UNIVERSIDADE FEDERAL DE MINAS GERAIS
Instituto de Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

Julio César Alves

Aplicação e Comparação de Métodos *Policy Gradient* em Problema de Cadeias de Suprimentos Multiestágio com Incertezas

Belo Horizonte
2021

Julio César Alves

Aplicação e Comparação de Métodos *Policy Gradient* em Problema de Cadeias de Suprimentos Multiestágio com Incertezas

Versão final

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais, como requisito parcial à obtenção do título de Doutor em Ciência da Computação.

Orientador: Geraldo Robson Mateus

Belo Horizonte

2021

Alves Julio César.

A474a Aplicação e comparação de métodos policy gradient em problema de cadeias de suprimentos multiestágio com incertezas [manuscrito] / Julio César Alves — 2021.
xxvi,166 f.

Orientador: Geraldo Robson Mateus.
Tese (Doutorado) - Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação.
Referências: f.147-152.;

1. Computação – Teses. 2. . Markov, Processos de – Teses. 3. Aprendizado por reforço – Teses. 4. Aprendizado profundo – Teses. I. Mateus, Geraldo Robson. II. Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação. III Título.

DU 519.6*82(043)

Ficha catalográfica elaborada pela bibliotecária Belkiz Inez Rezende Costa
CRB 6ª Região nº 1510

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Aplicação e Comparação de Métodos Policy Gradient em Problema de Cadeias de Suprimentos Multi-estágio com Incertezas

JULIO CÉSAR ALVES

Tese defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. GERALDO ROBSON MATEUS - Orientador
Departamento de Ciência da Computação - UFMG

PROF. ANDRÉ CARLOS PONCE DE LEON FERREIRA DE CARVALHO
Instituto de Ciências Matemáticas e de Computação - USP

PROF. ADRIANO ALONSO VELOSO
Departamento de Ciência da Computação - UFMG

PROF. CRISTIANO ARBEX VALLE
Departamento de Ciência da Computação - UFMG

PROF. DILSON LUCAS PEREIRA
Departamento de Ciência da Computação - UFLA

Belo Horizonte, 6 de Outubro de 2021.

Dedico este trabalho à minha esposa Fernanda e ao meu filho Vinícius por termos passado juntos as alegrias e os desafios desta jornada.

Agradecimentos

Agradeço primeiramente a Deus por ter proporcionado as condições para que esta conquista pudesse ser alcançada. Agradeço especialmente à minha esposa Fernanda por termos vencido juntos essa jornada! Passamos por muitos momentos, cada um com seus desafios e alegrias, desde mudança para BH, volta para Lavras, viagens e distância, pandemia, e, especialmente, a chegada do nosso filho Vinícius. Agradeço ao nosso filho por ter nos inspirado e tornado essa aventura mais feliz e mais completa. Amo vocês!

Agradeço aos meus pais por sempre terem sido exemplos de vida e de trabalho e por terem me incentivado nos estudos desde minha infância. Agradeço também aos meus irmãos, meus sogros, cunhados e sobrinhos por todo apoio, companheirismo e orações. E agradeço aos demais familiares e amigos pelas orações e torcida.

Agradeço ao meu orientador prof. Geraldo Robson por todo o aprendizado e parceria. Espero ter adquirido pelo menos um pouquinho de toda sua experiência e que eu consiga repetir com meus futuros orientados a sua forma de trabalho, sempre franca, tranquila e amistosa. Agradeço aos colegas do LaPO pelas trocas de experiências, pelos trabalhos conjuntos nas disciplinas e pelas conversas no laboratório, no café e no WhatsApp. Agradeço ao meu amigo Diego por reeditarmos agora no doutorado uma parceria que já havia acontecido antes na empresa que trabalhamos juntos e no mestrado.

Agradeço à UFLA e aos meus colegas de departamento pela possibilidade de fazer este doutorado em um período de afastamento para capacitação. Agradeço à UFMG, à UFLA, e a todas as instituições brasileiras que se dedicam à Ciência que é a base para a construção do progresso do nosso país. Por fim, agradeço ao contribuinte brasileiro por possibilitar, com seus impostos, que este doutorado se tornasse realidade. Espero, em retorno, contribuir com a sociedade brasileira sendo um bom professor e um bom pesquisador, ajudando na formação qualificada de futuros profissionais e pesquisadores, e também trabalhando em parcerias que transformem os frutos desta e de futuras pesquisas em progresso para a nossa sociedade.

Resumo

Algoritmos de Aprendizado por Reforço (AR) Profundo têm sido cada vez mais utilizados em diversas áreas do conhecimento e, recentemente, este interesse tem crescido também na comunidade de Otimização. Neste trabalho, aplicamos e comparamos algoritmos do tipo *Policy Gradient* em um problema clássico de otimização de planejamento de produção e distribuição de produtos em uma cadeia de suprimentos com múltiplos estágios. A maior parte dos trabalhos anteriores que utilizam métodos similares, considera somente cadeias de suprimentos seriais ou com até dois estágios, geralmente limitando as possibilidades de solução, e nenhum deles considera tempos de espera estocásticos. Nós consideramos uma cadeia com quatro estágios e dois nós por estágio, com incertezas nas demandas sazonais dos clientes finais e nos tempos de espera de produção nos fornecedores e de transporte ao longo da cadeia. Pelo nosso conhecimento, este trabalho é o primeiro a aplicar, nesta configuração de cadeia, métodos de AR Profundo, considerando uma abordagem centralizada para o problema, na qual todas as decisões são tomadas por um único agente, a partir das demandas incertas dos clientes finais. Propomos uma formulação de Processo de Decisão de Markov (PDM) e um modelo de Programação Linear (PL) com parâmetros incertos. A formulação PDM é adaptada de forma a se obter bons resultados com a aplicação dos algoritmos *Policy Gradient*. Em uma primeira fase, depois de um estudo de caso inicial, aplicamos o algoritmo Proximal Policy Optimization (PPO) em 17 cenários experimentais, considerando demandas incertas sazonais e regulares, com diferentes níveis de incerteza para as demandas, e tempos de espera constantes e estocásticos. Nesta fase, um agente construído a partir da solução de um modelo de Programação Linear (dado por considerarmos demandas esperadas e tempos de espera médios) é usado como *baseline*. Em uma segunda fase, comparamos cinco algoritmos, Advantage Actor-Critic (A2C), Deep Deterministic Policy Gradient (DDPG), PPO, Soft Actor-Critic (SAC) e Twin Delayed DDPG (TD3), em 8 dos 17 cenários anteriores, utilizando ferramentas estatísticas para comparação adequada dos algoritmos. Os algoritmos PPO e SAC alcançaram melhor desempenho nos experimentos realizados, sendo que o primeiro tem

um melhor tempo de execução. Os resultados experimentais indicam que métodos *Policy Gradient*, especialmente o PPO, são ferramentas adequadas e competitivas para o problema proposto. Em uma terceira fase, passamos a trabalhar com uma versão multiproduto do problema, generalizando a formulação PDM e o modelo PL com parâmetros incertos. Foram realizados experimentos com o algoritmo PPO em 16 cenários multiproduto, considerando dois e três produtos, e diferentes configurações de custos e de demandas. Os resultados encontrados indicam que, como no problema original, o PPO tem desempenho melhor que o baseline nos cenários com tempos de espera estocásticos.

Palavras-chave: Cadeias de suprimentos multiestágio, tomada de decisão sequencial sob incerteza, aprendizado por reforço, aprendizado profundo, métodos *Policy Gradient*.

Abstract

Deep Reinforcement Learning (DRL) methods have been increasingly used in several areas of knowledge and, recently, this interest has also grown in the Optimization community. In this work, we apply and compare Policy Gradient methods in the problem of planning the production and distribution of products in a supply chain with multiple stages. Most of the previous works that use similar methods only consider serial supply chains or only two echelons, generally limiting the solution possibilities, and none of them consider stochastic lead times. We consider a chain with four echelons and two nodes per echelon, with uncertainties regarding seasonal demands from customers and lead times of production at suppliers and transport along the chain. To our knowledge, this work is the first to apply, in such chain configuration, DRL methods considering a centralized approach to the problem, in which all decisions are taken by a single agent based on the uncertain demands of the end customers. We propose a Markovian Decision Process (MDP) formulation and a Linear Programming (LP) model with uncertain parameters. The MDP formulation is adapted to obtain good results with the application of Policy Gradient methods. In the first phase, after an initial case study, we applied the Proximal Policy Optimization (PPO) algorithm in 17 experimental scenarios, considering seasonal and regular uncertain demands (with different levels of uncertainty) and constant and stochastic lead times. In this phase, an agent built from the solution of a Linear Programming model (given by considering expected demands and average lead times) is used as a baseline. In the second phase, we have compared five algorithms, Advantage Actor-Critic (A2C), Deep Deterministic Policy Gradient (DDPG), Proximal Policy Optimization (PPO), SAC, and Twin Delayed DDPG (TD3), in 8 of the 17 previous scenarios, using statistical tools for proper comparison of the algorithms. The PPO and SAC algorithms had the best performance in the experiments, the first having a better execution time. Experimental results indicate that Policy Gradient methods, especially PPO, are suitable and competitive tools for the proposed problem. In the third phase, we started to work with a multi-product version of the problem, generalizing the MDP formulation and

the NLP model. Experiments were carried out with the PPO algorithm in 16 multi-product scenarios, considering two and three products and different cost and demand configurations. The results indicate that, as in the original problem, the PPO performs better than the baseline in scenarios with stochastic lead times.

