto, calcula $f(\mathbf{x}_c), h(\mathbf{x}_c)$ e checa se $|h(\mathbf{x}_c)| \leq \varepsilon$. Caso for, insere $\mathbf{x}_c$ em R e termina o algoritmo.

Quando a amostra não é inserida no conjunto R, a mesma deve ser inserida ou em E ou em S. É iniciado, então, um loop que só termina quando a mesma é inserida em um dos dois grupos. No loop, as outras amostras mudam de conjunto, alterando seus valores θ_i e θ_c.

Na Parte 2 do algoritmo define-se a direção de $\Delta\theta_c$ e depois inicia-se o loop. Nele, atualiza-se inicialmente os valores β e γ. Dá-se início ao procedimento Bookeeping, cujo objetivo é definir o valor da variável Flag. É calculado, então, para cada uma das amostras em cada um dos conjuntos, uma variação denominada pela letra L [1]e calculada de acordo com os valores de β e γ. A amostra $\mathbf{x}_a$ escolhida será a que tem o menor valor (em módulo) de todo o seu conjunto e de todos os conjuntos, sendo essa a utilizada para calcular $\Delta\theta_c$ e, a depender de qual conjunto ela pertence, determinará valor da variável Flag como definida no Algortimo 3. Por fim, as variáveis θ_c, b e $\theta_i, i \in S$ são atualizadas, assim como os valores da função margem $h(\mathbf{x}_i)$.

A Parte 3 do Algoritmo 3 define se o loop vai continuar ou se vai parar. Para que o mesmo pare, é necessário que o valor da variável Flag seja 1, onde a amostra $\mathbf{x}_c$ será adicionada ao conjunto S, caracterizando um vetor de suporte. Outra situação em que o loop para é quando o valor da variável Flag é 2, onde a amostra será adicionada ao conjunto E. Nos dois casos a matriz $\mathbf{R}$ deve ser atualizada com a Equação (3.8) dado que se adicionou uma amostra ao conjunto S.

Para o outros valores da variável Flag, o loop vai continuar. No caso dos valores 4 ou 5, a amostra $\mathbf{x}_a$ é movida ao conjunto S a matriz $\mathbf{R}$ atualizada (como se adicionou uma amostra à S) com a Equação (3.8). No caso do valor de Flag ser 3, caso θ_N seja 0, deve-se mover $\mathbf{x}_a$ para R ou, se $|\theta_N|$ seja igual à C, deve-se mover ela para o conjunto E. A matriz $\mathbf{R}$ deve ser atualizada, neste caso, com a Equação 3.9.

[1] A letra L utilizada vem do inglês *Least Variations* como utilizada por Parrella (2007).

Algoritmo 3 *AOSVR* para incrementar uma nova amostra

Entrada: Conjunto de dados $U = (\mathbf{x}_i, y_i), i = 1, \ldots, N$
Os coeficientes $\theta_i, i = 1, \ldots, N$ e b
Amostras em conjuntos S, E e R
Matriz $\mathbf{R}$
A nova amostra $(\mathbf{x}_c, y_c)$
Os parâmetros d (para calcular $\mathcal{Q}_{ij}$), C e ϵ

Saída: Os coeficientes atualizados b e $\theta_i, i = 1, \ldots, N + 1$
Os conjuntos de amostras R, S e E atualizadas
Matriz $\mathbf{R}$ atualizada

(Parte 1) Defina $\theta_c = 0$
Calcule $f(\mathbf{x}_c) = \sum_{i \in E \cup S} \theta_i \mathcal{Q}_{ic} + b$
Calcule $h(\mathbf{x}_c) = f(\mathbf{x}_c) - y_c$
Se $|h(\mathbf{x}_c)| \leq \varepsilon$, inserir $\mathbf{x}_c$ para R e termine o algoritmo
(Parte 2)
Defina a direção de $\Delta\theta_c$ como $q = \text{sign}(-h(x_c))$
enquanto a amostra $\mathbf{x}_c$ não for adicionada a um conjunto **faça**:
Atualizar β, γ de acordo com as equações 3.4 e 3.6
Começar o procedimento Bookeeping:
Calcular os valores para $\mathbf{x}_c$:
- $Lc_1 = (-h(\mathbf{x}_c) - q\varepsilon)/\gamma_C$ (caso 1)
- $Lc_2 = qC - \theta_C$ (caso 2)
Calcular para cada amostra $\mathbf{x}_i$ no conjunto S (caso 3)
- Se $q\beta_i > 0$ e $C > \theta_i \geq 0$ então $L_i^S = (C - \theta_i)\beta_i$
- Se $q\beta_i > 0$ e $0 > \theta_i \geq -C$ então $L_i^S = -\theta_i/\beta_i$
- Se $q\beta_i > 0$ e $C \geq \theta_i > 0$ então $L_i^S = -\theta_i/\beta_i$
- Se $q\beta_i < 0$ e $0 \geq \theta_i > 0$ então $L_i^S = (-C - \theta_i)\beta_i$
Calcular para cada amostra $\mathbf{x}_i$ no conjunto E (caso 4)
- $L_i^E = (-h(\mathbf{x}_i) - \text{sign}(q\beta_i)\varepsilon)/\beta_i$
Calcular para cada amostra $\mathbf{x}_i$ no conjunto R (caso 5)
- $L_i^R = (-h(\mathbf{x}_i) - \text{sign}(q\beta_i)\varepsilon)/\beta_i$
Seja $\Delta\theta_c = q \min(|L_{c1}|, |L_{c2}|, |L^S|, |L^E|, |L^R|)$ onde $L^S = \{L_i^S, i \in S\}, L^E = \{L_i^E, i \in E\}$ e $L^R = \{L_i^R, i \in R\}$.
Seja Flag o número do caso que determina $\Delta\theta$.
Seja $\mathbf{x}_a$ a amostra particular em U que determina $\Delta\theta_c$.
Finaliza o procedimento Bookeeping.
Atualizar θ_c, b e $\theta_i, i \in S$ de acordo com a Equação 3.5
Atualizar $h(\mathbf{x}_i), i \in E \cup R$ de acordo com a Equação 3.7
(Parte 3) Escolha:
Caso Flag=1: Adicionar a amostra $\mathbf{x}_c$ para o conjunto S; atualizar a matriz $\mathbf{R}$ de acordo com a Equação 3.8
Caso Flag=2: Adicionar a amostra $\mathbf{x}_c$ no conjunto E;
Caso Flag=3: Se $\theta_N = 0$, mover $\mathbf{x}_a$ para o conjunto R; Se $|\theta_N| = C$, mover $\mathbf{x}_a$ para o conjunto E; atualizar a matriz $\mathbf{R}$ de acordo com a Equação 3.9;
Caso Flag=4: Mover $\mathbf{x}_a$ para o conjunto S; atualizar $\mathbf{R}$ de acordo com a Equação 3.8;
Caso Flag=5: Mover $\mathbf{x}_a$ para o conjunto S; atualizar $\mathbf{R}$ de acordo com a Equação 3.8;
Se Flag ≤ 2 termine a execução; caso contrário continue o loop;
fim enquanto

3.3 PROPOSTA DO Q-SVR

Uma vez descrito o algoritmo AOSVR, é possível apresentar a proposta do algoritmo Q-SVR. O método proposto consiste em representar a função que aproxima a ação-valor da seguinte forma:

$$\check{Q}(s_t, a_t) = \mathbf{w}^T \phi(s_t, a_t) + b.$$

A função $\check{Q}(\cdot)$ é uma aproximação não-linear da função $Q(\cdot)$, uma vez que função $\phi(\cdot)$ é não-linear. Pela definição da função $q_\pi(s, a)$, pode-se escrever então

$$q_\pi(s_t, a_t) = \mathbb{E}_\pi\Big[G_t | S_t = s_t, A_t = a_t\Big]$$

O retorno pode ser escrito como

$$\begin{aligned} G_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \gamma^4 r_{t+5} + \dots \\ G_t &= r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} + \gamma^3 r_{t+5} + \dots) \\ G_t &= r_{t+1} + \gamma G_{t+1}. \end{aligned}$$

Então pode-se escrever, com a equação de avaliação linear de Bellman, que

$$q_\pi(s_t, a_t) = \mathbb{E}_\pi\Big[r_{t+1} + \gamma G_{t+1} | S_t = s_t, A_t = a_t\Big].$$

Quando $\pi = \pi_*$, a função ação-valor ótima $q_*(\cdot)$ satisfaz a seguinte relação:

$$q_*(s_t, a_t) = \mathbb{E}_\pi\Big[r_{t+1} + \gamma \max_{a_{t+1}} q_*(s_{t+1}, a_{t+1}) | S_t = s_t, A_t = a_t\Big].$$

Substituindo pelas funções estimadas, temos:

$$\check{Q}(s, a) = r_{t+1} + \gamma \max_{a_{t+1}} \check{Q}(s_{t+1}, a_{t+1}) \tag{3.10}$$

Como argumenta Mnih et al. (2013), essa variável alvo é utilizado por vários outros algoritmos que utilizam função para aproximar as funções valor.

A variável de entrada do Q-SVR pode ser reescrita sendo um vetor onde o vetor de estado e de ação são empilhados, ou seja, $\mathbf{x}_t = [s_t, a_t]$. As ações, assim como os estados, devem ser mapeadas para um número real para que $\mathbf{x}$ seja um vetor numérico como, por exemplo, $a = 1$ quando a ação refere-se ao agente ir para a direita e $a = 0$ quando deve ir para a esquerda. Para prever o valor de um novo par estado-ação, utiliza-se a relação

$$\check{Q}(\mathbf{x}) = \sum_{i=1}^{N} \theta_i k(\mathbf{x}_i, \mathbf{x}) + b \tag{3.11}$$

sendo $k(\cdot)$ é uma função kernel arbitrária e θ_i são os parâmetros estimados pelo AOSVR. Então, a Equação (3.11) é uma função que aproxima a função da Equação (3.10).

Uma vez definida a variável alvo e as variáveis de entrada, o Q-SVR funciona da seguinte maneira. Inicialmente um modelo SVR com T, S, E, R vazios e a matriz $\mathbf{R}$ nula é inicializado, sendo esse o agente em questão. O agente vai interagir com o ambiente e capturar s_t, tomando ações com base em uma política derivada de $\check{Q}$ (ϵ-gulosa, por exemplo) por até M passos ou até o estado ser um estado terminal. Após cada episódio, atualiza o modelo SVR até que o número de episódios seja alcançado. O Algoritmo 4 ilustra o procedimento completo.

Algoritmo 4 *Q-SVR*

Entrada: C, ε, d
Inicializar a função $\check{Q}$ com o modelo SVR(C, ε, d) com T, S, E, R vazios e a matriz $\mathbf{R}$ nula
para episódio **faça**
Inicialize s_t de acordo com o ambiente
enquanto s_t não for terminal ou número de passos for menor que o M **faça**
Escolha a_t de s_t de acordo com a política derivada de $\check{Q}$ (por ex. ϵ-gulosa)
Execute ação a_t e observe r_{t+1} e s_{t+1}
$$y_t \leftarrow \begin{cases} r_{t+1} & \text{, se o episódio terminar em } t+1 \\ r_{t+1} + \gamma \max_{a_{t+1}} \check{Q}(s_{t+1}, a_{t+1}) & \text{, caso contrário} \end{cases}$$
Incremente o modelo de acordo com o Algoritmo 3 com $\mathbf{x}_c = [s_t, a_t]$ e $y_c = y_t$
$s_t \leftarrow s_{t+1}$
fim enquanto
fim para

4 EXPERIMENTOS

Visando testar a factibilidade do algoritmo proposto, foram realizados testes em ambientes com estados discretos. Os resultados dos experimentos foram comparados com os de dois outros modelos: o *Q-Learning* e o DQN. Os dois foram escolhidos por terem arquiteturas diferentes: o primeiro ser um algoritmo tabular enquanto o segundo ser um algoritmo de aproximação de função.

A primeira seção deste capítulo é dedicada a elucidar os detalhes sobre a implementação do modelo e dos modelos que foram comparados, assim como os hiperparâmetros utilizados nos testes. A seguir são apresentados os três ambientes usados como ambientes de teste para o algoritmo junto com o resultado dos testes realizados em cada um deles.

4.1 CONFIGURAÇÕES INICIAIS

A implementação do algoritmo Q-SVR[1] foi feita em Python 3.8. Esta usa como base duas implementações do AOSVR: a de Francesco Parrella (PARRELA, 2024), que escreveu o código em Matlab e C++, e a de Adam Werries (WERRIES, 2024), que adaptou o código para a linguagem Python. O computador utilizado para a implementação do Q-SVR foi equipado com um processador AMD *Ryzen* 5700x3D, 16 *gigabytes* de memória RAM e o sistema operacional Ubuntu.

No que tange à implementação dos modelos de comparação, o Q-Learning foi implementado pelo autor seguindo o algoritmo em Sutton e Barto (2020) enquanto a DQN foi adaptado da implementação de Zhang (2024). É importante salientar que a última utiliza apenas a CPU, não tendo o processamento paralelizado com o uso de GPU.

Os hiperparâmetros utilizados foram constantes em todos os três ambientes. Para o Q-SVR, foi utilizado a função kernel RBF Gaussiano com $d = 0.7$, $C = 1$ e $\varepsilon = 0.1$, valores baseados nos trabalhos de Ma, Theiler e Perkins (2003) e Christmann e Steinwart (2008). Para os modelos de comparação, no *Q-Learning* a taxa de aprendizado foi utilizado o valor 0.1. Já para a DQN foi utilizado a taxa de aprendizado igual à 10^{-4}, tamanho do *batch* de 32 e a frequência da atualização igual à 5 episódios, sendo que os valores foram baseados em An et al. (2018). Em todos os experimentos foram utilizados $\epsilon_0 = 0.99$ e $\epsilon_d = 0.99$.

[1] A implementação utilizada do algoritmo está disponível em https://github.com/vinybrasil/qsvr.

4.2 AMBIENTES TESTADOS

Serão testados três ambientes: a caminhada aleatória, a caminhada em cadeia de 4 estados e ao problema do caminho mínimo. Os três ambientes têm a característica de ter o seu espaço de estados discreto cujos podem ser representados como grafos.