Keywords: Multi-echelon supply chain, sequential decision-making under uncertainty, reinforcement learning, deep learning, Policy Gradient methods.

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Siglas

A2C Advantage Actor-Critic. 6, 8, 30, 61, 71, 72, 84–86, 122–124, 126–128, 141, 167, 168

A3C Asynchronous Advantage Actor-Critic. 36, 70, 71

AM Aprendizado de Máquina. 26, 48, 49, 86, 87, 89–91

AR Aprendizado por Reforço. 6, 16, 26–29, 31, 34, 35, 38–40, 42, 43, 48–51, 53, 55, 56, 60, 69, 70, 86, 87, 107, 118, 120–122, 124, 144, 169, 171, 180, 184, 185, 190

DDPG Deep Deterministic Policy Gradient. 6, 8, 30, 35–37, 61, 74, 76–81, 84–86, 122–124, 126–128, 141, 167, 168

DQN Deep Q-Network. 35, 36, 60

GAE Generalized Advantage Estimation. 69, 110, 132

GPI Generalized Policy Iteration. 55, 56, 58

MLP Multilayer Perceptron. 109, 126, 156

PDM Processo de Decisão de Markov. 6, 7, 26, 29, 31, 34, 39, 51–53, 100, 122, 143, 144, 163, 167, 168, 179

PL Programação Linear. 6, 7, 11, 12, 14–18, 29–31, 34, 37, 38, 44–46, 97, 101, 104, 105, 107, 109, 111–119, 121, 122, 143, 147–150, 153–156, 158–164, 167, 168, 171, 180–186

PO Pesquisa Operacional. 26–28

PPO Proximal Policy Optimization. 6–18, 29, 30, 36–39, 42, 43, 61, 72–74, 84–86, 96, 97, 99, 100, 102–124, 126–132, 135–139, 141, 142, 152, 154–164, 167–169, 171, 178, 180–188, 190

RNA Rede Neural Artificial. 10, 63, 64, 73, 76, 89, 93, 118, 120

RNAs Redes Neurais Artificiais. 10, 26, 51, 63, 71, 73, 74, 83, 86–88, 92–94, 108, 110, 126, 132, 133, 156, 157, 188

SAC Soft Actor-Critic. 6, 8, 13, 17, 30, 39, 61, 81–86, 122–124, 126–131, 133–137, 139, 141, 142, 152, 164, 167, 168

SB3 Stable Baselines. 108, 126

SGD Stochastic Gradient Descent. 19, 90–92

TD Temporal Difference. 57–59, 68, 69, 87

TD3 Twin Delayed DDPG. 6, 8, 30, 61, 78–82, 84–86, 122–124, 126–129, 141, 167, 168

TPE Tree-structured Parzen Estimator. 105, 106, 125, 130

epigraphtext

Sumário

Capítulo 1

Introdução

Nos dias atuais há um crescente interesse na aplicação de métodos de Aprendizado de Máquina (AM) em problemas de diversas áreas. Aprendizados Supervisionado e Não-Supervisionado tiveram um grande crescimento na última década devido a um conjunto de fatores. Dentre eles podemos destacar a crescente disponibilidade de dados, o renascimento das Redes Neurais Artificiais (RNAs) (com o desenvolvimento de técnicas que permitiram o uso de redes com várias camadas ocultas, o aprendizado profundo), e a evolução de *hardware* que possibilitou que se pudesse lidar com problemas cada vez maiores. AR é um outro ramo de AM que começou a ganhar mais atenção um pouco depois, principalmente pelos trabalhos que usaram este tipo de abordagem para resolver problemas de jogos em níveis sem precedentes, como jogos de Atari [Mnih et al., 2015] e Go [Silver et al., 2016].

O presente trabalho usa métodos de AR para resolver um problema de otimização de planejamento de produção e de distribuição produtos em uma cadeia de suprimentos de multiestágios [Pochet & Wolsey, 2006; Pinedo, 2009] com parâmetros incertos (demandas sazonais e tempos de espera). A ideia não é trabalhar na teoria subjacente às duas áreas (Otimização e AR), mas sim contribuir na forma de modelar e resolver problemas considerando as potencialidades dos recentes desenvolvimentos da área de AR. O problema abordado é geralmente tratado com ferramentas de Pesquisa Operacional (PO) e a ideia é verificar a adequação de algoritmos estado-da-arte de AR na solução do problema. Em problemas de PO do mundo real é muito comum a ocorrência de incerteza nos parâmetros de um modelo e, geralmente, tais incertezas são compensadas por margens de segurança e *buffers* flexíveis, o que gera excessos de capacidades e estoques não utilizados [Seelenmeyer & Kißler, 2020]. AM é uma abordagem alternativa para resolver essa questão, por aprender a partir de dados e/ou simulação. O problema tratado nesta pesquisa pode ser formulado como um PDM, e como o modelo resultante

não pode ser resolvido numericamente devido ao alto número de dimensões dos espaços de estados e ações [Laumanns & Woerner, 2017], AR Profundo é uma alternativa para solução do problema. Nesta pesquisa usamos métodos de AR, mais especificamente do tipo *Policy Gradient*, para resolver o problema tratado. Vale ressaltar que problemas de tomada de decisão sequencial da área de PO têm, em geral, muito mais decisões a serem tomadas em cada período (ou *time step*) do que aqueles geralmente tratados pela comunidade de AR. Mesmo assim, nesta pesquisa conseguimos aplicar com sucesso e comparar diferentes algoritmos estados-da-arte de AR para resolver o problema proposto.

No problema abordado nesta tese, consideramos o caso no qual as decisões de toda a cadeia são baseadas nas demandas dos clientes finais, e, portanto, há um único tomador de decisões centralizado, e os estágios da cadeia trabalham em regime de colaboração para minimizar os custos totais de operação. A cadeia de suprimentos considerada é uma cadeia de quatro estágios composta por dois fornecedores, duas fábricas, dois distribuidores e dois varejistas. Fornecedores produzem e fornecem matérias-primas que são processadas pelas fábricas para gerar produtos finais. Os produtos são distribuídos pelas fábricas para os distribuidores, e estes, por sua vez, enviam os produtos para os varejistas. Os varejistas são responsáveis por atender às demandas incertas sazonais dos clientes. Cada nó da cadeia tem um estoque local capacitado, fornecedores e fábricas armazenam matérias-primas, enquanto distribuidores e varejistas estocam produtos finais. Há tempos de espera (*lead times*) estocásticos para produzir matéria-prima nos fornecedores e para transportar material de um nó para outro da cadeia. Há ainda limites de capacidade referentes à produção dos fornecedores e ao processamento de matéria-prima nas fábricas. O objetivo é operar toda a cadeia, dado um horizonte de planejamento, para atender às demandas dos clientes e minimizar os custos totais de operação. Os custos são associados com a produção e o processamento de matérias-primas e com o estoque e transporte de matérias-primas e produtos. Há ainda um custo de penalização quando uma demanda de cliente não é atendida. Como as demandas e os tempos de espera são incertos, não é trivial definir a melhor política que possa atender às demandas sazonais dos clientes e, ao mesmo tempo, minimizar os custos totais de operação. A Figura 1.1 ilustra a cadeia de suprimentos tratada no presente trabalho. Uma descrição mais detalhada do problema é apresentada na Seção 2.1.

Alguns trabalhos na literatura usam AR Profundo em problemas relacionados, mas eles geralmente lidam com cadeias de suprimentos menores, com dois estágios, ou com cadeias seriais (ou seja, com um nó por estágio) [Oroojlooyjadid, 2019; Kemmer et al., 2018; Hutse, 2019; Gijsbrechts et al., 2019; Peng et al., 2019]. Considerando cadeias de suprimentos seriais, a dimensionalidade do espaço de ações cresce linearmente

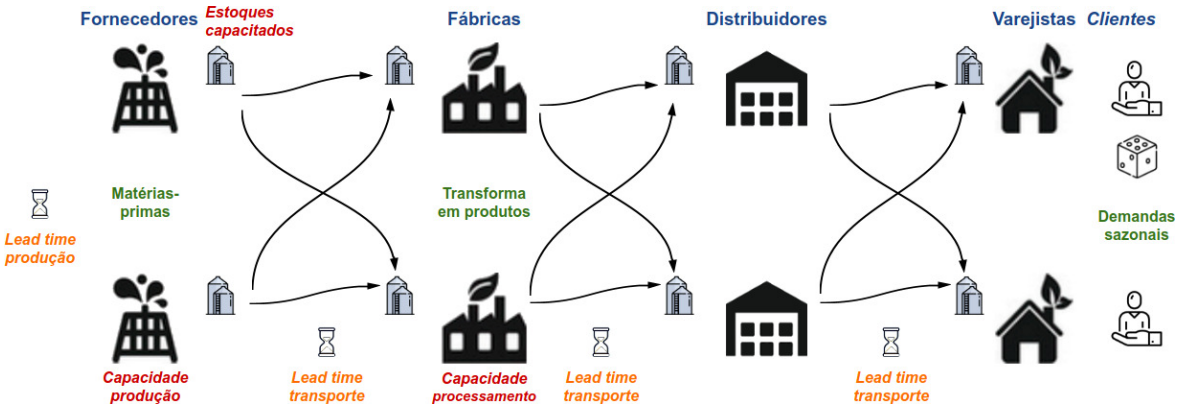

Figura 1.1. Cadeia de suprimentos tratada: há quatro estágios (fornecedores, fábricas, distribuidores e varejistas) com dois nós por estágio. Todos os nós têm estoques locais capacitados, e os fornecedores e fábricas também são capacitados. Há tempos de espera estocásticos para produzir matéria-prima nos fornecedores e para transportar material de um nó para outro. Há demandas incertas, potencialmente sazonais, de clientes a serem atendidas pelos varejistas.