4.2.1 Caminhada aleatória

O ambiente, como apresentando por An et al. (2018), é um grafo com n nós onde o agente inicia no nó central do mesmo, tendo $\lfloor \frac{n}{2} \rfloor$ nós à sua direita e $\lfloor \frac{n}{2} \rfloor$ nós à esquerda. As ações possíveis são se mover para o nó da direita (D) ou da esquerda (E). Caso ele chegue na extremidade mais à direita, recebe a recompensa arbitrária de +10 e o episódio acaba. Caso o agente chegue no nó mais à esquerda, recebe a recompensa -10 e o episódio também acaba. O agente tem um limite arbitrário de $5n$ passos possíveis até o episódio acabar. A política ótima para esse ambiente é $(D, D, \ldots, D)$. A Figura 5 ilustra o ambiente.

FIGURA 5 – Caminhada aleatória.

FONTE: An et al. (2018).

Foram realizados 10 testes com sementes aleatórias diferentes. Os agentes agiram durante 150 episódios em três ambientes. Cada ambiente contava com um número de nós (n) diferentes, onde $n \in \{7, 11, 15\}$ escolhidos como em An et al. (2018).

Em cada um dos episódios, é testada a política que o algoritmo considera a melhor naquele momento. Caso a política não mude até o final dos episódios, é considerado que a política convergiu.

A Tabela 2 mostra a porcentagem dos testes de cada algoritmo que convergiram para a política ótima para os diferentes valores de n. É possível notar que o Q-SVR tem a performace degradada com o aumento no número de estados, fazendo com que, no caso do ambiente com o maior número de estados, venha a convergir em apenas 2 dos 10 testes. Já os modelos de comparação não apresentam tal característica onde, no caso do *Q-Learning*, o resultado até melhorou com o aumento no número de estados, convergindo em todos os testes para a política ótima.

TABELA 2 – Porcentagem dos testes que convergiram para a política ótima.

n	Q-SVR	Q-Learning	DQN
7	80%	90%	90%
11	50%	90%	90%
15	10%	100%	90%

FONTE: O autor (2024).

Apesar de ter o melhor resultado na quantidade de testes que convergiram, o Q-Learning tem o tempo esperado maior dos três algoritmos para convergir. É o que mostra a Tabela 3, onde estão as medianas dos números de episódios até que, a partir deste episódio, o algoritmo considere a política como política ótima. Nesta tabela as medianas foram calculadas apenas nos casos em que o algoritmo convergiu. É possível ver também nessa tabela que o Q-SVR teve um bom resultado relativo ao *Q-Learning* e tendo as medianas parecida com o melhor dos três, o DQN. Este, apesar de ter a melhor mediana, é o algoritmo que tem os maiores números máximos de episódios até a convergência, chegando a 139 episódios no pior caso quando $n = 7$, como mostra a Tabela 4. O Q-SVR teve o melhor resultado entre os três algoritmos, convergindo em no máximo 7 iterações quando $n = 7$.

TABELA 3 – Mediana dos números de episódios até a estabilização na política ótima.

n	Q-SVR	Q-Learning	DQN
7	1,5	7	1
11	1	6	1
15	1,5	6,5	1

FONTE: O autor (2024).

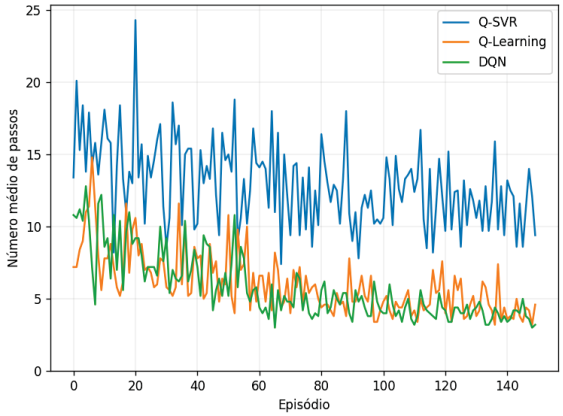
Com relação ao número médio de passos, o Q-SVR tem uma queda significativa com o aumento no número de episódios apenas quando no ambiente com 7 estados, como mostra a Figura 6. Nos outros dois ambientes não há a redução característica do número médio de passos como ilustra a Figura 7 para o caso dos 15 estados, sendo que tal redução esperada acontece nos dois algoritmos de comparação. Os gráficos em todos os testes individuais podem ser encontrados no Anexo A.

TABELA 4 – Máximo do números de episódios até a estabilização na política ótima.

n	Q-SVR	Q-Learning	DQN
7	7	18	139
11	6	13	70
15	2	14	22

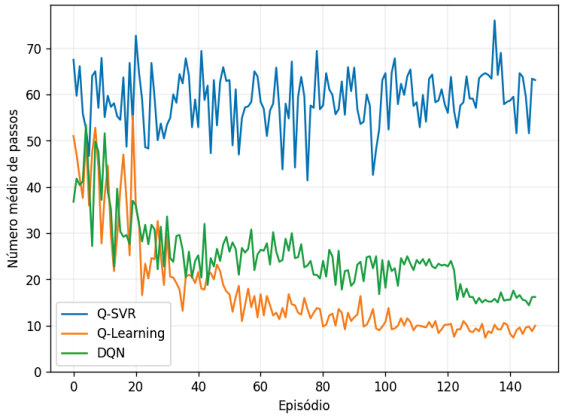
FONTE: O autor (2024).

FIGURA 6 – Número médio de passos no ambiente caminhada aleatória com 7 estados.



FONTE: O autor (2024).

FIGURA 7 – Número médio de passos no ambiente caminhada aleatória com 15 estados.

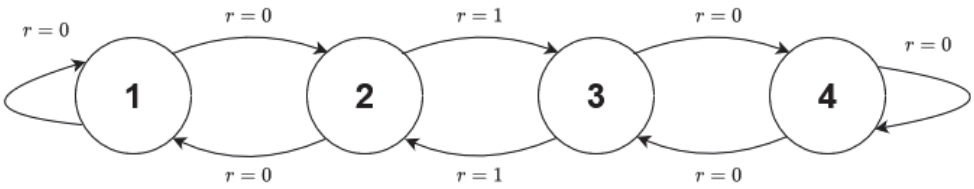


FONTE: O autor (2024).

4.2.2 Caminhada em cadeia de 4 estados

O ambiente, como apresentado em Lee et al. (2009), consiste em um grafo onde cada nó representa um dos 4 estados possíveis ($S = \{1, 2, 3, 4\}$). As ações possíveis (a) são ir para a direita (D) ou ir para a esquerda (E). As recompensas são as descritas na Tabela 5 e o grafo é ilustrado na Figura 8.

FIGURA 8 – Cadeia com quatro estados



FONTE: Lee et al. (2009).

TABELA 5 – Função de recompensa do ambiente da caminhada em cadeia de 4 estados

s_t	a_t	s_{t+1}	r_{t+1}
1	E	1	0
1	D	2	0
2	E	1	0
2	D	3	1
3	E	2	1
3	D	4	0
4	E	3	0
4	D	4	0

FONTE: O autor (2024).

Se o agente decide tomar uma ação, tem 90% de chance da ação ser executada e 10% de executar a ação contrária. O agente tem o limite de 4 ações até o episódio acabar. A política ótima para o ambiente é $\pi(\mathbf{D}|1) = 1$, $\pi(\mathbf{D}|2) = 1$, $\pi(\mathbf{E}|3) = 1$ e $\pi(\mathbf{E}|4) = 1$.

Foram realizados 10 testes com sementes aleatórias diferentes para cada um dos três algoritmos testados. A Tabela 6 mostra a porcentagem de testes que convergiram para a política ótima. Há uma clara superioridade do algoritmo *Q-Learning*, onde o mesmo convergiu para a política ótima em quase todos os testes realizados. Já o QSVR teve um resultado parecido com a DQN, embora o último se sobressaia sobre o anterior. A convergência aqui é definida como a iteração chega na política ótima e se estabiliza ao não mudar.

Para os testes que estimaram a política ótima corretamente, a Tabela 7 mostra as medianas dos números de episódios até que chegassem à essa conclusão, além do mínimo e do máximo número de episódios. O Q-SVR teve o melho resultado, cuja mediana de 10,5 é menos da metade do segundo algoritmo melhor colocado, o *Q-Learning*. O resultado do algoritmo proposto é também melhor quando se compara o valor máximo ao *Q-Learning*, sendo o primeiro tendo o pior caso um número de episódios pouco acima da metade do pior caso no Q-Learning, cujos valores são 27 e 44, respectivamente. No único caso que chegou na política ótima, o DQN teve um

TABELA 6 – Porcentagem de testes que convergiram para a política ótima por algoritmo na cadeia com quatro estados.

	Política ótima	Outra política
Q-SVR	100%	0%
Q-Learning	100%	0%
DQN	10%	90%

FONTE: O autor (2024).

TABELA 7 – Medianas dos números de episódios até a política ser a política ótima na cadeia com quatro estados.

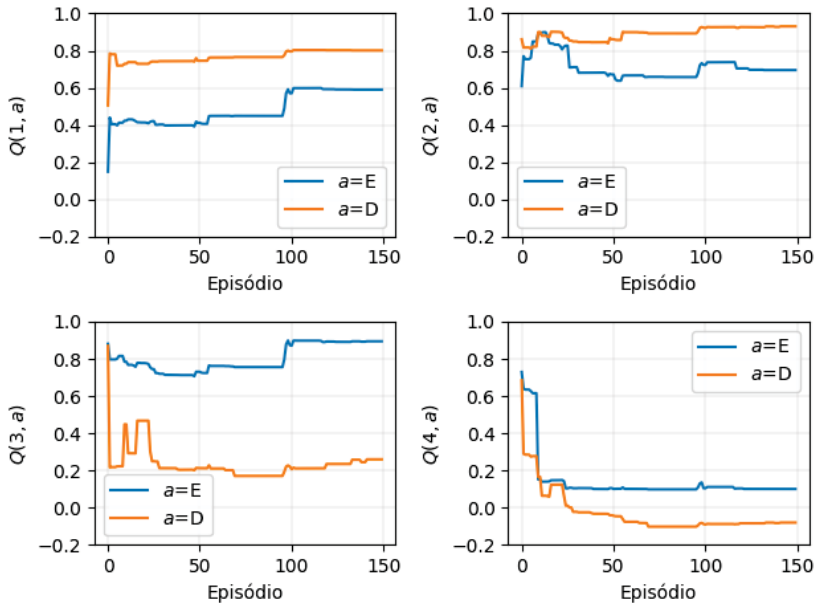
	mediana	máximo	mínimo	mínimo sem 1
Q-SVR	10,5	27	1	2
Q-Learning	24	44	8	8
DQN	192	192	192	192

FONTE: O autor (2024).

resultado muito pior que os outros dois algoritmos, onde demorou 192 episódios para chegar nela.

No que tange aos valores mínimos, há casos onde o valor mínimo é 1 na Tabela 7. Neles, já após o primeiro episódio, a política gerada é a política ótima. A Figura 9 ilustra os valores estimados da função ação-valor pelo Q-SVR em um dos dois desses casos. Quando se retira esses casos, pode-se ver que mesmo assim o Q-SVR tem um resultado bom, levando apenas 2 episódios para chegar à política ótima.

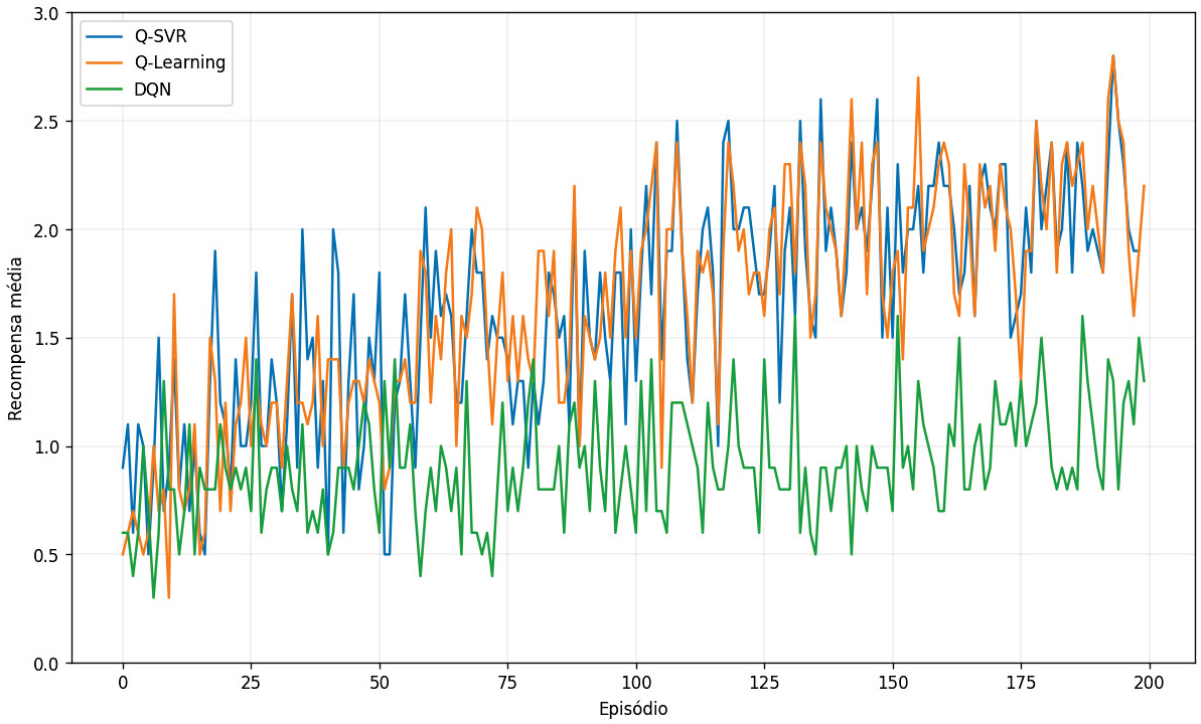
FIGURA 9 – Valores estimados do Q-SVR que geram a política ótima desde o primeiro episódio



FONTE: O autor (2024).

A Figura 10 ilustra a recompensa média dos testes no ambiente testado. Os algoritmos Q-SVR e *Q-Learning* apresentam um aumento na recompensa média com o número de episódios embora não seja possível eleger um acima do outro, uma vez que o desempenho é parecido no que tange à recompensa média. Entretando, é possível eleger que o desempenho do DQN é consideravelmente pior, uma vez que o seu aumento da recompensa média é menor que os outros algoritmos quando o número de episódios aumenta.