com o número de estágios. Mas em relação às cadeias não-seriais, a dimensionalidade cresce a uma taxa maior, uma vez que cada nó precisa enviar material para mais de um nó do próximo estágio. Uma cadeia de suprimentos serial de quatro estágios, por exemplo, tem três vias de transporte (fornecedor-fábrica, fábrica-distribuidor e distribuidor-varejista), enquanto uma cadeia de suprimentos não-serial de quatro estágios e dois nós por estágio tem 12 possíveis vias de transporte. Quando consideramos também a dimensionalidade do espaço de estados, o horizonte de planejamento utilizado e a aleatoriedade advinda da incerteza nas demandas e nos tempos de espera, notamos que o problema abordado sofre do chamado *mal da dimensionalidade*. *Vemos, portanto, que é um desafio resolver o problema apresentado usado métodos de AR.* Assim, o uso de métodos de AR Profundo em um problema de PO cujo espaço de ações é contínuo e tem mais dimensões é uma das contribuições da presente pesquisa. Além disso, pelo nosso conhecimento, este é o primeiro trabalho que lida com o problema considerando tempos de espera estocásticos e usando métodos *Policy Gradient*. Na Seção 2.2, apresentamos os trabalhos relacionados, classificando-os pelo tipo de abordagem (planejamento de produção ou baseada em pedidos) e explicamos em mais detalhes as contribuições da presente pesquisa.

De forma geral, este trabalho tem por objetivo avaliar a adequação de métodos de AR do tipo *Policy Gradient* para a solução do problema de cadeia de suprimentos apresentado. As principais perguntas a serem respondidas para atender a esse objetivo são listadas a seguir.

1. É possível resolver o problema com um único (agente) tomador de decisões centralizado, em cadeia com quatro estágios e dois nós por estágio, considerando demandas estocásticas e tempos de espera constantes, utilizando métodos *Policy Gradient*?

2. Tais métodos resolvem também o problema com demandas incertas sazonais e tempos de espera estocásticos?

3. Como é possível adaptar a modelagem PDM do problema para que o mesmo seja resolvido de forma satisfatória com os métodos *Policy Gradient*?

4. Quão boa é a solução encontrada por tais métodos em relação ao *baseline* utilizado (agente baseado em modelo de PL)?

5. Quais métodos *Policy Gradient* são mais apropriados para o problema em questão?

6. Quais são os passos a serem utilizados na metodologia experimental para se alcançar bons resultados com tais métodos?

7. Métodos *Policy Gradient* conseguem resolver uma versão estendida do problema, que considera multiprodutos?

Para responder a tais perguntas, foi necessário primeiramente aprofundarmos o entendimento do problema e dos métodos a serem empregados. Na primeira parte do Capítulo 2 apresentamos o detalhamento do problema abordado e avaliamos os trabalhos similares da literatura, comparando-os com a nossa abordagem. Em seguida, propomos uma formulação do problema como PDM e apresentamos um modelo de PL com parâmetros incertos para o problema. Já no Capítulo 3 apresentamos os conceitos de AR, partindo da definição de um problema como PDM e apresentando os conceitos da área a partir da abordagem de Sutton & Barto [2018]. Um maior enfoque é dado aos métodos *Policy Gradient* utilizados para resolver o problema que abordamos neste trabalho.

Em uma primeira fase de aplicação dos métodos, usamos o algoritmo Proximal Policy Optimization (PPO) [Schulman et al., 2017] para resolver o problema, considerando tempos de espera constantes e demandas incertas regulares (não-sazonais), em um cenário utilizado como estudo de caso. Os resultados desta fase foram apresentados na ICCL 2020, *International Conference on Computational Logistics*, e o artigo que escrevemos [Alves & Mateus, 2020] foi incluído nos anais da conferência publicados na série *Lecture Notes in Computer Science*. Tais resultados são apresentados neste texto

no Apêndice A. Já no Capítulo 4, apresentamos a segunda fase da pesquisa, na qual estendemos o trabalho para considerar demandas incertas sazonais, tempos de espera estocásticos e capacidades de processamento nas fábricas. Além disso, aprofundamos a análise experimental, considerando 17 diferentes cenários para melhor avaliar a adequação do PPO para resolver o problema. Os resultados desta segunda fase foram descritos em um artigo científico [Alves & Mateus, 2021] submetido para a revista *Annals of Operations Research*, e que encontra-se, no momento da escrita deste texto, em processo de revisão.

Em uma terceira fase, apresentada no Capítulo 5, comparamos diferentes algoritmos *Policy Gradient* (A2C, DDPG, PPO, TD3 e SAC) para verificar quais seriam aqueles com melhor desempenho para o problema abordado. Nos experimentos usamos 8 dos 17 cenários da segunda fase, e aprofundamos a análise incluindo ferramental estatístico para comparações múltiplas. Os resultados encontrados foram apresentados na ICAISC 2021, *The 20th International Conference on Artificial Intelligence and Soft Computing*, e o artigo escrito [Alves et al., 2021] será incluído nos anais da conferência a serem publicados na série *Lecture Notes in Artificial Intelligence*. Na última fase deste trabalho, apresentada no Capítulo 6, passamos a trabalhar em uma versão estendida do problema, que considera multiprodutos. Foram definidos cenários com dois ou três produtos, com diferentes configurações de demandas e de custos por tipo de produto. Nos experimentos utilizamos o algoritmo PPO, por ter alcançado o melhor desempenho nos experimentos anteriores, e o comparamos com um *baseline* baseado em modelo PL.

Por fim, no Capítulo 7, apresentamos as conclusões deste texto e apontamos possíveis caminhos para trabalhos futuros.

Capítulo 2

Problema e Modelagem

O presente capítulo apresenta e detalha o problema abordado na presente tese. A descrição geral do problema é apresentada na Seção 2.1. Na Seção 2.2 são apresentados trabalhos que lidaram com problemas similares, destacando-se as diferenças para o presente trabalho e as contribuições trazidas pelo mesmo. A formulação do problema como um PDM é apresentada na Seção 2.3. E, por fim, o modelo de PL com parâmetros incertos desenvolvido para o problema é apresentado na Seção 2.4.

Vale notar que a definição apresentada neste capítulo foi utilizada nos experimentos apresentados nos capítulos 4 e 5. Já no Capítulo 6, são apresentados experimentos referentes a uma versão estendida do problema, que considera multiprodutos. As diferenças na definição do problema para esta extensão são apresentadas no próprio Capítulo 6.

2.1 O Problema Abordado

Nesta seção, detalhamos o problema de cadeia de suprimentos que é considerado na presente pesquisa. Trabalhos relacionados que aplicam métodos de AR em cadeias de suprimentos multiestágio são geralmente inspirados no chamado *MIT Beer Game* [Sterman, 1989]. Este jogo foi proposto para o estudo do efeito chicote (*Bullwhip effect*[1]) em uma cadeia de suprimentos serial (ou seja, com apenas um nó por estágio), através do comportamento de indivíduos que tenham acesso apenas a informações locais (apenas do nó que controla). Neste contexto, cada nó da cadeia pode ser visto como um ator independente, que precisa atender às demandas de seu sucessor direto na cadeia e decidir quanto pedir para seu predecessor. No *MIT Beer Game* há um

[1]O efeito chicote se refere ao nível crescente de oscilações nos estoques, à medida que se sobe na cadeia de suprimentos, em resposta a mudanças na demanda do cliente final.

fornecedor, uma fábrica, um distribuidor e um varejista. O varejista, nó do último estágio, atende aos pedidos dos consumidores finais e precisa fazer pedidos de compra para o distribuidor. O distribuidor, por sua vez, atende os pedidos do varejista, e coloca pedidos de compra para as fábricas, e assim sucessivamente. O fornecedor, nó do primeiro estágio, obtém material de uma fonte externa. Portanto, neste caso, o qual chamamos aqui de *abordagem baseada em pedidos*, as decisões, de certa forma, *sobem* a cadeia, já que um pedido de um nó é para o estágio predecessor. Uma outra possível abordagem é considerar que as decisões de toda a cadeia de suprimentos são baseadas apenas nas demandas dos consumidores finais, como recomendado por Lee et al. [1997], para combater o efeito chicote. Neste contexto, há um único agente, um tomador de decisões centralizado, que controla todas as operações da cadeia, e o problema pode ser visto como uma *abordagem de planejamento de produção multiperíodo* [Pinedo, 2009; Stadtler et al., 2015]. Nesta configuração, as decisões em cada período são: quanta matéria-prima produzir em cada fornecedor, quanto material transportar de um nó para seus sucessores e quanto material estocar no armazém local de cada nó. Portanto, as decisões a serem tomadas, de certa forma, *descem* a cadeia, já que o transporte de material é decidido de cada nó para os nós do estágio subsequente. Neste trabalho, nós consideramos o segundo caso, ou seja, um problema de planejamento de produção multiperíodo com um único agente, que tem por objetivo atender às demandas incertas dos clientes com o mínimo custo total de operação.

Uma cadeia de suprimentos pode operar com abordagens de *backlog* ou *lost sales*. No caso da abordagem com *backlog*, uma demanda de cliente de um período pode ser atendida posteriormente, enquanto na abordagem *lost sales* uma demanda não atendida é descartada. Nós consideramos a abordagem *lost sales* neste trabalho, e um custo de penalização é considerado quando uma demanda de cliente não é atendida. Uma outra definição importante é que consideramos o caso de indústrias (processos) de manufatura contínua [Pinedo, 2009], logo todas as quantidades de materiais são valores contínuos. De qualquer forma, acreditamos que a metodologia proposta neste trabalho poderia ser adaptada para o caso de indústrias que trabalham com quantidades discretas, e sugestões sobre como isso poderia ser feito são dadas na Seção 4.2.2. Por fim, como apresentado no Capítulo 1, nós consideramos que os armazéns locais de cada nó da cadeia são capacitados e que os fornecedores e as fábricas possuem capacidades de produção e de processamento de matéria-prima, respectivamente.

Apresentamos agora a dinâmica da cadeia de suprimentos que estudamos no presente trabalho. Todos os cenários experimentados consideram uma cadeia de suprimentos de quatro estágios com dois nós por estágio, como a apresentada na Figura 1.1. No começo do horizonte de planejamento (período $t = 0$), há uma quantidade inicial de

material (matéria-prima ou produto) estocada no armazém local de cada nó, há também matérias-primas sendo produzidas pelos fornecedores (e que estarão disponíveis nos próximos períodos), e matérias-primas e produtos sendo transportados de um nó para seus sucessores na cadeia. Há ainda pedidos, ou demandas, de clientes para cada varejista que deverão ser atendidas no próximo período ($t = 1$). Os passos apresentados abaixo são repetidos até o final do horizonte de planejamento:

1. O tomador de decisões centralizado (ou seja, o agente) decide a quantidade de material a ser produzida em cada fornecedor e a quantidade de material a ser transportada de cada nó para cada um dos seus sucessores. A quantidade de material a ser mantida em estoque é definida indiretamente pelo material que sobrar em cada nó. Varejistas não são controlados, já que eles atendem às demandas dos clientes sempre que possível.

2. Um novo período é considerado (agora $t = t + 1$).

3. Fluxo de material:

   - Matérias-primas que estavam sendo produzidas em cada fornecedor e que agora estão disponíveis, por ter expirado o tempo de espera, são estocadas no armazém do fornecedor.
   - Matérias-primas e produtos em transporte, com tempo de espera expirado, são entregues em cada nó e estocados em seu armazém.
   - Possíveis excessos de materiais (quantidades que estejam acima das capacidades dos estoques) são descartados e custos de penalização são contabilizados para cada unidade de material.

4. Varejistas atendem às demandas dos clientes usando seus estoques de produto. Se não há produto suficiente, um custo de penalização é considerado para cada unidade de produto faltante.

5. As decisões do agente são seguidas:

   - A produção de matéria-prima é disparada em cada fornecedor. O tempo de espera de produção é realizado, ou seja, é definido quando a matéria-prima estará disponível no estoque do fornecedor. Vale ressaltar que o material sendo produzido impacta a capacidade do fornecedor somente no período no qual a produção é disparada.

- Para cada nó (exceto os varejistas), a quantidade de material a ser transportada para cada nó sucessor é removida do estoque e despachada para o sucessor correspondente. O tempo de espera de transporte é realizado, ou seja, é definido quando o material estará disponível no estoque do sucessor. No caso das fábricas, as matérias-primas são processadas em produtos finais antes de serem despachadas. Vale ressaltar que consideramos que o tempo de processamento das matérias-primas é irrisório em comparação aos tempos de transporte e, portanto, não o tratamentos diretamente.

6. Demandas incertas, e potencialmente sazonais, de clientes para cada varejista a serem atendidas no próximo período são realizadas.

Ao final do horizonte de planejamento, considera-se que haveria um novo planejamento, tratando condições atualizadas da cadeia, tais como custos atualizados, nova distribuição de demanda, etc. Portanto, a ideia é resolver o problema de forma a minimizar os custos totais de operação considerando o período determinado pelo horizonte de planejamento.

É importante enfatizar que as demandas são incertas e que, a cada período (t) o agente vê apenas a realização das demandas para o próximo período ($t + 1$). Como os varejistas não são controlados pelo agente, ele não pode se beneficiar por saber as demandas apenas do próximo período. Um outro ponto importante é que, como os tempos de espera para produzir matéria-prima e para transportar material são incertos, o agente tem que tomar as decisões de produção e transporte sem saber quando os materiais estarão disponíveis (ou serão entregues). Os valores de tempo de espera são realizados depois que as decisões já foram tomadas e, somente no próximo estado, o agente saberá quando o material chegará. Além disso, e como será apresentado na formulação PDM (Seção 2.3), o agente sabe a quantidade de material exata que chegará (ou será entregue) apenas no período seguinte, já que as quantidades de material chegando nos períodos após o próximo são todas somadas gerando um único valor.

Sobre os métodos de solução, há diferentes abordagens dependendo de como são tratados os parâmetros incertos, como demandas e tempos de espera. Na prática, é comum o uso de previsões de demanda e tempos de espera médios e, neste caso, o problema se torna determinístico e pode ser resolvido por modelos de PL [Stadtler et al., 2015]. Uma outra abordagem é resolver o problema levando em conta a incerteza dos parâmetros através do uso de técnicas que podem tratar estocasticidade, como métodos de Programação Estocástica ou AR Profundo. Nós usamos métodos de AR Profundo para resolver o problema e um agente baseado na solução de um modelo de PL é usado como *baseline*. Há ainda a opção de lidar com o problema usando abordagens com um

único ou com múltiplos agentes. Abordagens multiagente são mais comuns quando o problema é modelado considerando que cada nó da cadeia é um ator independente. Na nossa abordagem, o problema é resolvido com um único agente que toma várias decisões diferentes no mesmo período.

2.2 Trabalhos Relacionados

Há alguns trabalhos relacionados que usam AR para resolver problemas de cadeias de suprimentos, mas vários deles são baseados em métodos tabulares, como Q-Learning, por exemplo [Giannoccaro & Pontrandolfo, 2002; Chaharsooghi et al., 2008; Mortazavi et al., 2015]. Como métodos tabulares não podem lidar de forma apropriada com problemas com espaços de estado e ação contínuos, como o endereçado neste trabalho, nós focamos aqui nos trabalhos que usam abordagens de AR Profundo para resolver problemas similares. Primeiramente, apresentamos trabalhos que tratam o problema com a abordagem de planejamento de produção, como fazemos em nossa pesquisa. Depois apresentamos trabalhos que tratam o problema com a abordagem baseada em pedidos de compra.

Kemmer et al. [2018] usam *Aproximate SARSA* e três versões do algoritmo *REINFORCE* em uma cadeia de dois estágios. O cenário é formado por uma fábrica e três armazéns com demandas crescentes, sem tempos de espera, e um horizonte de planejamento de 24 períodos. O estado é formado pelos níveis de estoque e as demandas dos últimos dois períodos, já as ações se referem à produção da fábrica e ao transporte de produto (mas o espaço de ações é reduzido a somente três níveis de produção e de transporte). As recompensas são o lucro, considerando os custos de operação e os *backlogs*. O *baseline* utilizado é a política (r-Q) que tem como ideia central disparar um pedido de Q unidades caso o estoque caia a um nível mínimo r. E, geralmente, os trabalhos utilizam alguma heurística para encontrar bons valores para r e Q. Todos os agentes são melhores que o *baseline* no cenário com apenas um armazém, e duas versões do *REINFORCE* ganham do *baseline* no cenário com três armazéns. O trabalho é estendido por Hutse [2019], incluindo tempos de espera determinísticos (não-nulos), dois tipos de produto, espaços de ação contínuos, e quatro tipos de cenários com demandas estocásticas. O autor usa uma Deep Q-Network (DQN) para ações discretas e DDPG para ações contínuas. O estado é composto pelos níveis de estoque, produção, transporte e as últimas x (um parâmetro) demandas, para cada combinação nó-produto. As ações são, para cada produto, quanto produzir e enviar para cada varejista; usando níveis agregados no caso discreto e limitando os valores máximos de ação no caso con-

tínuo. As recompensas são o lucro, considerando custos operacionais, inclusive faltas de estoque. O *baseline* é a política (r-Q) e os agentes são melhores que o *baseline* em todos os cenários (com uma fábrica, dois ou três varejistas, e um ou dois produtos).

Peng et al. [2019] usam *Vanilla Policy Gradient* em uma cadeia de suprimentos capacitada com uma fábrica, com depósito local, e três varejistas, considerando custos, balanceados e não-balanceados, e demandas estocásticas regulares e sazonais. O estado é formado pelos níveis de estoque e pelas últimas duas demandas; e as ações são o quanto produzir e o quanto enviar para cada varejista. As ações são dependentes do estado, e os autores usam mecanismos para lidar com as dificuldades inerentes ao uso desta abordagem ao tratar as saídas da rede neural. As recompensas são o lucro, considerando os custos operacionais e penalização pelo não atendimento à demanda. O agente de AR Profundo alcança resultados melhores que o *baseline*, a política (r-Q), em todos os cenários experimentados.

Os trabalhos a seguir tratam o problema usando uma abordagem baseada em pedidos. Gijsbrechts et al. [2019] propõem uma prova de conceito do uso de AR Profundo em três problemas diferentes: *dual-sourcing* ou *dual-mode*, *lost sales*, e modelos de estoque multiestágio. Na configuração de multiestágio, os estados são representados pelos níveis de estoque e pedidos dos depósitos e varejistas, enquanto as ações são os pedidos para cada nó (agregados por níveis-base de estoque dependentes de cada estado). Os autores aplicam A3C em dois cenários diferentes com um depósito e dez varejistas, considerando demandas estocásticas. O agente A3C tem desempenho melhor que a política de estoque-base usada como *baseline*. Tal política consiste em fazer pedidos de forma a manter o estoque sempre acima de um nível-base pré-determinado.