FIGURA 10 – Recompensa média na cadeia com quatro estados.



FONTE: O autor (2024).

4.2.3 Caminho mínimo em um grafo ponderado

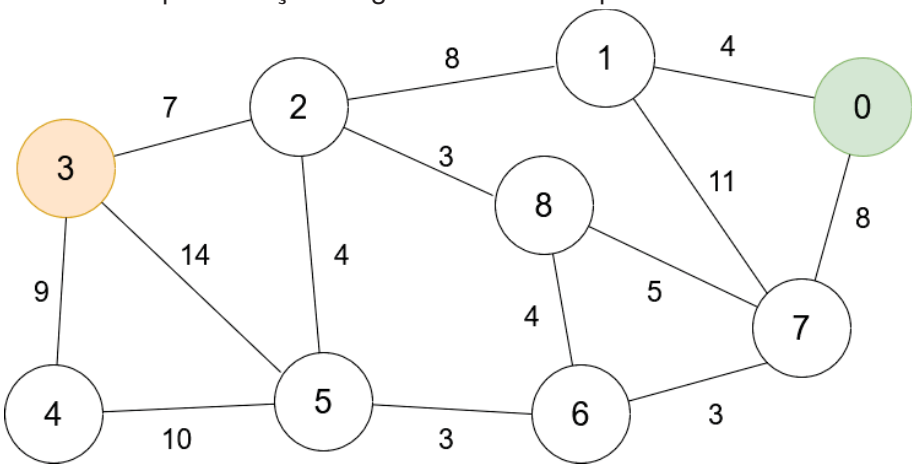
O ambiente consiste em um grafo com 9 nós cujo objetivo é sair de um nó arbitrário chamado de fonte e chegar em outro nó arbitário chamado de alvo. Os estados são um número indicando qual nó o agente está ($S = \{0, 1, \ldots, 8\}$). As ações disponíveis ao agente são ir nos nós a qual o nó em que ele se encontra estão conectados. A recompensa é $-c$ para cada mudança de nó, sendo c o peso da aresta, o que incentiva o agente a tomar o caminho com o menor custo possível.

Em cada episódio, o agente tem o limite de 150 passos antes que o episódio acabe. O grafo do experimento é adaptado de Geeks for Geeks(2024) e representado na Figura 11. No experimento, vértice 3 é o nó fonte e o vértice 0 é o nó alvo, sendo representado em laranja e verde, respectivamente.

O caminho ótimo é, saindo de 3, ir para o nó 2, depois para o nó 1 e, finalmente, ir para o nó 0, sendo representado por (3, 2, 1, 0). Este caminho tem a recompensa de -19, a maior recompensa entre todas as disponíveis.

Foram realizados 10 testes com sementes aleatórias distintas onde os agentes tentavam encontrar o caminho ótimo durante 150 episódios. Após cada episódio, a política é avaliada para ver qual caminho ela gera. A convergência é definida como se o caminho não se altera após chegar no mesmo com o passar dos episódios.

FIGURA 11 – Representação do grafo utilizado no problema do caminho mínimo



FONTE: adaptado de Geeks for Geeks(2024).

A Tabela 8 mostra a porcentagem de vezes em que os algoritmos convergiram para o caminho ótimo. O algoritmo Q-SVR convergiu para caminho ótimo em 40% dos testes. Em 20% convergiu para a caminho (3, 5, 6, 7, 0), com recompensa de -28 e, em 10%, convergiu para (3, 5, 6, 8, 7, 0), com recompensa de -34. Em 10% convergiu para (3, 2, 5, 6, 7, 0) com recompensa de -25. Em 20% dos testes o mesmo convergiu para nenhum caminho onde, quando era testado, ficava em um loop entre nós e não chegava até o nó alvo. Apesar de tais testes, o algoritmo conseguiu aumentar a recompensa média ao longo do passar dos episódios, como mostra a Figura 12.

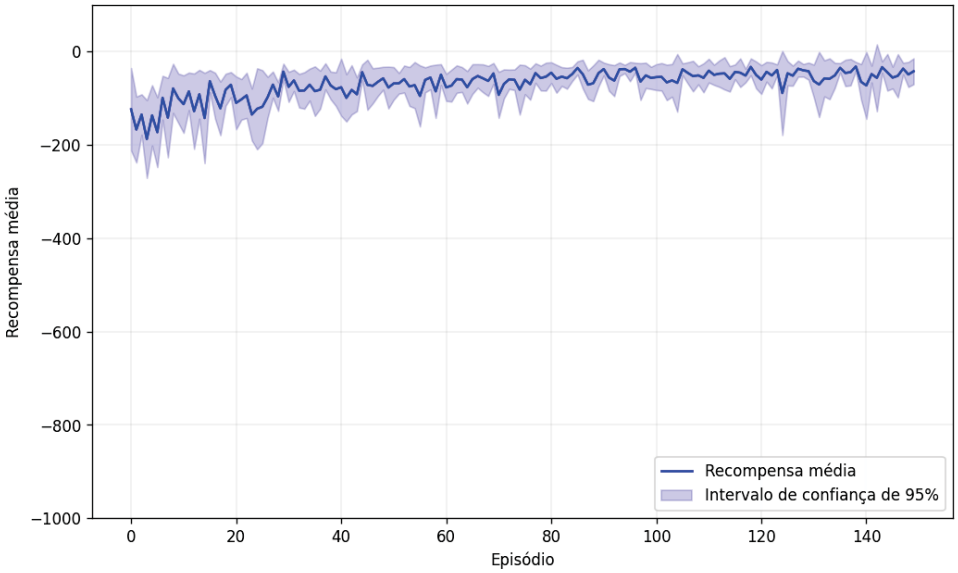
TABELA 8 – Porcentagem de testes que convergiram para a caminhos por algoritmo.

Recompensa do caminho	Q-SVR	Q-Learning	DQN
$r = -19$	40%	0%	30%
$r = -25$	10%	0%	10%
$r = -28$	20%	0%	30%
$r = -33$	0%	0%	10%
$r = -34$	10%	0%	0%
outras recompensas	20%	100%	20%

FONTE: O autor (2024).

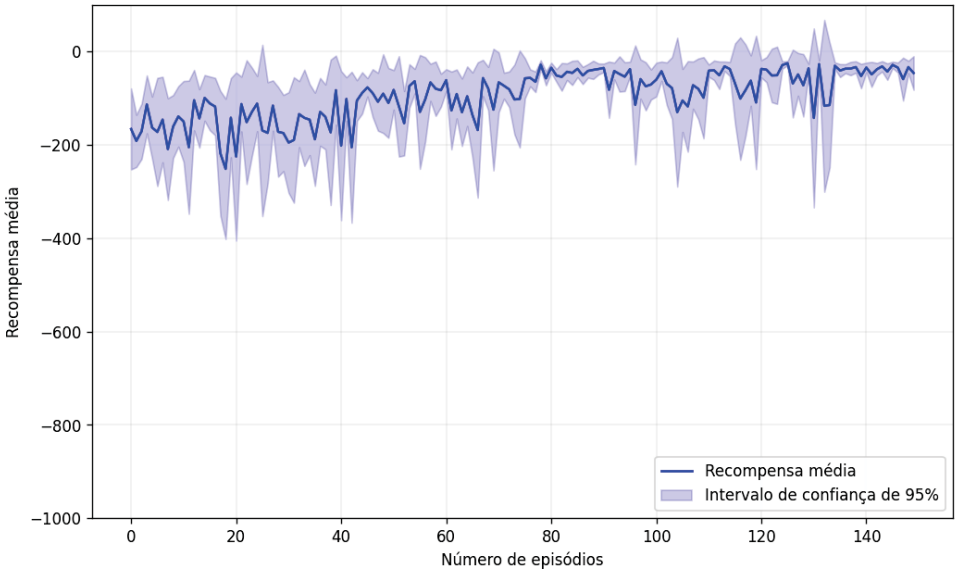
O DQN alcançou o caminho ótimo em 30% das vezes. Em 10%, chegou no caminho (3, 2, 8, 7, 0) com recompensa de -23, enquanto em 10% chegou em (3, 2, 5, 6, 7, 0) com recompensa de -25. Já em 30% chegou em (3, 5, 6, 7, 0) com recompensa de -28 e em 10% chegou em (3, 4, 5, 6, 7, 0) com recompensa de -33. Nos outros 10% dos testes convergiu à um caminho que não chegava no nó alvo. A Figura 13 ilustra o crescimento da recompensa média, caracterizando um aprendizado do ambiente.

FIGURA 12 – Recompensa média por episódio do algoritmo Q-SVR no ambiente do caminho mínimo.



FONTE: O autor (2024).

FIGURA 13 – Recompensa média por episódio do algoritmo DQN no ambiente do caminho mínimo.



FONTE: O autor (2024).

O Q-SVR também teve um resultado melhor que o DQN no que tange à quantidade de episódios até encontrar o caminho ótimo. A Tabela 9 mostra a mediana, o máximo e o mínimo de episódios a estabilização da política. O Q-SVR tem resultados melhores, cuja mediana é consideravelmente menor que o DQN. Seus valores mínimos e máximos também apresentam melhores resultados.

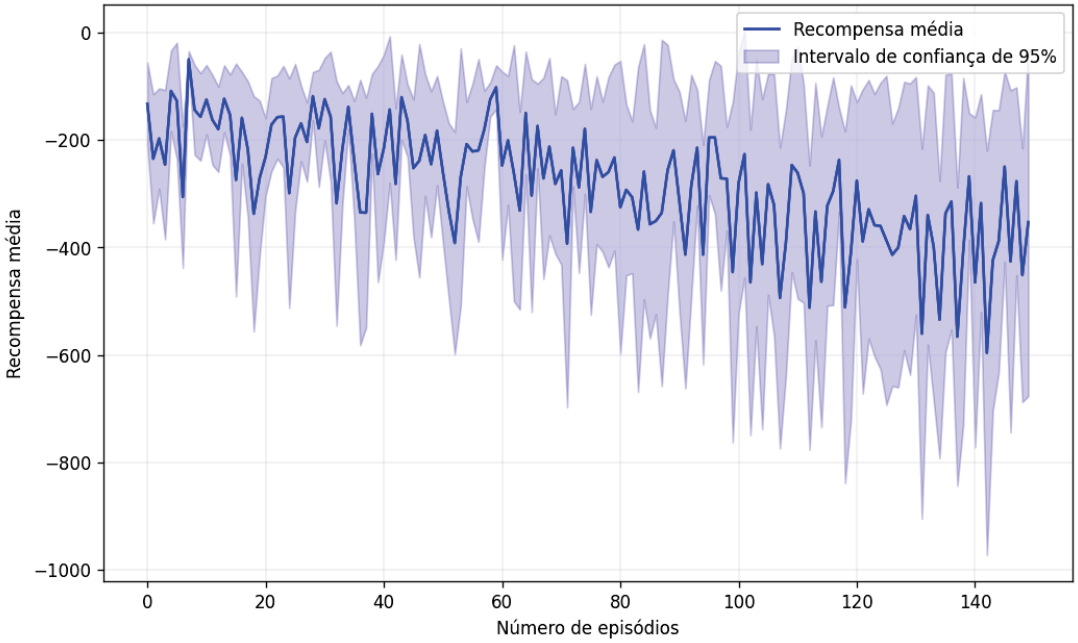
TABELA 9 – Medianas dos números de episódios até a estabilização na política

	mediana	máximo	mínimo
Q-SVR	24	75	6
DQN	142	149	132

FONTE: O autor (2024).

O algoritmo *Q-Learning* acabou por não convergir para o caminho ótimo em nenhum dos testes realizados. Como mostra a Figura 14, a recompensa média dos testes reduz com o passar dos episódios, o que caracteriza que o agente não está aprendendo a aumentar a recompensa no ambiente em que está.

FIGURA 14 – Recompensa média por episódio do algoritmo *Q-Learning* no ambiente do caminho mínimo.



FONTE: O autor (2024).

Apesar de não convergirem sempre para tal caminho, todos os testes de todos os algoritmos visitaram, em algum momento, o caminho ótimo. Isso mostra que a política ϵ-gulosa funcionou em explorar o ambiente.

No que tange ao desempenho, a Tabela 10 ilustra o tempo médio de execução dos algoritmos para 10 testes com a mesma semente aleatória. Dado que mudanças de implementação impactam no tempo de execução, a mesma não é a melhor métrica de desempenho, mas mostra uma característica do Q-SVR em relação aos outros algoritmos: um tempo de execução bem mais elevado. Logo, de maneira geral para o ambiente, uma quantidade maior de recompensa vem com o custo de um maior tempo de execução computacional.

TABELA 10 – Tempo de execução para uma semente aleatória.

	média (segundos)	desvio padrão (segundos)
Q-SVR	76,6149	0,1166
Q-Learning	0,2819	0,0083
DQN	4,3920	0,7641

FONTE: O autor (2024).

5 APLICAÇÃO NO JOGO DE CARTAS BLACKJACK

Neste capítulo será apresentada uma aplicação do algoritmo desenvolvido Q-SVR no jogo de cartas Blackjack. A primeira seção elucida as regras básicas do jogo, seguida de sua modelagem como um problema de aprendizado por reforço. A última seção do capítulo é dedicada à mostrar os resultados do modelo em relação à uma estratégia estática.

5.1 REGRAS BÁSICAS DO BLACKJACK

O jogo *Blackjack*, conhecido no Brasil também como 21, é um jogo de cartas onde um ou mais jogadores jogam, de forma independente, contra o *dealer* cujo objetivo é formar uma combinação de cartas que vença a combinação de cartas do *dealer*.

A cada rodada, o jogador aposta uma quantia e recebe duas cartas viradas para cima, enquanto o *dealer* recebe também duas cartas mas apenas uma virada para cima. O objetivo do jogador é ter a soma das duas cartas mais próxima de 21 do que o *dealer* mas sem exceder tal valor. O valor de cada carta é o seu número, enquanto as cartas Valete (J), Dama (Q) e Rei (K) valem 10. Já carta Ás pode valer 1 ou 11. Enquanto a soma das cartas não exceder 21, o jogador pode comprar (do inglês *hit*) mais cartas, adicionando o valor da mesma à sua soma. Outra ação disponível é a de parar (do inglês *stand*), onde o jogador não recebe mais cartas. Quando todos os jogadores terminam de tomar ações, o *dealer* revela sua carta que estava virada para baixo e joga seguindo uma regra estática: caso sua soma esteja igual ou abaixo de 16 o mesmo compra mais uma carta; caso esteja igual ou acima a 17, ele para.