Já Oroojlooyjadid [2019] usa uma DQN customizada para resolver o *MIT Beer Game*, uma cadeia de suprimentos linear de quatro estágios, considerando demandas determinísticas e estocásticas. O autor trata o problema como descentralizado, com multiagentes cooperativos. Cada agente conhece apenas a informação local e, para evitar competição, há um mecanismo engenhoso que fornece um retorno para cada agente ao final de um episódio. Nos experimentos, somente um agente usa DQN e os outros seguem uma heurística de estoque-base. O estado é formado pelos níveis de estoque, demandas, pedidos, e produtos sendo entregues, dos últimos m (um parâmetro) períodos. As ações se referem a quanto pedir a mais ou a menos do que o pedido recebido do nível subsequente, e os intervalos usados são $[-2, 2]$ e $[-8, 8]$. As recompensas são os custos de estoque e *backlog*. Experimentos mostram que usar DQN para um nó obtém resultados melhores que usar a política de estoque-base para todos os nós.

Hachaïchi et al. [2020] usam PPO e DDPG para resolver um problema de reposição de estoque em uma cadeia de dois estágios. O fornecimento é ilimitado, demandas

de clientes são regulares (não-sazonais) e menores do que as capacidades de estoque. Estados são compostos por níveis de estoque, material em transporte e demandas e cliente dos últimos m (um parâmetro) períodos. Ações são representadas pelos pedidos dos centros de distribuição e das lojas. O objetivo é maximizar o lucro e, portanto, as recompensas são dadas pelos valores de venda menos os custos de estoque e dos pedidos. Experimentos com um cenário fixo mostram que os resultados com o DDPG são instáveis e que o PPO alcança uma diferença de 6,4% para um *baseline* que considera o atendimento de todas as demandas observadas.

Geevers [2020] usa PPO em três cenários, considerando cadeias de suprimentos lineares e divergentes de dois estágios. Considerando o último cenário, um estudo de caso industrial, os estoques são capacitados e o fornecimento é ilimitado. O problema é resolvido considerando um tipo de produto, tempos de espera constantes, demandas incertas regulares, e um horizonte de planejamento de 50 períodos. Estados são formados pelo estoque total, número total de pedidos, os níveis de estoque de cada nó, e, para cada par de nós, os pedidos de compra e o material em transporte. Já as ações são formadas pelos pedidos para cada ponto de estoque. O objetivo é minimizar os custos de estoque e *backlog*. Nos experimentos, o agente PPO alcança resultados com grande variância. Considerando 10 execuções de treinamento, algumas execuções são melhores que o *baseline* (heurística de estoque-base), enquanto outras alcançam resultados ruins. O autor argumenta que o método adotado é instável e, portanto, ainda não é adequado para uso prático.

Perez et al. [2021] usam PPO para resolver um problema de gestão de estoque em um cadeia de suprimentos de quatro estágios com abordagem *make-to-order*. Os nós têm estoque local (sem limites de capacidade), a produção é limitada, há um único produto, e o horizonte de planejamento considerado é de 30 períodos. Há um varejista para atender às demandas incertas (regulares) dos clientes, e os tempos de espera são heterogêneos (mas sem incerteza). Estados são formados pela demanda, pelos níveis de estoque, e o material em transporte. Já as ações são formadas pelos pedidos. As recompensas são o lucro calculado para cada nó da cadeia. Os autores experimentam com um cenário de estudo de caso, considerando as opções de *backlog* e *lost sales*, e comparam PPO com três modelos PL determinísticos (padrão, com *rolling horizon* e com *shinking horizon*) e com um modelo estocástico de múltiplos estágios. Todos os modelos PL são melhores que o PPO na abordagem com *backlog*. No cenário com *lost sales*, PPO só é melhor que um dos modelos (PL determinístico com *rolling horizon*). Os autores argumentam que a solução do PPO tem uma carga mais balanceada e que, portanto, teria o potencial de ser mais resiliente a possíveis disrupções.

A Tabela 2.1 apresenta uma comparação dos trabalhos relacionados com as abor-

dagens da presente pesquisa. Os trabalhos são agrupados pelo tipo de abordagem (planejamento de produção ou baseada em pedidos). Para cada trabalho é apresentada a configuração da cadeia de suprimentos (nós por estágio, produtos e horizonte de planejamento), o tipo de incerteza para as demandas (regular ou sazonal), indicação se os tempos de espera são determinísticos ou estocásticos, informação sobre os espaços de estado e ação (se são contínuos ou discretos, e o número de dimensões), o método de AR e o *baseline* utilizados. Como pode ser visto na tabela, pelo nosso conhecimento, nosso trabalho é o primeiro que lida com o problema considerando uma abordagem de planejamento de produção em uma cadeia de suprimentos com mais de dois estágios e com tempos de espera estocásticos. Como nós consideramos mais de um nó por estágio, o número de dimensões dos espaços de estados e ações são maiores que dos trabalhos similares, e, portanto, o problema é maior e mais complexo. Se incluirmos os trabalhos baseados em pedidos na comparação, a maioria dos trabalhos relacionados lida com cadeias de dois estágios, com tempo de espera constante e espaços de ação com menos dimensões. Portanto, essa pesquisa é a primeira a aplicar e comparar métodos estado-da-arte de AR Profundo em cadeias de suprimentos com tais características.

O trabalho de Perez et al. [2021] usa PPO e um *baseline* baseado em um modelo PL em uma cadeia de quatro estágios. Contudo, eles lidam com o problema em uma abordagem baseada em pedidos e não consideram estoques capacitados, demandas sazonais, nem tempos de espera estocásticos, os espaços de estados e de ações são discretos, e o horizonte de planejamento é menor. Além disso, a metodologia experimental do trabalho citado carece de passos importantes, tais como, o ajuste de hiperparâmetros, execução de múltiplos treinamentos com diferentes sementes de números aleatórios, e normalização de recompensas [Henderson et al., 2018; Raffin et al., 2019], enquanto todos esses passos são considerados em nossa pesquisa. Geevers [2020] também usou PPO e considerou espaços de ação contínuos com várias dimensões, mas ele trata apenas demandas regulares e tempos de espera determinísticos, em uma cadeia de dois estágios, e, além disso, os resultados alcançados são instáveis.

Nossas principais contribuições em relação aos trabalhos da literatura podem ser sumarizadas como se segue:

1. Nosso trabalho é o primeiro a usar AR Profundo para lidar com o problema em uma abordagem de planejamento de produção em uma cadeia com mais de dois estágios.

2. Somos os primeiros a considerar tempos de espera estocásticos para o problema abordado, considerando tanto abordagens de planejamento de produção, quanto baseada em pedidos.

3. Nossa pesquisa e o trabalho de Perez et al. [2021] são os únicos que lidam com uma cadeia de suprimentos não-serial de quatro estágios usando AR Profundo. Isso leva a um problema com espaços de estados e de ações com mais dimensões, e, portanto, mais difícil de resolver. Mas, diferente do trabalho de Perez et al. [2021], nós consideramos uma abordagem de planejamento de produção com demandas incertas sazonais, tempos de espera estocásticos, estoques capacitados, espaços de estado e de ação contínuos, e um horizonte de planejamento maior.

4. A formulação PDM e, especialmente, as adaptações para utilizá-la propostas neste trabalho (na Seção 4.2.2) permitem obter bons resultados com métodos de AR Profundo.

5. Nós aplicamos e comparamos cinco métodos *Policy Gradient* estado-da-arte para resolver o problema abordado.

6. Nós adotamos uma metodologia experimental robusta, alcançando bons resultados com PPO e SAC, considerando espaços de ação contínuos com mais dimensões que os trabalhos relacionados.

7. Por fim, nós usamos testes-t de Welch para comparar e selecionar os algoritmos utilizados.

2.3 Formulação PDM

A formulação do problema como um Processo de Decisão de Markov (PDM) é muito importante já que, mais do que apenas um passo necessário, ela pode impactar diretamente na qualidade dos resultados alcançados pelos algoritmos de AR. A formulação PDM envolve a definição dos estados, ações, recompensas e da dinâmica do ambiente (ou função de transição) a serem usados para resolver o problema. Na modelagem, tomamos algumas decisões importantes que ajudam na obtenção de bons resultados com algoritmos de AR Profundo. A primeira delas é que procuramos manter os espaços de estados e ações com o menor número de dimensões possível. A segunda foi termos modelado o espaço de ações de forma independente dos estados e gerando apenas estados viáveis. E, por fim, decidimos reescalar os espaços de estados e ações para a faixa $[-1, 1]$. Neste texto, preferimos apresentar aqui uma definição direta da formulação PDM e, depois, ao explicar a metodologia de aplicação dos algoritmos de AR Profundo (Seção 4.2.2), apresentamos as adaptações feitas na formulação (como a normalização de estados e ações, por exemplo).

Tabela 2.1. Comparação com os trabalhos relacionados. Os trabalhos são agrupados por tipo de abordagem: planejamento de produção ou baseada em pedidos. Para a configuração da cadeia: $Config.$ indica o número de nós por estágio, P o número de produtos e H o horizonte de planejamento. A coluna $Dem.Saz.$ indica se as demandas são sazonais; a coluna $T.Esp.Est.$ mostra se os tempos de espera são estocásticos. Nas colunas *Estados* e *Ações*, $D(M)$ ou $C(M)$ indicam se é um espaço discreto ou contínuo, respectivamente, de M dimensões. A coluna *Alg. AR* apresenta os métodos de AR usados, e a última coluna apresenta o *baseline* utilizado. Os valores se referem aos cenários mais complexos experimentados em cada trabalho.