A rodada termina ao se comparar as mãos de cada jogador com a do *dealer*: caso a soma da mão do *dealer* ultrapasse 21, todos os jogadores que parar vencem a rodada e recebem o dobro da aposta inicial. Caso a soma da mão do *dealer* não ultrapasse 21, vencem a rodada e recebem o dobro da aposta inicial os jogadores que possuem mãos com somas mais altas que o *dealer*. Caso o valor da mão do jogador seja igual à mão do *dealer*, tem-se um empate e o jogador recebe sua aposta de volta. Caso jogador tenha um Ás e uma das três cartas (Dama, Valete ou Rei) ele tem um Blackjack e ganha duas vezes e meia sua aposta original. Caso contrário, o jogador perde a rodada, assim como a aposta inicial (JUNIOR; OTTONI, 2020).

5.2 MODELAGEM COMO UM PROBLEMA DE APRENDIZADO POR REFORÇO

5.2.1 Espaço de estados e ações

O espaço de estados é um vetor s com três componentes. A primeira é a soma da mão atual do jogador, que podem inicialmente ir do valor 4 até o valor 12. A segunda é a soma das cartas do *dealer*, que inicialmente vão do valor 2 até 11. A última componente do vetor de estados é uma variável binária que diz se na mão atual há um Ás que pode ser utilizado como 1 ou 11. Quando há um Ás que pode ser utilizado, a mão é chamada de *soft* e, no caso contrário, é chamada de *hard*.

No que tange ao espaço de ações, as únicas ações possíveis são comprar uma carta ou parar, dado que o ambiente é uma versão simplificada do jogo.

5.2.2 Função de recompensa

A função de recompensa $r(s, a)$ é definida com os seguintes valores:

$$r(s,a) = \begin{cases} 1, & \text{se ganha o jogo} \\ 0, & \text{se empata o jogo} \\ -1, & \text{se perde o jogo} \end{cases}$$

Tal função, ao apenas levar em conta resultado do jogo, faz o agente ponderar eventos futuros e não apenas ao estado mais próximo.

5.3 RESULTADOS

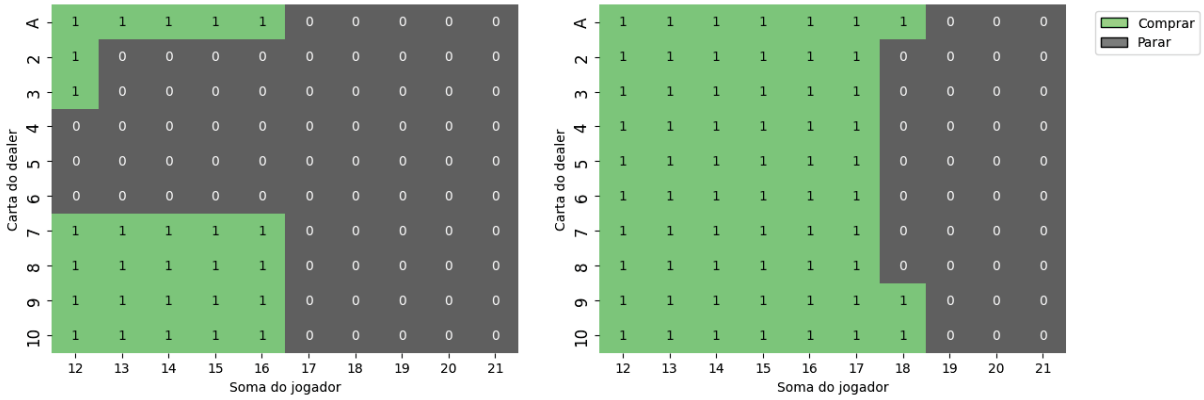
Para se comparar, foi utilizada uma estratégia estática apresentada no *website* especializado em apostas de jogos de cartas Wizard of odds (2024). As estratégias foram adaptadas para as regras do jogo utilizada e a Figura 15 exibe a estratégia para cada uma das somas do jogador e do dealer, sendo a estratégia à esquerda para o caso de mãos *hard* e à direita a estratégia no caso de mãos *soft*.

TABELA 11 – Valores numéricos dos parâmetros.

Parâmetro	Valor númerico
Número de episódios	2500
d	0.7
ε	0.1
C	1
ϵ_0	0.99
ϵ_d	0.999

FONTE: O autor (2024).

FIGURA 15 – Estratégia de referência para as mãos *hard* (à esquerda) e para as mãos soft (à direita).



FONTE: adaptado de Wizard of odds, 2024.

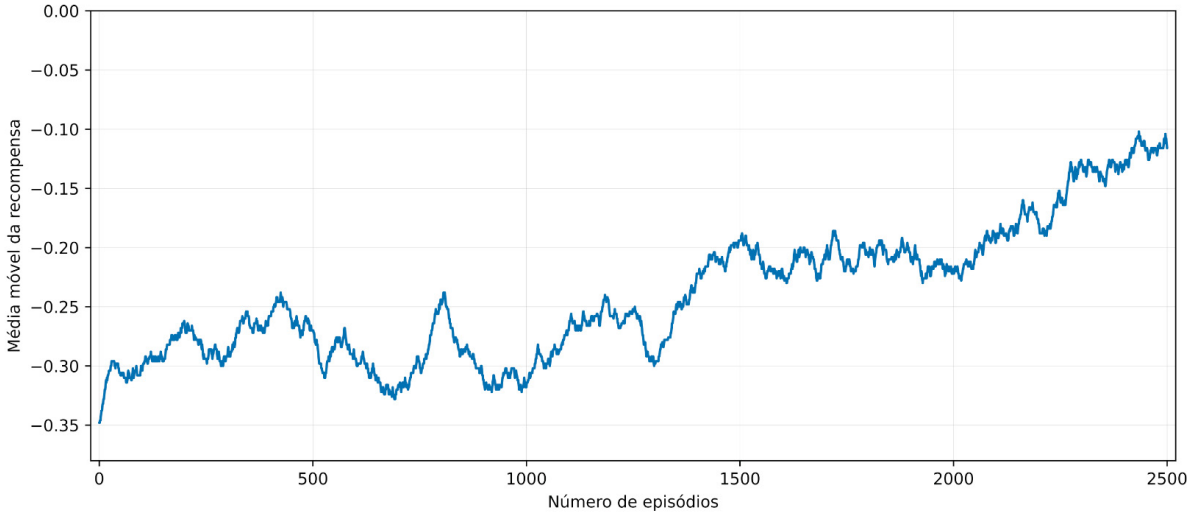
O agente com o Q-SVR treinou durante 2500 episódios. Tal número foi escolhido ao invés dos 150 usados nos experimentos do Capítulo 3 pois o agente toma poucas ações durante cada episódio. Para compensar, o número de episódios foi aumentado para que o agente tenha experimentação suficiente para conseguir entender o ambiente.

Para balancear o aproveitamento e exploração, foi utilizada a estratégia ϵ-gulosa apresentada na Seção 2.3. Com relação aos hiperparâmetros, foram utilizados os mesmos dos experimentos do Capítulo 4 e apresentados na Tabela 11. A única diferença é o parâmetro ϵ_d, que teve o seu valor aumentado para $0,999$ dado que número de episódios aumentou, o que garante mais episódios de exploração.

Para se analisar as recompensas, a média móvel é utilizada para suavizar a visualização das mesmas, uma vez que, por se tratar de valores que tem uma difícil visualização. A Figura 16 mostra a média móvel de 500 episódios da recompensa recebida durante o treinamento, onde é notório o crescimento da mesma com o passar dos episódios. Tal aspecto geralmente caracteriza o aprendizado do agente no ambiente.

A estratégia gerada pela modelo está presente na Figura 17 no caso de mãos *hard* e no caso de mãos *soft*. A política gerada para a mão *soft* é mais errante que a de mãos *hard*. Isso se deve ao fato que apenas aproximadamente 15% das mãos totais são mãos soft, o que dificulta o mapeamento da superfície dos valores ação-estado. A Figura 18 mostra os estado-valor estimados pelo Q-SVR para à direita onde é uma superfície mais irregular que para as mãos hard.

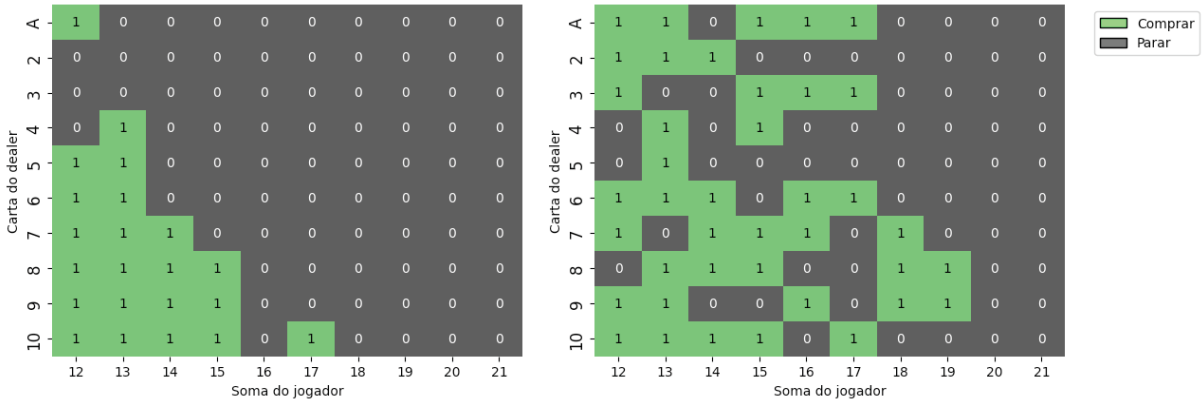
FIGURA 16 – Média móvel de 500 episódios da recompensa recebida.



FONTE: O autor (2024).

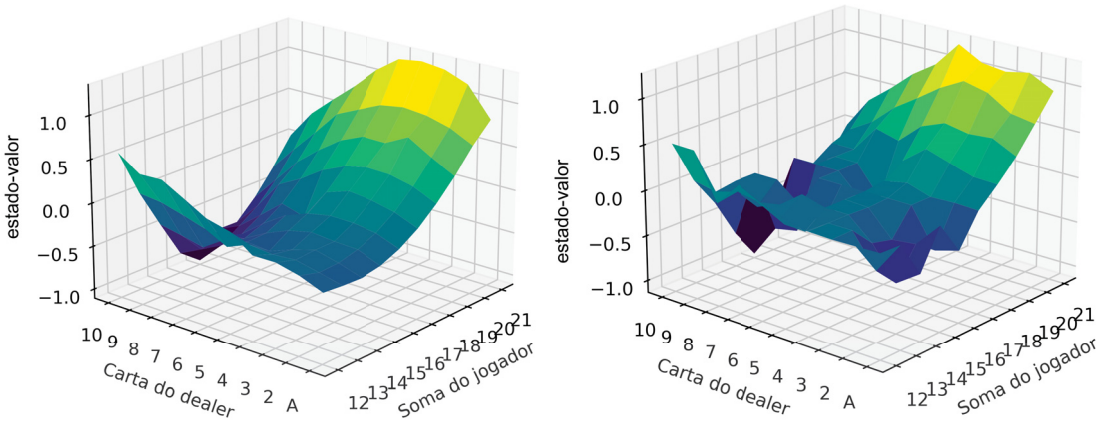
Para se comparar as duas estratégias, foram simulados 1000 jogos por 100 vezes e a estratégia do agente gerada pela sua política foi testada, assim como a estratégia de referência. A Tabela 12 ilustra a média de vitórias, empates e derrotas dos Q-SVR e da estratégia de referência. O número de empates permanece praticamente o mesmo, enquanto a diferença está no número de vitórias e no de derrotas. A estratégia com o agente usando Q-SVR tem um número médio de vitórias maior que o do agente com a estratégia de referência, assim como tem um número de derrotas menor. A Figura 19 compara visualmente as métricas da Tabela 12, onde é possível distinguir suas diferenças.

FIGURA 17 – Estratégia gerada pelo Q-SVR para as mãos *hard* (à esquerda) e para as mãos *soft* (à direita).



FONTE: O autor (2024).

FIGURA 18 – Estado-valor das mãos hard (à esquerda) e das mãos soft (à direita) gerado pelo Q-SVR.



FONTE: O autor (2024).

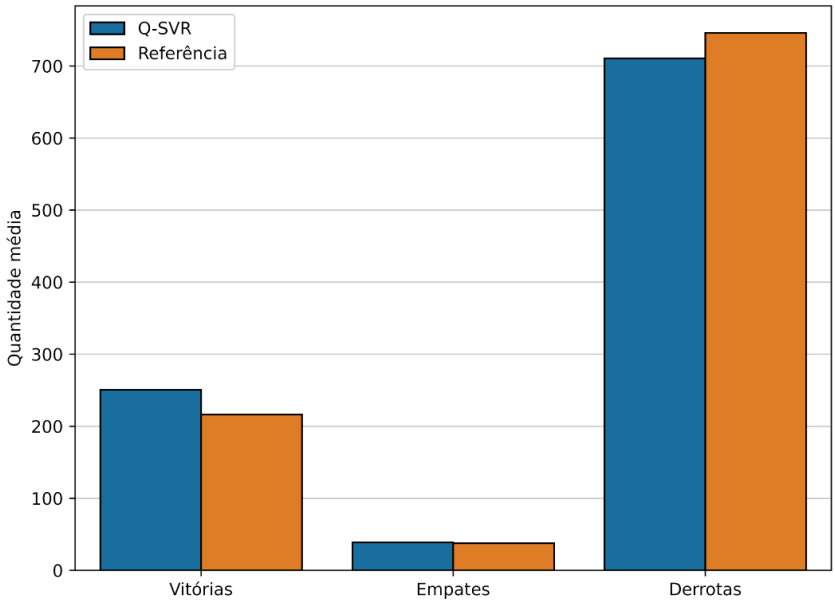
Tratando cada teste como uma amostra, um histograma pode ser feito para apresentar os resultados graficamente. A Figura 20 mostra a distribuição das vitórias dos dois modelos, onde é possível ver que há uma clara distinção entre as duas distribuições, apesar da mesma ser de pequena magnitude.

TABELA 12 – Quantidade média de jogos vencidos.