Autores	**Cadeia**			**Dem.**	**T.Esp.**	**Estados**	**Ações**	**Alg. AR**	**Baseline**
	Config.	**P**	**H**	**Saz.**	**Estoc.**				
Abordagens de planejamento de produção									
Kemmer et al. [2018]	1-3	1	24	✓		D (10)	D (4)	VPG	Heurística (r, Q)
Hutse [2019]	1-3	2	52	✓		D (30)	C (4)	DQN,DDPG	Heurística (r, Q)
Peng et al. [2019]	1-3	1	25	✓		C (12)	C (4)	VPG	Heurística (r, Q)
Este trabalho	2-2-2-2	1	360	✓	✓	C (27)	C (14)	A2C,DDPG,PPO,SAC,TD3	baseado em PL
Abordagens baseadas em pedidos									
Gijsbrechts et al. [2019]	1-10	1	*cont.*			D (35)	D (2)	A3C	Heurística estoque base
Oroojlooyjadid [2019]	1-1-1-1	1	*fixo*			D (50)	D (1)	DQN cust.	Heurística estoque base
Hachaïchi et al. [2020]	1-3	1	52			C (48)	C (4)	PPO,DDPG	Heurística customizada
Geevers [2020]	4-5	1	50			C (48)	C (9)	PPO	Heurística estoque base
Perez et al. [2021]	2-3-2-1	1	30			D (68)	D (11)	PPO	baseado em PL

2.3.1 Espaço de Estados

Um estado, para um dado período t, é um vetor contínuo de 27 dimensões contendo os seguintes valores (um exemplo de estado é apresentado na Figura 2.1).

- O nível atual de estoque de cada nó.
- Para cada fornecedor:
  - a quantidade de matéria-prima que está sendo produzida e que estará disponível no próximo período $(t + 1)$;
  - a soma das quantidades de matéria-prima que estão sendo produzidas e que estarão disponíveis nos períodos após o próximo.
- Para cada outro nó:
  - a quantidade de material em transporte e que chegará no nó em questão no próximo período (é a soma do material enviado pelos dois nós do estágio anterior);
  - a soma das quantidades de material em transporte que chegarão nos períodos após o próximo.
- As demandas dos clientes finais de cada varejista para o próximo período $(t + 1)$.
- O número de períodos restantes até o fim do episódio, já que a política aprendida é não-estacionária.

Disponível no fornecedor		*estoque*	*Chegará nas fábricas*		*estoque*	*Chegará nos distribuidores*		*estoque*	*Chegará nos revendores*		*estoque*	*Demandas clientes*	*Períodos restantes*
> t+1	*t+1*		*> t+1*	*t+1*		*> t+1*	*t+1*		*> t+1*	*t+1*		*t+1*	
		400			**15**			**382**			**118**		
340	**0**	Fornec.1	**420**	**280**	Fábrica1	**124**	**0**	Distrib.1	**340**	**100**	Varejista1	**138**	***360-t***
105	**330**	Fornec.2	**250**	**25**	Fábrica2	**87**	**99**	Distrib.2	**423**	**124**	Varejista2	**109**	
		325			**0**			**111**			**4**		

Figura 2.1. Exemplo de um estado para um dado período t.

Nesta definição, optamos por considerar o material em transporte do ponto de vista do destino, ou seja, a soma dos materiais independente da origem. Fizemos isso porque, caso separássemos o material por origem, teríamos um espaço de estados com mais dimensões, e essa informação adicional não traria nenhuma vantagem para

o agente de AR. Já a ideia de usar um valor sumarizado para o material que estará disponível após o próximo período é para evitar um aumento nas dimensões do estado com informação que não é tão precisa, uma vez que com tempos de espera estocásticos, tais valores têm alta probabilidade de sofrerem alterações nos períodos seguintes.

Para obter bons resultados com algoritmos de AR Profundo, como o PPO, é importante normalizar (ou reescalar para o intervalo $[-1, 1]$) os valores dos estados [Raffin et al., 2019]. Na Seção 4.2.2 apresentamos como a normalização dos valores dos estados é feita em nossos experimentos.

2.3.2 Espaço de Ações

Uma ação é um vetor contínuo de 14 dimensões com os seguintes valores (um exemplo de ação é apresentado na Figura 2.2).

- A quantidade de material a ser produzida em cada fornecedor.
- Para cada nó (exceto os varejistas):
  - a quantidade de material a enviar para cada um dos dois nós do próximo estágio.

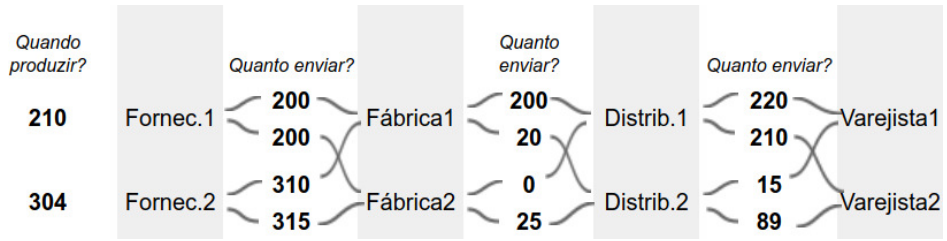

Figura 2.2. Exemplo de uma ação: as decisões são relacionados à produção de matérias-primas nos fornecedores e transporte entre nós (níveis de estoque são definidos indiretamente e revendedores não são controlados).

Quanto menor o espaço de ações (menos dimensões), menos complexo é o problema. Dessa forma, nossa primeira decisão foi considerar que os varejistas não precisam ser controlados, já que eles devem atender às demandas dos clientes sempre que possível. Outra decisão tomada para reduzir o espaço de ações está relacionada ao fato de que, uma vez decidida a quantidade de material a ser transportada, o material a ser mantido em estoque pode ser calculado indiretamente.

Um aspecto importante a ser considerado na definição da representação de ações é como lidar com possíveis ações inviáveis. Em relação à produção de material em cada

fornecedor, é simples evitar valores inviáveis apenas limitando a ação à capacidade do fornecedor. Mas, para o material a ser transportado de um nó para outro, a situação é mais complexa, já que o nível de estoque dos nós muda ao longo da simulação. Na Seção 4.2.2 apresentamos como lidamos com esta questão e geramos apenas ações viáveis em nossos experimentos. Na mesma seção, apresentamos como tratamos a normalização dos valores de ação, já que ela é importante para obter bons resultados com alguns algoritmos de AR Profundo, como o PPO [Raffin et al., 2019].

2.3.3 Dinâmica do Ambiente

Como nós usamos abordagens de AR Profundo *model-free*, uma simulação da cadeia de suprimentos é usada para representar a dinâmica do ambiente. Quase todas as operações da cadeia são simuladas de forma determinística, ou seja, qualquer quantidade de material definida pelos valores de ação (a ser fornecida ou transportada) é seguida pela simulação, já que as ações sempre representam quantidades viáveis. O comportamento não-determinístico da simulação se deve às demandas incertas dos clientes e aos tempos de espera. Demandas de cliente e tempos de espera são amostrados em cada período a partir de uma distribuição estatística e são realizados como apresentado na definição do problema (Seção 2.1). Há ainda um tratamento para um possível excesso de material nos armazéns, que pode acontecer quando um nó recebe mais material que ele é capaz de estocar. Isso é possível porque a soma do material que está chegando dos dois nós predecessores com o material atualmente em estoque, pode alcançar um valor maior que a capacidade de estoque do nó. A simulação considera que todo o material que chega deve passar pelo estoque, mesmo que ele não seja mantido para o próximo período. O excesso de material é descartado e um custo de penalização é considerado para cada unidade de material.

2.3.4 Recompensas

A definição das recompensas é crucial para o sucesso de algoritmos de AR. Para muitos problemas não é clara a melhor forma de definir a recompensa para o agente, especialmente porque, em muitos casos, pode ser fácil definir um retorno no final de um episódio (sucesso ou falha), mas pode ser difícil definir um retorno a cada passo da simulação. Contudo, no problema proposto neste trabalho, como ele é um problema de minimização de custos, parece fazer muito sentido usar como recompensa o negativo dos custos totais de operação[2]. Portanto, a recompensa usada é o negativo da soma de

[2]Nós também experimentamos como recompensa o inverso dos custos de operação multiplicado por uma constante (veja no Apêndice B), mas não obtivemos bons resultados com essa abordagem.

todos os custos incorridos em um período (custos de produção, transporte, fabricação, estoques, e penalização pelo material descartado devido às capacidades de estoque e pelo não atendimento às demandas dos clientes).