	Vitórias	Empates	Derrotas
Q-SVR	249,34	39,10	711,56
Referência	218,86	37,05	744,09

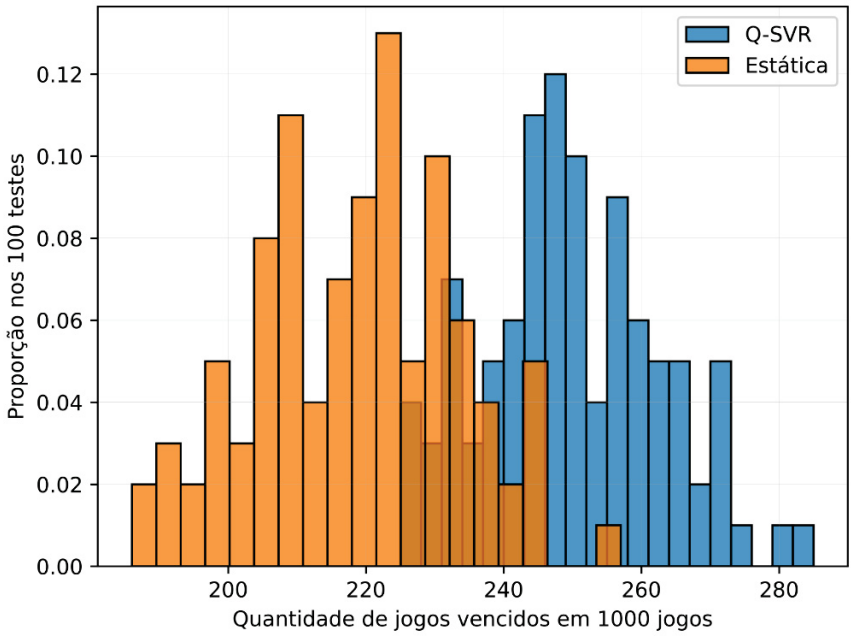
FONTE: O autor (2024).

FIGURA 19 – Comparação entre os números dos cenários possíveis.



FONTE: O autor (2024).

FIGURA 20 – Proporção do número de vitórias das duas estratégias.



FONTE: O autor (2024).

Dado os dados apresentados, é possível afirmar que os testes apresentados apontam que o Q-SVR conseguiu performar melhor que a estrategia de referência apesar de ser pequena a diferença. Mais testes são necessários para testar as afirmações, assim como estudos mais aprofundados dos hiperparâmetros para diminuir o efeito que a escolha de estratégias de referência e de hiperparâmetros arbitrários trazem nos resultados.

6 COMENTÁRIOS FINAIS

Este trabalho dedicou-se ao estudo do modelo proposto que recebeu o nome de Q-SVR, que utiliza o algoritmo AOSVR para resolver o problema do aprendizado por reforço. O estudo limitou-se à classe de problemas de aprendizado por reforço que tem um número pequeno de estados discretos e de ações, embora não houvesse nenhuma limitação teórica para a aplicação em ambientes com estados contínuos.

Testes foram realizados para testar a viabilidade do algoritmo proposto e resultados quantitativos dos mesmos foram analisados. Para fins de comparação, foram utilizados dois algoritmos já bem estabelecidos na literatura: o Q-Learning e o Deep Q-Network (DQN).

O primeiro teste foi a caminhada aleatória, teste cujo algoritmo proposto teve um resultado satisfatório quando o ambiente tinha poucos estados embora sua performance se deteriorava rapidamente quando o número de estados aumentava. Tal teste indica um ponto de atenção, uma vez que pode ser uma fraqueza do método apresentado.

O segundo teste foi a caminhada em cadeia de quatro estados onde pode-se concluir com as métricas que o Q-SVR superou o *Q-Learning*, uma vez que tem a recompensa média relativamente igual ao mesmo mas a mediana, mínimo e máximo números de episódios até a convergência menores, caracterizando um resultado melhor.

Foi realizado, por último dos experimentos, um teste no ambiente que simulava o problema do caminho mínimo em um grafo ponderado. Nele, o Q-SVR teve um resultado levemente melhor que o DQN, enquanto o *Q-Learning* não conseguiu realizar a tarefa. Outra possível fraqueza do método foi apontada pelos resultados: o alto tempo que o algoritmo demorou para concluir seus cálculos.

Uma aplicação também foi realizada ao simular o jogo de cartas Blackjack. O Q-SVR foi testado contra uma estratégia estática e performou levemente melhor que a estratégia de referência, embora mais testes devem ser realizados para confirmar a sua superioridade.

Em resumo, conclui-se que o Q-SVR apresentou desempenho relevante nos ambientes testados. Os testes também apontaram para duas possíveis pontos de atenção do algoritmo: perder capacidade de generalização com números de estados maiores que os testados e a alta quantidade de tempo utilizada na computação do algoritmo.

Para trabalhos futuros, é indicado o estudo dos dois pontos de atenção apresentados. Para o estudo do primeiro ponto, pode-se analisar em que tipo de ambiente tal efeito acontece (se é um só acontece em ambientes com recompensas esparsas, por exemplo) e em qual quantidade de estados tal efeito acontece. Já para o segundo ponto uma análise mais profunda pode ser feita no AOSVR, visando buscar passos que reduzam a quantidade de cálculos feita e visando a possibilidade de parelelização de parte do algoritmo.

É necessário apontar que os resultados aqui apresentados dependem da escolha arbitrária dos hiperparâmetros e podem ter resultados diferentes quando forem modificados. Logo, em conjunto com as indicações acima, o aprofundamento no estudo dos hiperparâmetros se faz necessário. Isso pode ser feito utilizando otimização Bayesiana ou *grid search* nos parâmetros e testando outros tipo de função kernel.

REFERÊNCIAS

AN, Y. et al. Discrete space reinforcement learning algorithm based on support vector machine classification. **Pattern Recognition Letters**, v. 111, p. 30–35, 2018.

CAUWENBERGHS, G.; POGGIO, T. Incremental and Decremental Support Vector Machine Learning. **Advances in Neural Information Processing Systems**, v. 1, 2001.

CHRISTMANN, A.; STEINWART, I. **Support Vector Machines**. 1. ed. Nova Iorque: Springer New York, 2008.

GEIST, M.; PIETQUIN, O. Algorithmic Survey of Parametric Value Function Approximation. **IEEE Transactions on Neural Networks and Learning Systems**, v. 24, n. 6, p. 845–867, 2013.

HAMBLY, B.; XU, R.; YANG, H. Recent advances in reinforcement learning in finance. **Mathematical Finance**, v. 33, n. 3, p. 437–503, 2023.

JUNIOR, H. M. R.; OTTONI, A. L. C. Aplicação de Aprendizado por Reforço no Blackjack: Estudo de Caso de Estimação de Parâmetros. **Anais do Congresso Brasileiro de Automática 2020**, sbabra, Campinas, 2020.

KAELBLING, L. P.; LITTMAN, M. L.; MOORE, A. W. **Reinforcement Learning: A Survey**. [S.l.: s.n.], 1996. arXiv: `cs/9605103 [cs.AI]`. Disponível em: `<https://arxiv.org/abs/cs/9605103>`.

KRULIKOVSKI, E. H. M. **Análise teórica de máquinas de vetores suporte e aplicação a classificação de caracteres**. dezembro 2017. Dissertação de Mestrado – Universidade Federal do Paraná, Curitiba, PR, Brasil.

LEE, D.-H. et al. Online Support Vector Regression based value function approximation for Reinforcement Learning. **2009 IEEE International Symposium on Industrial Electronics**, p. 449–454, 2009.

LINS, R. A. S. **Aprendizagem Por Reforço Profundo: Uma Nova Perspectiva Sobre O Problema Dos K-servos**. 2020. Tese (Doutorado em Engenharia Elétrica e de Computação) – Universidade Federal do Rio Grande do Norte, Natal.

LONG JIHAO; HAN, J. Reinforcement Learning with Function Approximation: From Linear to Nonlinear. **Journal of Machine Learning**, v. 2, n. 3, p. 161–193, 2023.

MA, J.; THEILER, J.; PERKINS, S. Accurate On-line Support Vector Regression. **Neural Computation**, v. 15, n. 11, p. 2683–2703, 2003.

MARTIN, M. On-Line Support Vector Machine Regression. **Machine Learning: ECML 2002**, Berlim, p. 282–294, 2002.

MELO, F.; RIBEIRO, I. Q-Learning with Linear Function Approximation. **Lecture Notes in Computer Science**, v. 4539, p. 308–322, jun. 2007.

MNIH, V. et al. Playing Atari with Deep Reinforcement Learning. **arXiv**, v. 1312.5602, 2013.

NORVIG, P.; RUSSELL, S. **Inteligência artificial**. 3. ed. São Paulo: Elsevier Brasil, 2014.

PARRELLA, F. Online Support Vector Regression, 2007. Disponível em: <https://api.semanticscholar.org/CorpusID:61599691>.

SANTOS, I. B. d. A. **Aprendizado por reforço profundo para navegação visual semântica com memória**. 2020. Dissertação (Mestrado em Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos.

SILVER, D. et al. Mastering the game of Go with deep neural networks and tree search. **Nature**, Nature Publishing Group, v. 529, n. 7587, p. 484–489, 2016.

SUTTON, R. S.; BARTO, A. G. **Reinforcement Learning: An Introduction**. 2. ed. Londres: The MIT Press, 2020.

VAPNIK, V. N. **The nature of statistical learning theory**. 2. ed. Nova Iorque: Springer-Verlag New York, 1995.

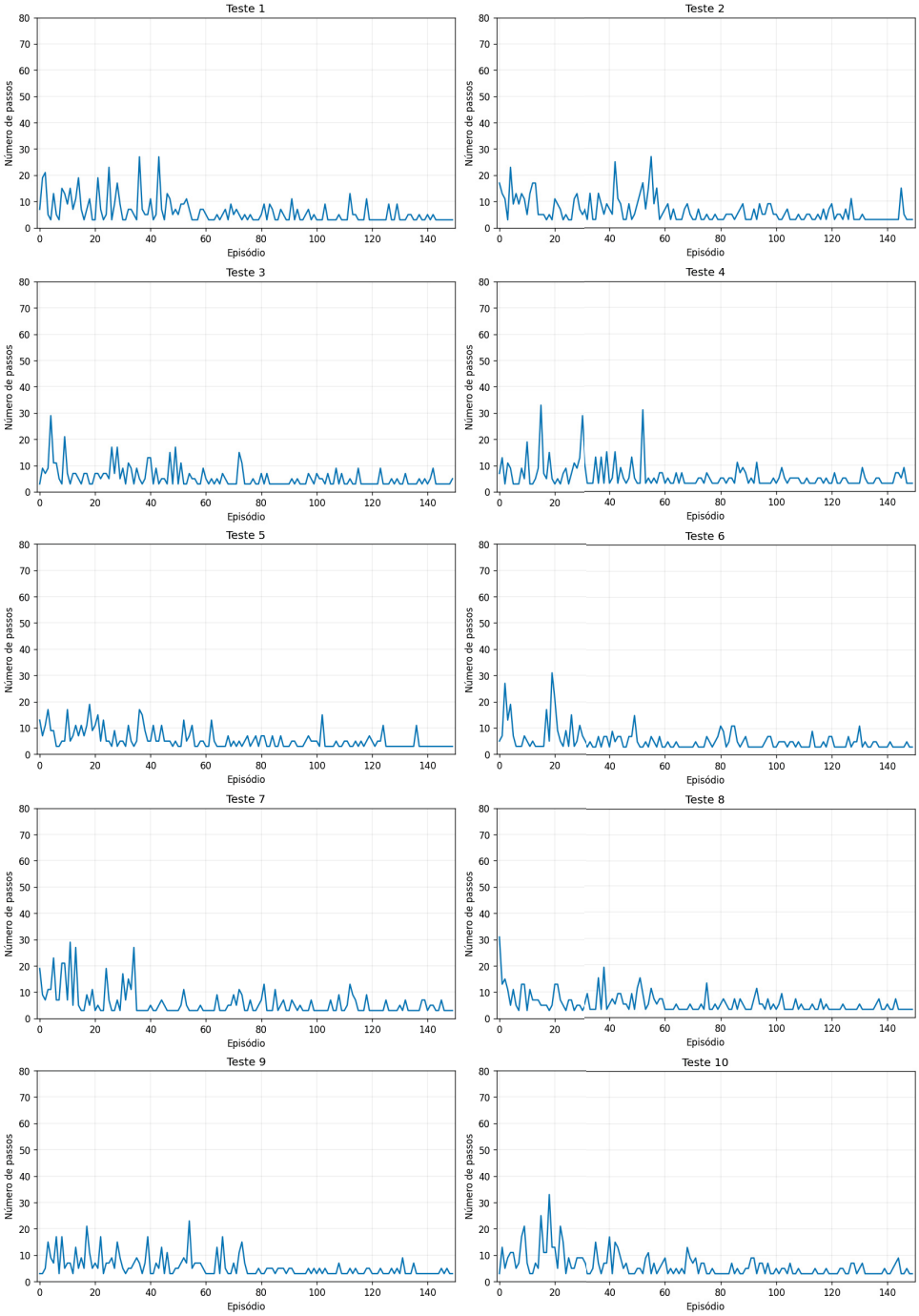
WATKINS, C.; DAYAN, P. Technical Note: Q-Learning. **Machine Learning**, v. 8, 1992.

YU, C. et al. Reinforcement Learning in Healthcare: A Survey. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 55, n. 1, nov. 2021. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3477600>.