2.4 Modelo de Programação Linear com Parâmetros Incertos

Nesta seção apresentamos um modelo de otimização para o problema. Nessa abordagem, o problema pode ser modelado matematicamente considerando uma função objetivo a ser maximizada (ou minimizada), sujeita a um conjunto de restrições. Se a função objetivo e as restrições forem lineares e os parâmetros determinísticos, podemos formular o problema como um Programa Linear[3], na sua forma canônica dado por $max\{c'x : Ax \leq b, x \geq 0\}$, onde $A \in \mathcal{R}^{m \times n}$, $c \in \mathcal{R}^n$ e $b \in \mathcal{R}^m$. Podemos dizer que c é o vetor de custos, e buscamos minimizar uma função de custo linear dada por $c'x$, sobre todos os vetores n-dimensionais $x \geq 0$ sujeitos a um conjunto de restrições de desigualdade dadas por $Ax \leq b$. A região viável para esse problema é um poliedro convexo definido pelos semi-espaços formados pelas equações de desigualdade (restrições). O objetivo então é encontrar um ponto no poliedro no qual a função de custo tem seu menor (ou maior) valor possível, se este ponto existir. Existem métodos eficientes para resolver Programas Lineares, como o método Simplex ou métodos de pontos interiores. Portanto, no caso do problema que abordamos nesta pesquisa, se as demandas e os tempos de espera fossem determinísticos, poderíamos resolver o problema eficientemente com tais métodos. Nós nos baseamos nessa ideia para criar o *baseline* utilizado nos experimentos, ou seja, nós resolvemos o problema considerando demandas esperadas e tempos de espera médios (e, portanto, determinísticos) e usamos a solução do modelo PL para criar um agente que atua nos cenários estocásticos. De toda forma, o modelo apresentado a seguir considera o problema como ele foi proposto, ou seja, considerando demandas e tempos de espera estocásticos. Neste caso podemos dizer que criamos um modelo de PL com parâmetros incertos para o problema, devido à estocasticidade dos parâmetros.

Os conjuntos do modelo são apresentados na Tabela 2.2, onde q é o número de nós da cadeia, h o horizonte de planejamento (ou tamanho do episódio), e l^{max} é o máximo valor possível de tempo de espera. Período 0 se refere ao estado inicial da cadeia. As variáveis do modelo são apresentadas na Tabela 2.3 e seus parâmetros na Tabela 2.4.

[3]Aos leitores interessados em mais informações sobre Programação Linear sugere-se o livro de Bertsimas & Tsitsiklis [1997].

Note que p_{ijn}, f_n e t_{ijnm} são binários, e todos os outros parâmetros são inteiros. f_n é igual a 1 para todas as fábricas, e vale 0 para os demais nós. p_{ijn} é usado para definir quais nós são fornecedores e também para mapear os tempos de espera. Há um valor estocástico de tempo de espera para cada fornecedor em cada período, de forma que $p_{ijn} = 1$ somente se n é um fornecedor, e a realização do tempo de espera para produzir material em cada fornecedor no período $i - j$ é j, caso contrário p_{ijn} é zero. De forma similar, t_{ijnm} define quais pares de nós podem podem ter transporte e também mapeia os tempos de espera. Vale ressaltar que optamos por definir um modelo que é genérico em relação à configuração da cadeia, ou seja, a definição do número de estágios, da quantidade de nós por estágio e de quais deles produzem matérias-primas, processam material ou atendem demandas de clientes, é feita através dos parâmetros do modelo. Dessa forma, ele poderia ser usado para resolver o problema também com cadeias com outras configurações além das experimentadas neste trabalho, como cadeias com mais estágios, por exemplo.

Tabela 2.2. Conjuntos do modelo PL

Conjunto	**Descrição**
$N = \{1, ..., q\}$	conjunto de nós da cadeia de suprimentos
$I = \{0, 1, ..., h + l^{max}\}$	conjunto de períodos (ou *time steps*)
$J = \{1, ..., l^{max}\}$	conjunto de tempos de espera possíveis

Tabela 2.3. Variáveis do modelo PL

Var.	**Descrição**
S_{in}	nível de estoque do nó n no período i
T_{ijnm}	quantidade de material enviada do nó n no período $i - j$ para chegar no nó m no período i
P_{ijn}	quantidade de matéria-prima produzida no fornecedor n no período $i - j$ que estará disponível no período i
F_{in}	quantidade de matéria-prima processada pela fábrica n no período i
D^e_{in}	excesso de material descartado por exceder a capacidade de estoque do nó n no período i
D^d_{in}	unidades de produto faltantes para atender à demanda de cliente pelo varejista n no período i

O objetivo do modelo PL é minimizar os custos totais de operação e a função

Tabela 2.4. Parâmetros do modelo PL

Par.	Descrição
q	número de nós da cadeia
h	horizonte de planejamento (tamanho do episódio)
c_n^s	custo de estocar uma unidade de material no nó n
c_n^p	custo de produzir uma unidade de matéria-prima no fornecedor n
c_n^f	custo de processar uma unidade de matéria-prima na fábrica n
c^t	custo de enviar uma unidade de material de um nó para outro
c^e	custo de uma unidade de material descartada por excesso de estoque
c^d	custo por não atendimento à demanda (por unidade de material)
b_n^s	capacidade de estoque do nó n
b_n^p	capacidade de produção do fornecedor n
b_n^f	capacidade de processamento no nó n
b_n^t	quantidade máxima de material que pode ser enviada a partir do nó n
r_n	razão de processamento no nó n
l^{max}	tempo de espera máximo (para produção nos fornecedores e para transporte)
l^{avg}	tempo de espera médio (para produção nos fornecedores e para transporte)
f_n	indica se é possível processar matéria-prima no nó n
p_{ijn}	indica se é possível produzir matéria-prima no nó n no período $i-j$ para estar disponível no período i
t_{ijnm}	indica se é possível enviar material do nó n no período $i-j$ para chegar no nó m no período i
s_n	nível inicial de estoque no nó n
p_{in}	quantidade inicial de material produzida pelo fornecedor n que estará disponível no período i (definido para $i \leq l^{avg}$)
t_{inm}	quantidade inicial de material enviado pelo nó n para o nó m que estará disponível no período i (definido para $i \leq l^{avg}$)
d_{in}	demanda estocástica de cliente a ser atendida pelo nó n no período i

objetivo[4] é dada pela Equação 2.1.

$$\begin{aligned} \min \quad & \sum_{i\in I}\sum_{n\in N}\left(c_n^s S_{in} + c_n^f F_{in} + c^e D_{in}^e + c^d D_{in}^d\right) \\ & + \sum_{i\in I}\sum_{j\in J}\sum_{n\in N}\left(c_n^p P_{ijn} + \sum_{m\in N} c^t T_{ijnm}\right) \end{aligned} \tag{2.1}$$

As restrições são definidas como se segue. As restrições 2.2 controlam o estoque, transporte de material, e as demandas. As capacidades são tratadas pelas restrições

[4]Em relação aos resultados apresentados nas seções de experimentos, os custos relacionados aos materiais iniciais (estoque, fornecimento e transporte) não são considerados na função objetivo.

2.3, 2.4, 2.5 e 2.6. As restrições 2.7 são usadas para calcular a quantidade de matéria-prima processada nas fábricas. E, finalmente, as restrições 2.8, 2.9 e 2.10 são usadas para se levar em conta os materiais iniciais de fornecimento, transporte e estoque.

$$S_{in} = S_{(i-1)n} + \sum_{j \in J} P_{ijn} + \sum_{j \in J} \sum_{m \in N} T_{ijmn} - D^{e}_{in} - r_n \Big(\sum_{j \in J} \sum_{m \in N} T_{(i+j)jnm} \Big) - d_{in} + D^{d}_{in} \quad \forall \quad i \in \{1, ..., h\}, n \in N \tag{2.2}$$

$$0 \leq P_{ijn} \leq p_{ijn} b^{p}_{n} \quad \forall \quad i \in I, j \in J, n \in N \tag{2.3}$$

$$0 \leq F_{in} \leq b^{f}_{n} \quad \forall \quad i \in I, j \in J, n \in N \tag{2.4}$$

$$0 \leq T_{ijnm} \leq t_{ijnm} b^{t}_{n} \quad \forall \quad i \in I, j \in J, n \in N, m \in N \tag{2.5}$$

$$0 \leq S_{(i-1)n} + \sum_{j \in J} P_{ijn} + \sum_{j \in J} \sum_{m \in N} T_{ijmn} - D^{e}_{in} \leq b^{s}_{n} \quad \forall \quad i \in \{1, ..., h\}, n \in N \tag{2.6}$$

$$F_{in} = f_n r_n \Big(\sum_{j \in J} \sum_{m \in N} T_{(i+j)jnm} \Big) \quad \forall \quad i \in \{1, ..., h\}, n \in N \tag{2.7}$$

$$P_{iin} = p_{in} \quad \forall \quad i \in \{1, ..., l^{avg}\}, n \in N \tag{2.8}$$

$$T_{iinm} = t_{inm} \quad \forall \quad i \in \{1, ..., l^{avg}\}, n \in N, m \in N \tag{2.9}$$

$$S_{0n} = s_n \quad \forall \quad n \in N \tag{2.10}$$

Capítulo 3

Aprendizado por Reforço

3.1 Introdução

Aprendizado por Reforço (AR) é uma sub-área de AM, portanto vale a pena apresentarmos primeiro uma ideia geral sobre AM. Segundo Mitchell [1997], podemos dizer que um programa de computador aprende de uma experiência E em relação a uma classe de tarefas T e uma medida de desempenho P se seu desempenho nas tarefas em T, como medido por P, melhora com a experiência E. Um dos exemplos apresentados pelo autor para melhor entendimento da definição se refere a um programa que aprenda a jogar xadrez: neste caso a tarefa T se refere a jogar xadrez, a medida de desempenho P se refere à porcentagem de jogos que o programa vence seus adversários e a experiência E se refere ao programa praticar jogando contra si mesmo. AM tem atualmente diversos usos práticos, e as aplicações aumentam a cada dia. Entre as classes de aplicações, podemos destacar:

- Classificação: atribuir uma categoria a cada item. Ex: classificação de assunto de documentos e reconhecimento de objetos em imagens.
- Regressão: prever um valor real para cada item. Ex: prever variação de variáveis econômicas, valores de estoque e demanda.
- Ranqueamento: ordenar itens de acordo com algum critério: Ex: busca de páginas na Internet.
- Agrupamento: particionar os itens em grupos homogêneos. Ex: identificar comunidades em redes sociais.