ANEXOS

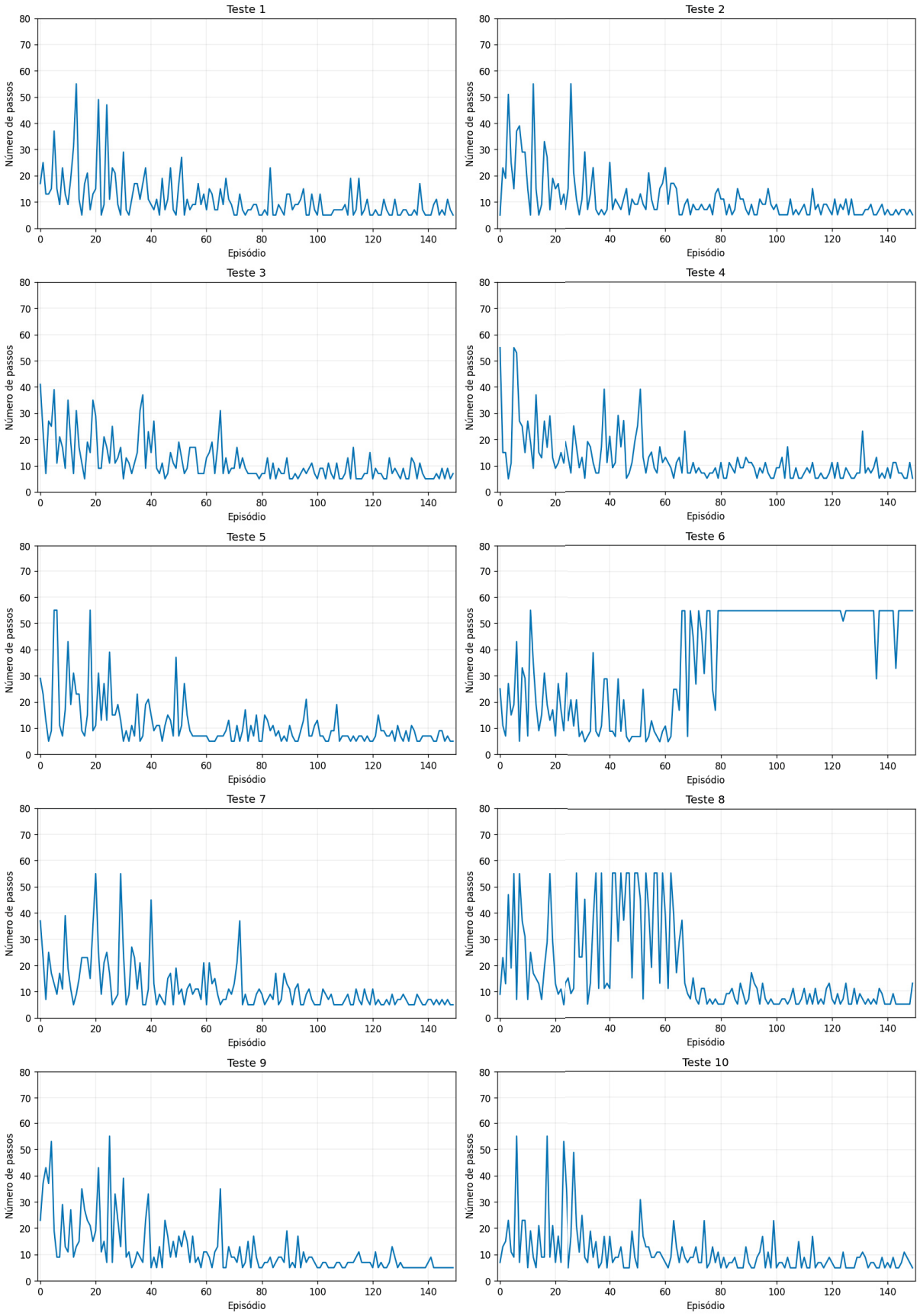
ANEXO A – NÚMERO DE PASSOS EM CADA UM DOS TESTES NO AMBIENTE CAMINHADA ALEATÓRIA

FIGURA 21 – Número de passos dos testes do algoritmo DQN no ambiente caminhada aleatória com 7 estados



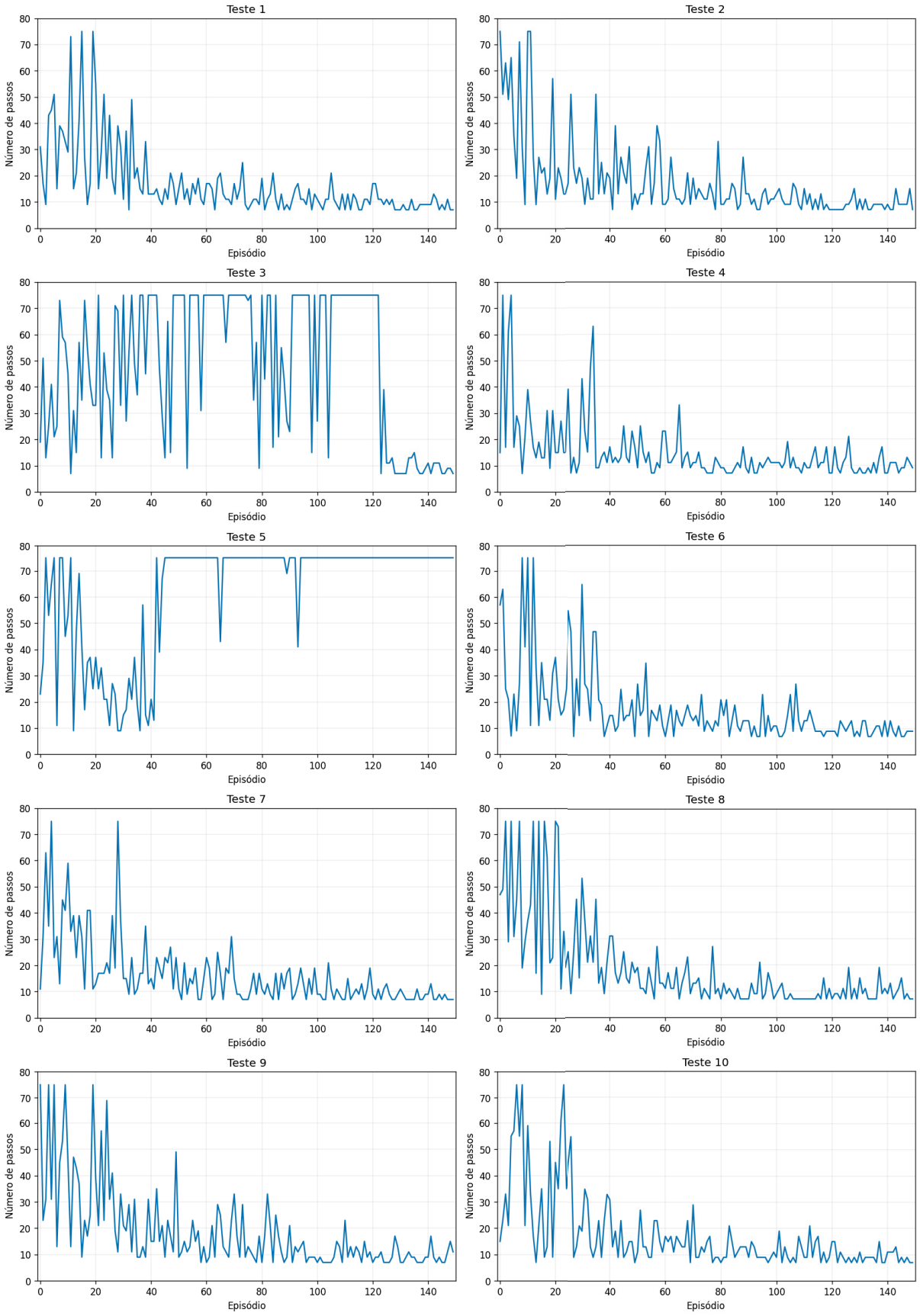
FONTE: O autor (2024).

FIGURA 22 – Número de passos dos testes do algoritmo DQN no ambiente caminhada aleatória com 11 estados



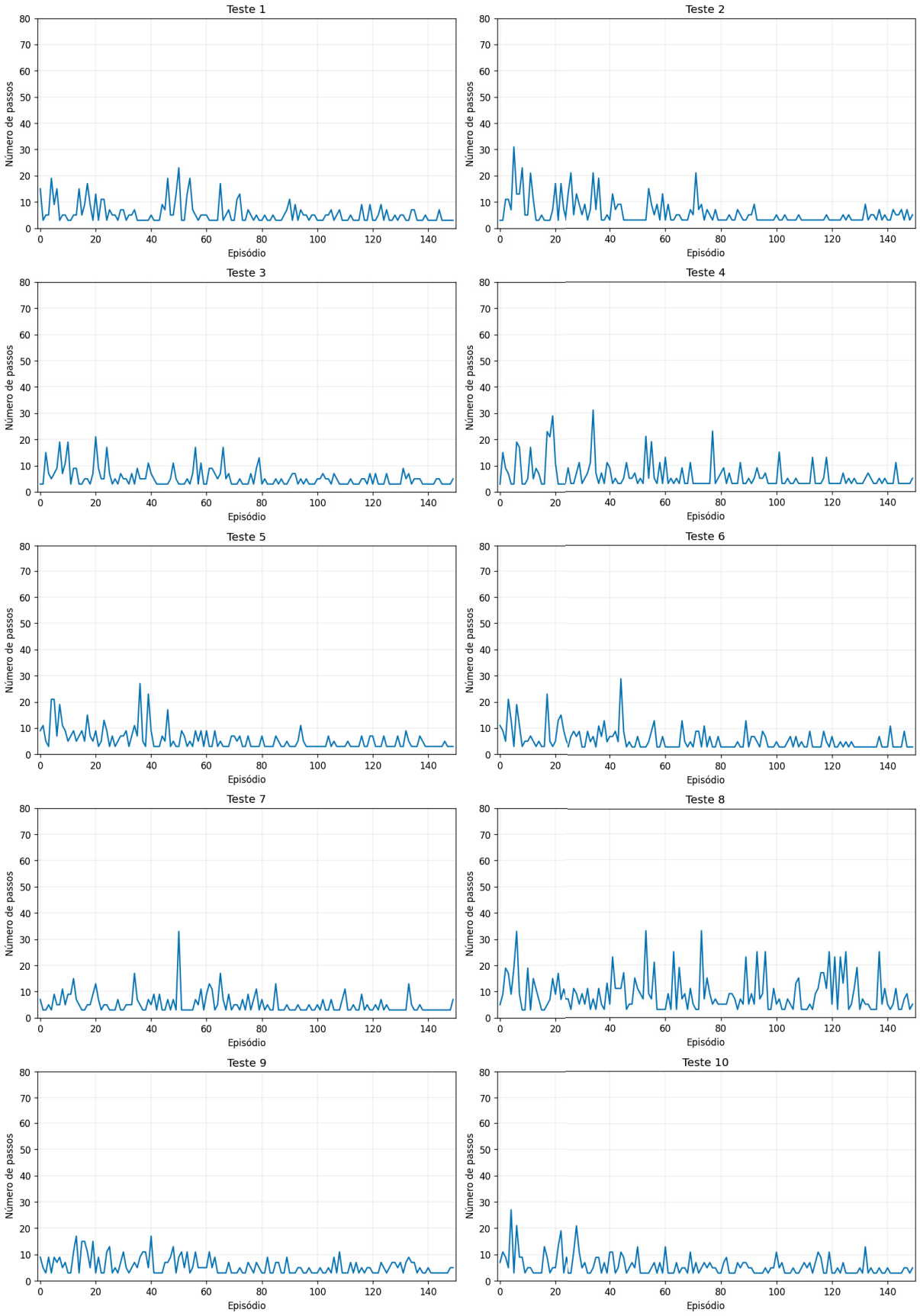
FONTE: O autor (2024).

FIGURA 23 – Número de passos dos testes do algoritmo DQN no ambiente caminhada aleatória com 15 estados



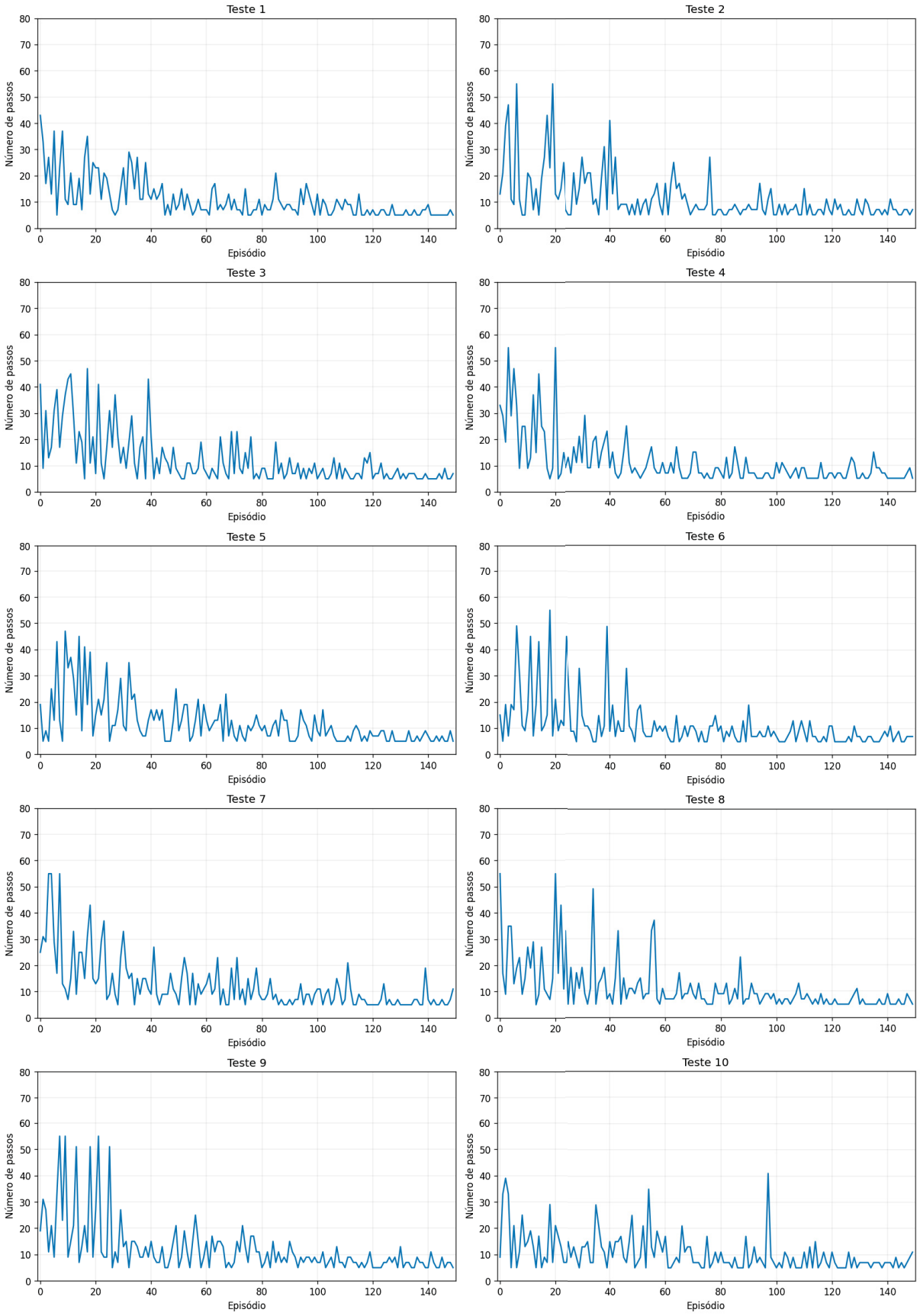
FONTE: O autor (2024).

FIGURA 24 – Número de passos dos testes do algoritmo Q-Learning no ambiente caminhada aleatória com 7 estados



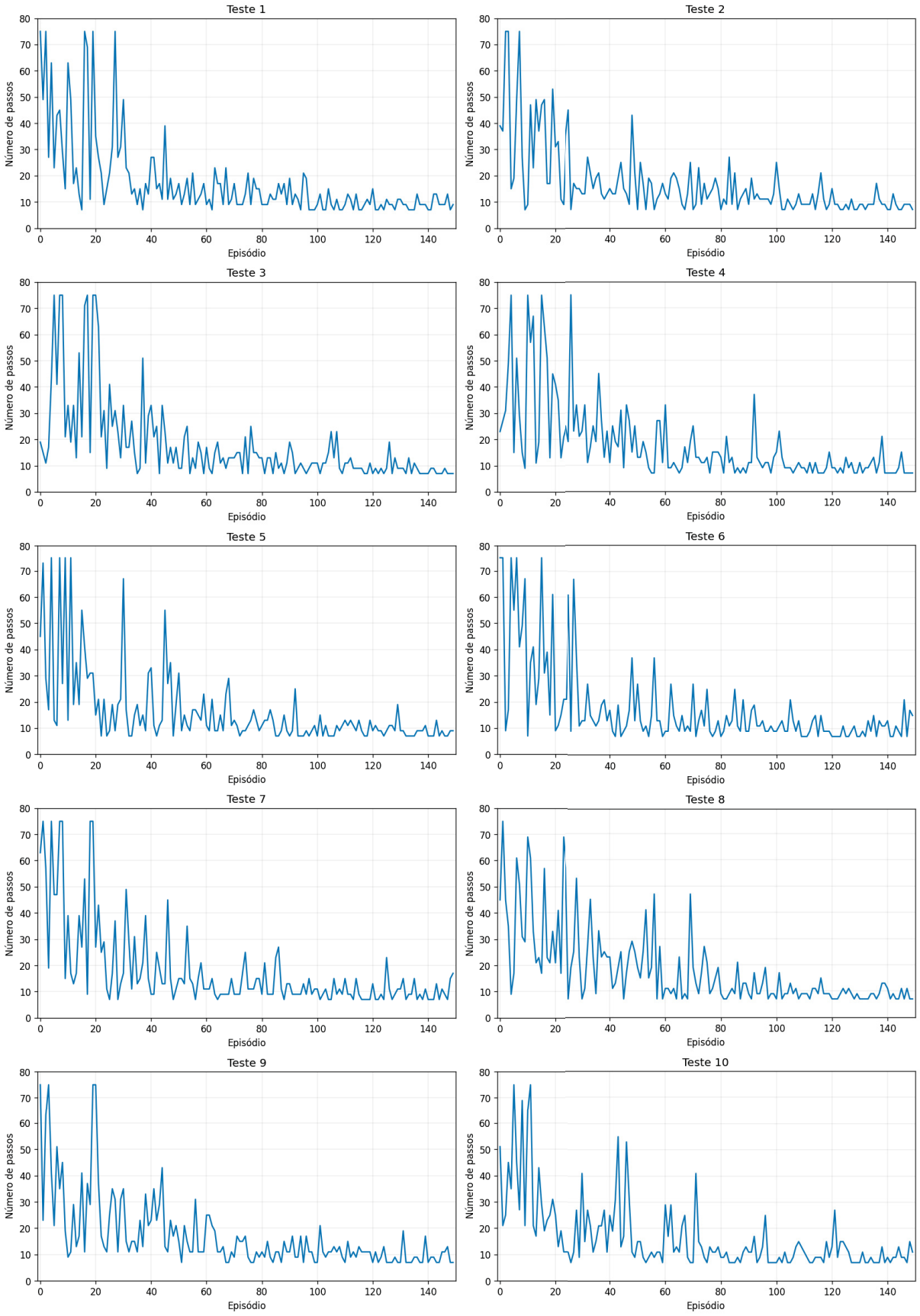
FONTE: O autor (2024).

FIGURA 25 – Número de passos dos testes do algoritmo Q-Learning no ambiente caminhada aleatória com 11 estados



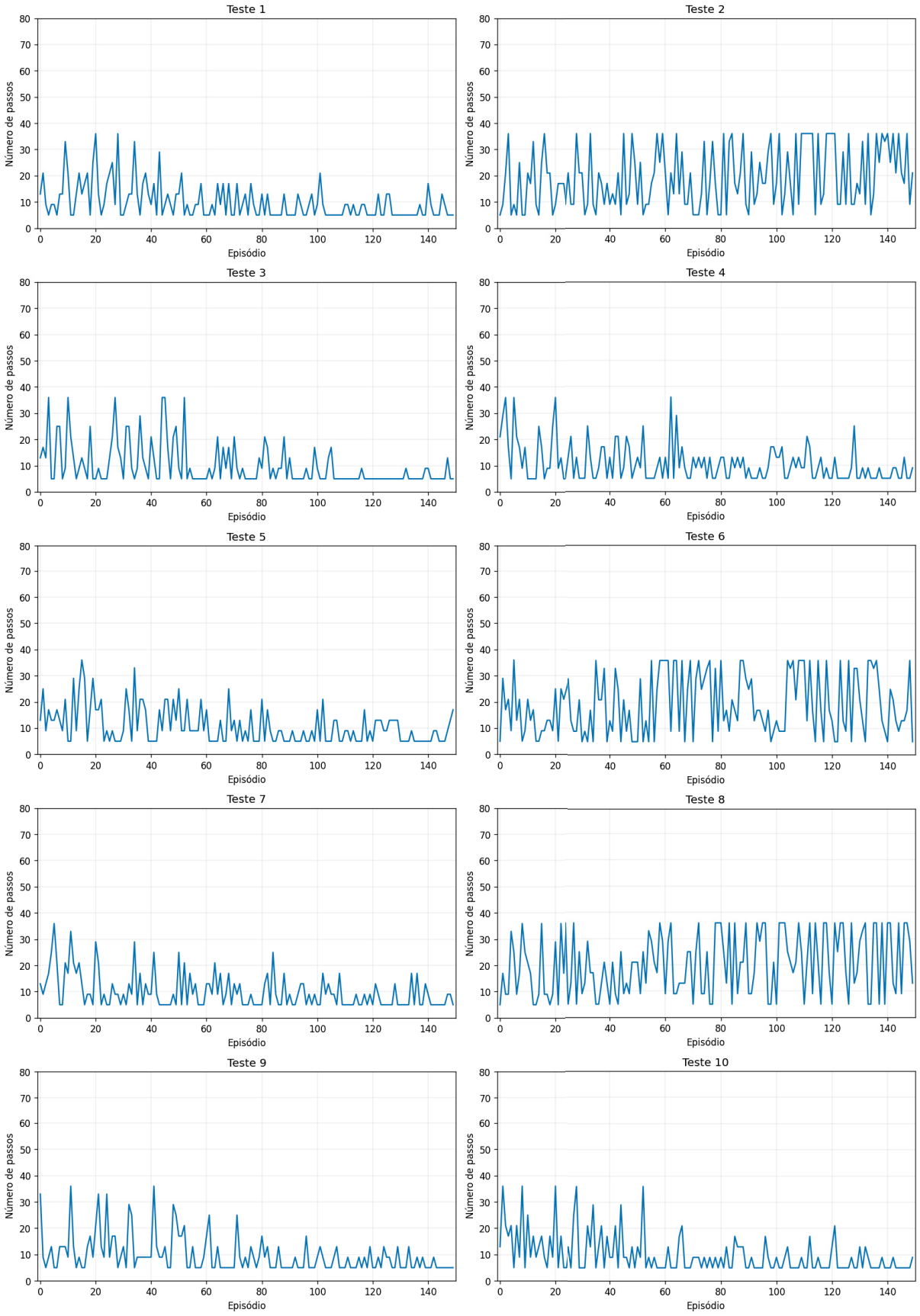
FONTE: O autor (2024).

FIGURA 26 – Número de passos dos testes do algoritmo Q-Learning no ambiente caminhada aleatória com 15 estados



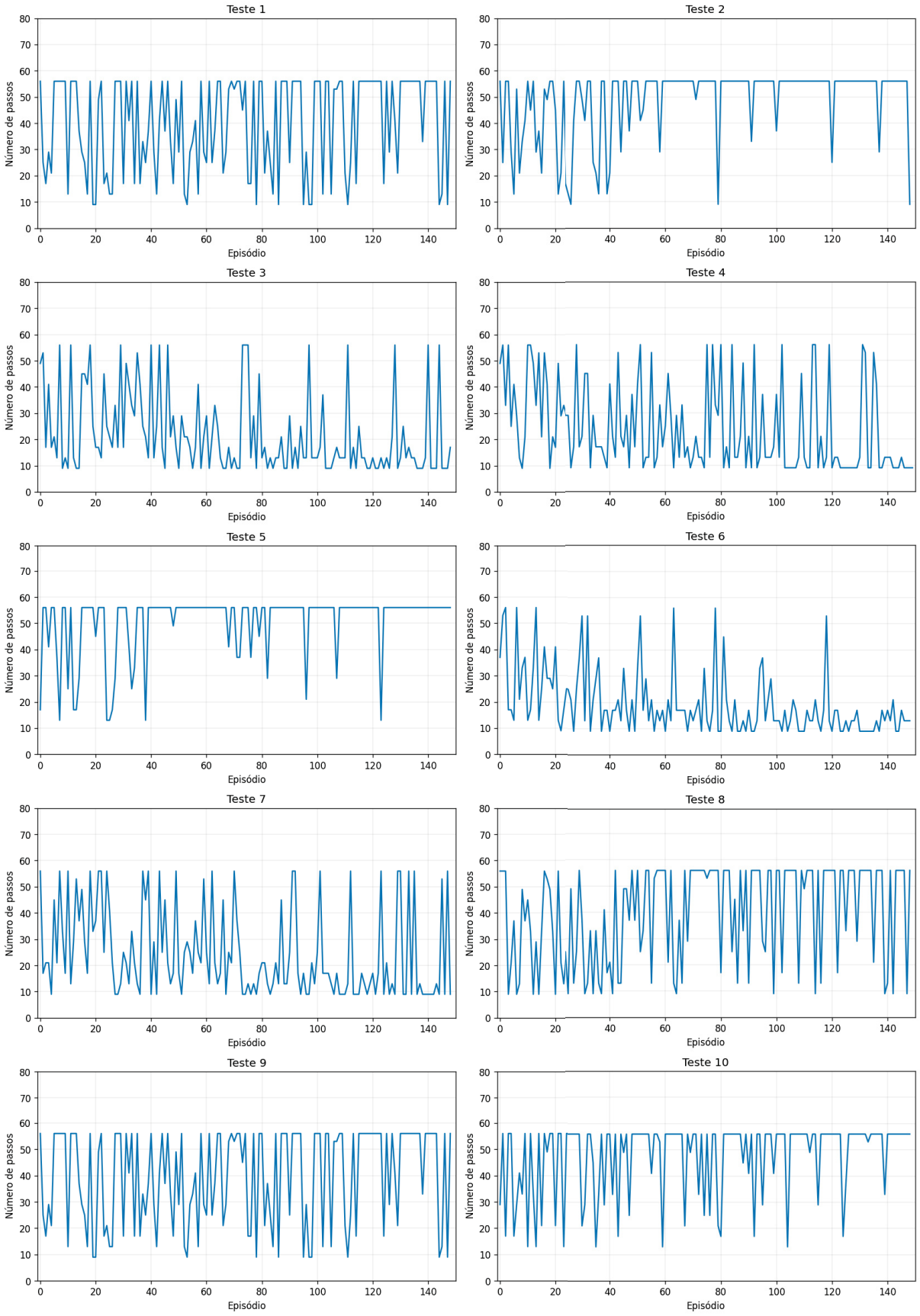
FONTE: O autor (2024).

FIGURA 27 – Número de passos dos testes do algoritmo Q-SVR no ambiente caminhada aleatória com 7 estados



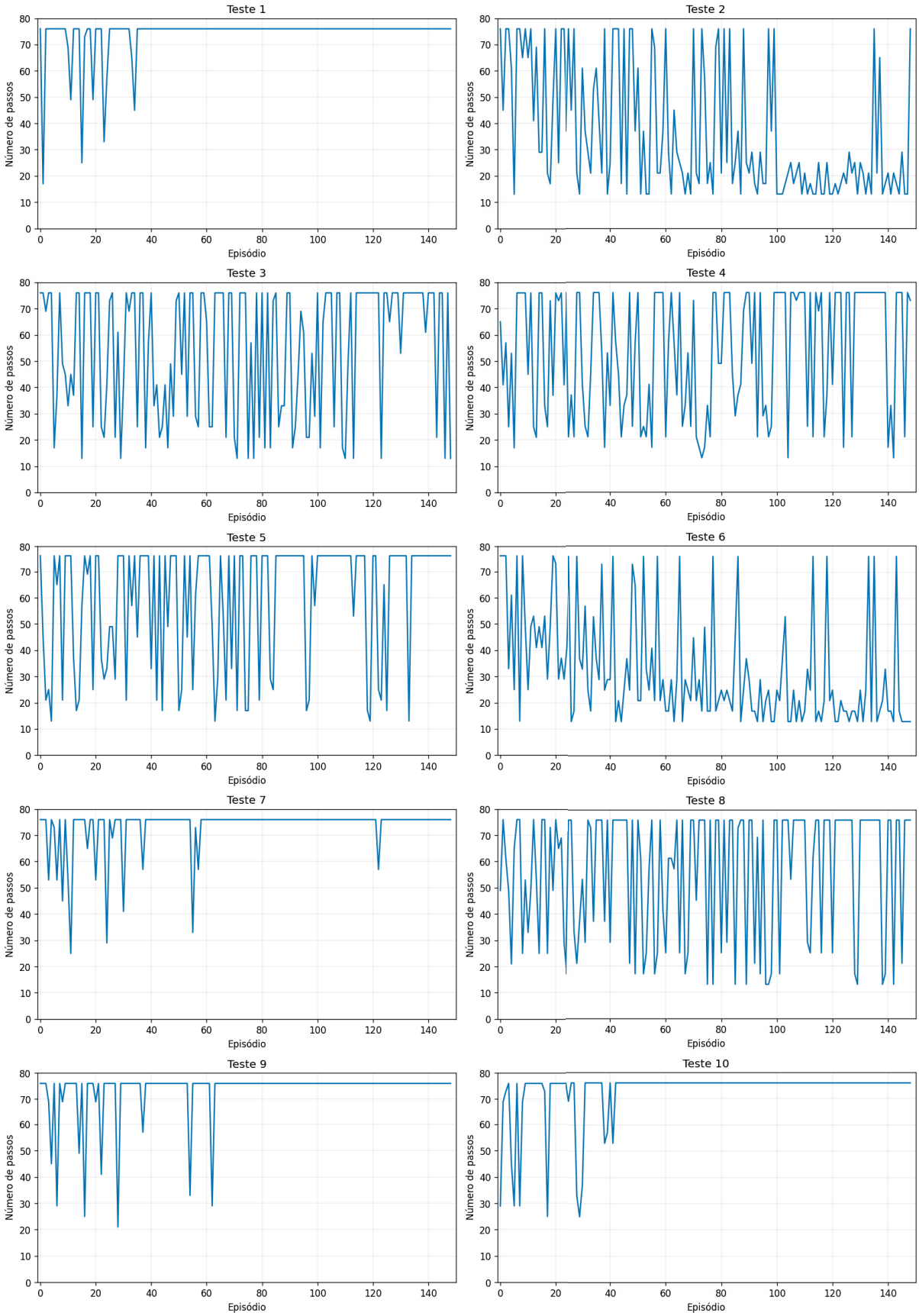
FONTE: O autor (2024).

FIGURA 28 – Número de passos dos testes do algoritmo Q-SVR no ambiente caminhada aleatória com 11 estados



FONTE: O autor (2024).

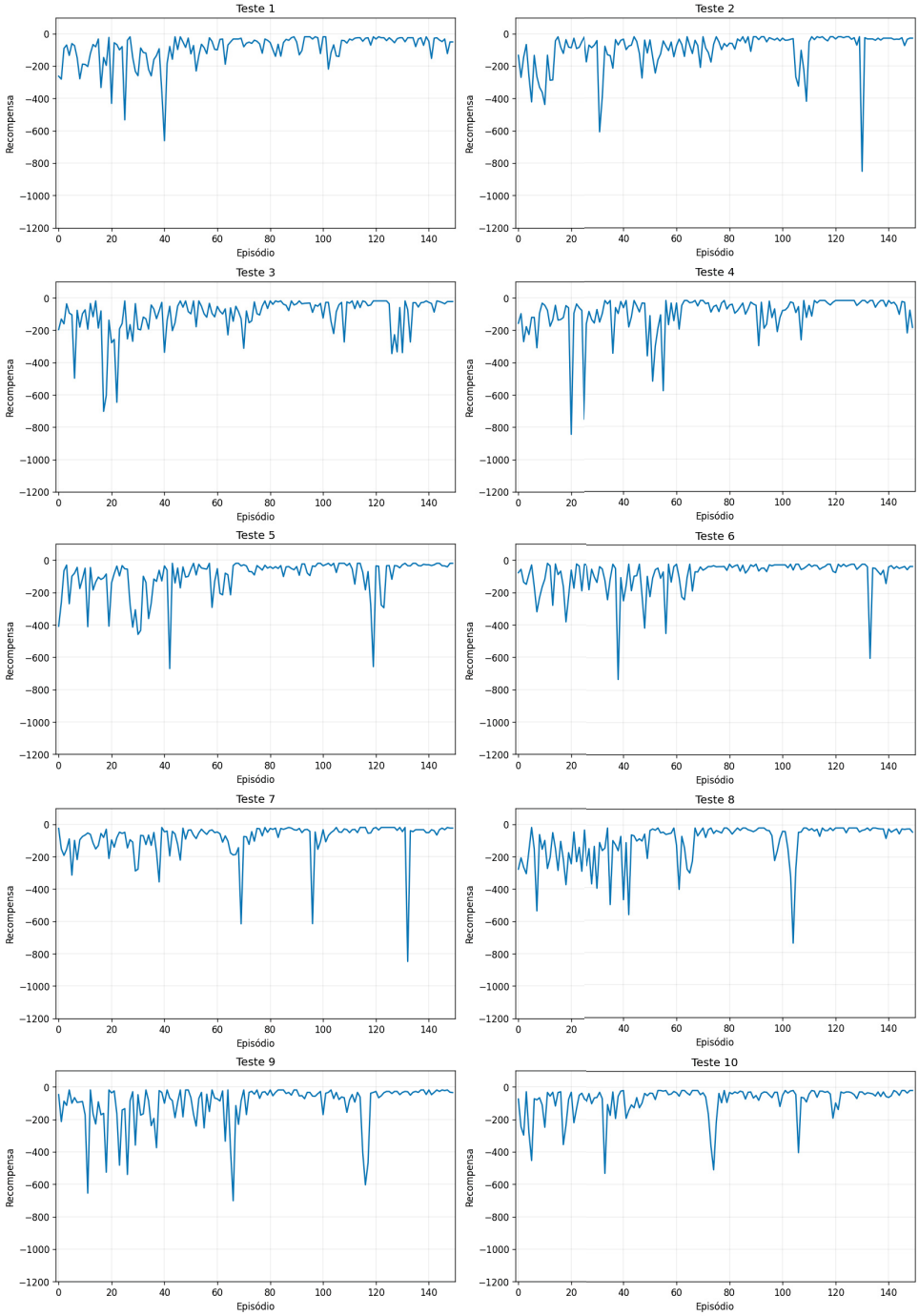
FIGURA 29 – Número de passos dos testes do algoritmo Q-SVR no ambiente caminhada aleatória com 15 estados



FONTE: O autor (2024).

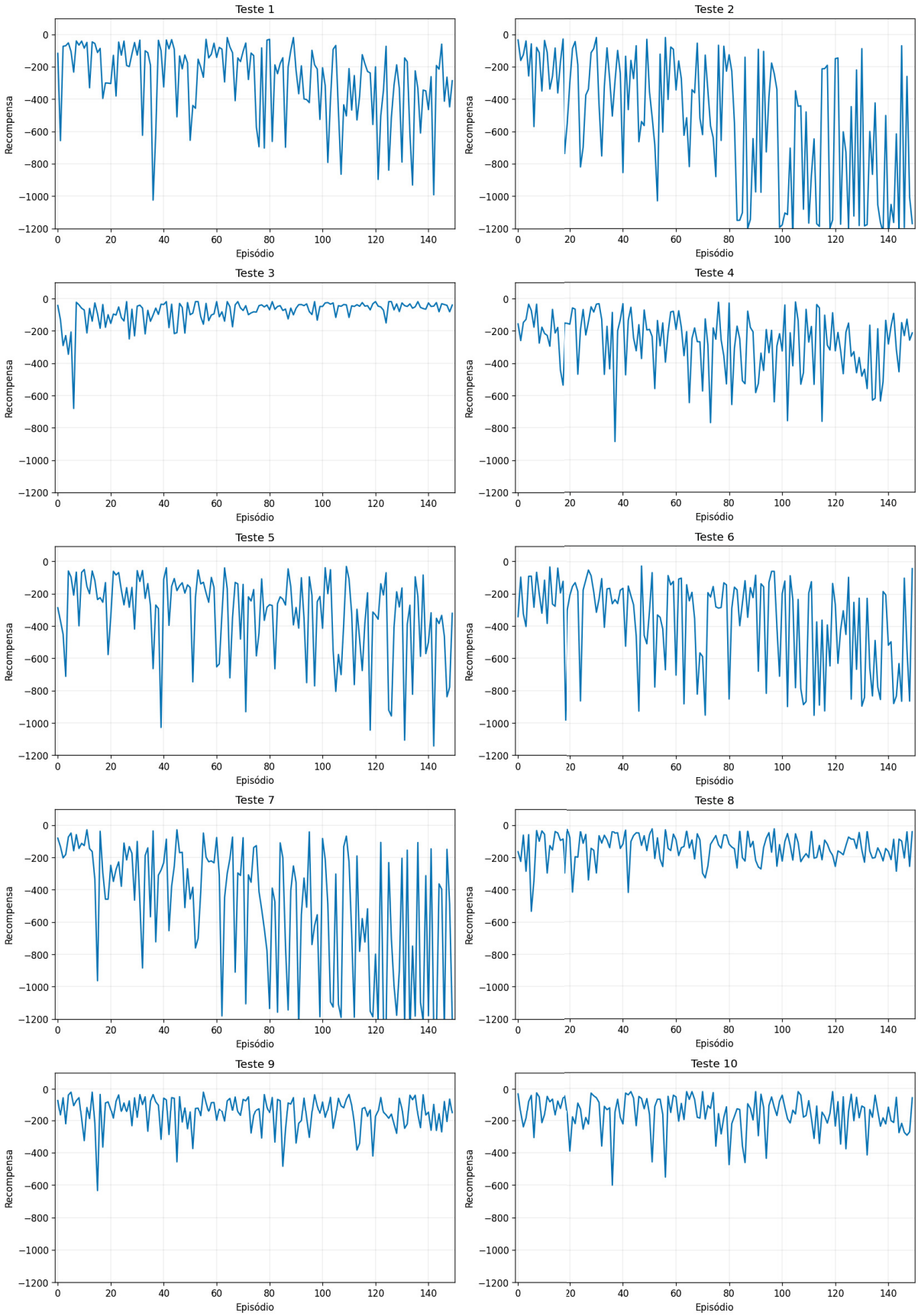
ANEXO B – RECOMPENSA EM CADA UM DOS TESTES NO AMBIENTE DO CAMINHO MÍNIMO

FIGURA 30 – Recompensa em cada teste por episódio do algoritmo DQN no ambiente do caminho mínimo



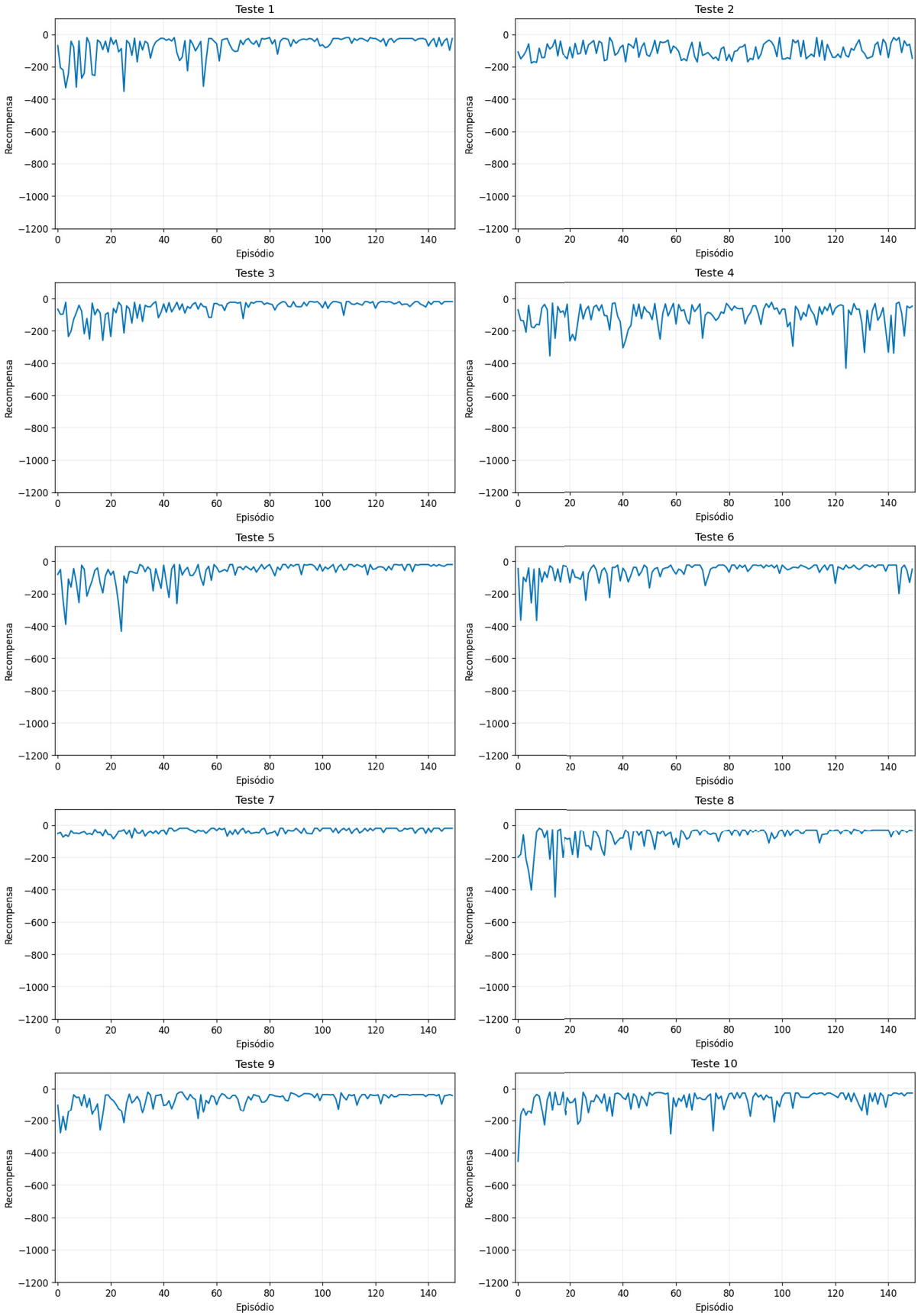
FONTE: O autor (2024).

FIGURA 31 – Recompensa em cada teste por episódio do algoritmo Q-Learning no ambiente do caminho mínimo



FONTE: O autor (2024).

FIGURA 32 – Recompensa em cada teste por episódio do algoritmo Q-SVR no ambiente do caminho mínimo



FONTE: O autor (2024).