É comum a classificação das abordagens de AM em três categorias de aprendizado: supervisionado, não-supervisionado e por reforço.

- No aprendizado supervisionado cada amostra do conjunto de dados possui um rótulo e deseja-se que o modelo forneça o rótulo previsto para os dados desconhecidos. Exemplos de problemas comumente abordados dessa forma são problemas de classificação, regressão e ranqueamento.

- No aprendizado não-supervisionado não existem rótulos nos dados e o interesse está na estrutura dos dados e/ou no relacionamento entre eles. As aplicações mais comuns são de agrupamento. Ex: segmentação de clientes.

- No aprendizado por reforço um agente interage ativamente com um ambiente (e, muitas vezes, o altera) e recebe uma recompensa por cada ação; o objetivo é maximizar as recompensas ao longo das interações. Ex: gestão de recursos em um *cluster* de computadores.

Este trabalho utiliza métodos de AR para resolver um problema de planejamento de produção em uma cadeia de suprimentos multiestágio. AR é um tipo de abordagem de AM que tenta imitar o comportamento de aprendizado dos animais. Uma criança, por exemplo, aprende a realizar várias tarefas através da chamada tentativa e erro, observando os resultados (retornos) do ambiente, até descobrir a melhor forma de lidar com a tarefa. AR usa um mecanismo similar de tentar diferentes ações e receber um retorno (recompensa) que guia o processo de aprendizado até a melhor política possível. Portanto, um agente de AR pode aprender a resolver problemas de tomada de decisão sequencial sob incerteza somente através da experiência, sem a necessidade de humanos rotularem os dados ou coletarem e projetarem uma coleção de dados. A Figura 3.1 apresenta o mecanismo básico de um agente de AR. Um agente (o algoritmo) recebe informação sobre o ambiente (estado) e toma uma ação; o ambiente muda (em parte devido à ação tomada pelo agente), e este processo gera um novo estado e um retorno (recompensa) para o agente. Este processo é repetido e a ideia é maximizar a recompensa esperada em longo prazo. No caso de uma tarefa episódica (como é o problema tratado neste trabalho), a ideia é maximizar a recompensa total até o final do horizonte de planejamento. O treinamento do agente consiste em realizar essa iteração repetidas vezes, começando com uma abordagem de tentativa e erro e, através do processo, o agente aprende a agir para obter melhores recompensas acumuladas.

Agentes de AR precisam lidar com retornos que são sequenciais e avaliativos. Sequenciais porque uma ação tomada em um período pode ter consequências tardias.

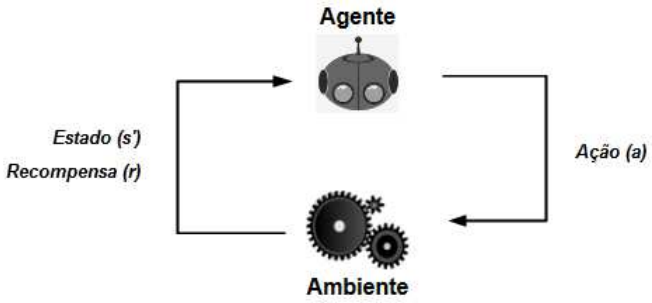

Figura 3.1. O mecanismo básico do Aprendizado por Reforço.

Avaliativos porque, como as recompensas não são supervisionadas, o agente precisa explorar o espaço de busca. AR é diferente do aprendizado supervisionado porque neste há um rótulo que especifica a ação correta que o sistema deve realizar para uma determinada situação, enquanto no AR a recompensa é um retorno para o agente, mas não diz a ele qual seria a ação correta. E AR também não é aprendizado não-supervisionado, uma vez que o agente tenta maximizar um sinal de recompensa em vez de tentar encontrar uma estrutura oculta.

Os métodos de AR podem ser categorizados em *model-based* ou *model-free* [Mazyavkina et al., 2021]. No caso de métodos baseados em modelo, as funções de transição são conhecidas ou podem ser aprendidas, e, portanto, podem ser utilizadas pelo método para tomar suas decisões. Um exemplo de de método desta categoria é o MCTS (*Monte-Carlo Tree Search*). Já os métodos *model-free* são baseados somente na experiência coletada pelo agente e não contam com a disponibilidade da função de transição. Como os métodos utilizados na presente pesquisa são *model-free*, nós focamos este texto nesta categoria de métodos. Dentro da categoria, os métodos podem ainda ser classificados em dois grupos: métodos *baseados em valor* ou métodos *baseados em política*. No primeiro caso, eles aprendem a função de valor para os estados (ou para pares estado-ação) e usam os valores da função para escolher as ações, ou seja, para derivar a política. Já quando são baseados em política, os métodos aprendem uma política parametrizada que permite escolher ações sem a necessidade de consultar estimativas de valores (Sutton & Barto [2018]). Este trabalho utiliza métodos baseados em política, de uma classe específica chamada métodos *Policy Gradient*. Mais detalhes sobre tais métodos são apresentados na Seção 3.4. Há ainda outros métodos baseados em política, geralmente chamados de *direct policy search*, que utilizam estratégias evolutivas, por exemplo. Mas este não é o foco do presente trabalho e, portanto, tais métodos não são tratados no presente texto. Há diferentes comunidades e grupos de pesquisa que trabalham com o problema de AR [Powell, 2019]. Este texto segue a linha

trazida por Sutton & Barto [2018], partindo dos conceitos de PDM e do uso de métodos tabulares como o Q-Learning, para então apresentar os métodos *Policy Gradient*. Mas vale destacar que existem grupos de pesquisa que trabalham em linhas diferentes, como aqueles que partem dos conceitos de Programação Dinâmica Aproximada ou os que trabalham com robótica que, geralmente, partem diretamente de métodos baseados em política.

O presente capítulo é organizado como se segue. Na Seção 3.2 são apresentados os conceitos de PDM que estão intimamente ligados aos conceitos de AR. Os conceitos relacionados aos métodos de AR baseados em valor são apresentados na seção 3.3. Já na seção 3.4 são apresentados tanto os conceitos relacionados aos métodos *Policy Gradient*, quanto os algoritmos utilizados nos experimentos da presente tese. Por fim, na Seção 3.5 é apresentada uma visão geral sobre RNAs e os métodos de gradiente estocásticos geralmente utilizados na atualização dos pesos das mesmas em algoritmos de AR.

3.2 PDM

A formalização de um problema de AR envolve a definição do problema como um PDM. Um PDM é um arcabouço matemático clássico para modelagem de problemas de tomada de decisão sequencial e é apresentado nesta seção conforme definido por Sutton & Barto [2018].

Um PDM é uma tupla formada por $(\mathcal{S}, \mathcal{A}, p, r)$, onde:

- $\mathcal{S}$ é o *conjunto de estados*.
- $\mathcal{A}$ é o *conjunto de ações*.
- $p : \mathcal{S} \times \mathcal{R} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ é a *função de transição* (conjunto de probabilidades condicionais de transição entre os estados).
- $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathcal{R} \subset \mathbb{R}$ é a *função de recompensa*.

No AR, o agente e o ambiente interagem em períodos discretos, $t = 0, 1, 2, 3, \cdots$. Em cada período t, o agente recebe alguma representação do *estado* do ambiente, $S_t \in \mathcal{S}$, e com base nele escolhe uma *ação*, $A_t \in \mathcal{A}(s)$. No período seguinte, em parte como consequência da ação tomada, o agente recebe uma *recompensa* numérica, $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$. Este processo gera uma sequência, ou *trajetória*, que começa com: $S_0, A_0, R_1, S_1, A_1, R_2, A_2, S_3, \cdots$. Em um PDM finito, os conjuntos de estados, ações e recompensas têm um número finito de elementos. Neste caso, as variáveis aleatórias R_t

e S_t têm distribuição de probabilidades discretas bem definidas que dependem apenas do estado e da ação anterior. Ou seja, para algum determinado valor $s' \in \mathcal{S}$ e $r \in \mathcal{R}$, há uma probabilidade destes valores ocorrerem no tempo t, dados valores particulares do estado e da ação anterior, conforme a Equação 3.1.

$$p(s', r|s, a) = Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\} \quad \forall \quad s', s \in \mathcal{S}, r \in \mathcal{R}, a \in \mathcal{A}(s) \tag{3.1}$$

A função p especifica uma distribuição de probabilidades para cada escolha de s e a, de forma a respeitar a Equação 3.2.

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) = 1 \quad \forall\, s \in \mathcal{S}, a \in \mathcal{A}(s) \tag{3.2}$$

No PDM, as probabilidades dadas por p caracterizam a dinâmica do ambiente. Um estado tem a propriedade de *Markov* se a probabilidade de cada valor possível S_t e R_t depende somente do estado e da ação anterior. Isso indica que o estado deve conter toda informação da interação passada ambiente-agente que possa fazer alguma diferença no futuro.

A partir da função p podemos calcular as probabilidades da transição de estados, que podem ser dadas pela Equação 3.3.

$$p(s'|s, a) = Pr\{S_t = s'|S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in \mathcal{R}} p(s', r|s, a) \tag{3.3}$$

A recompensa esperada para um par estado-ação pode ser calculada através da função r, como apresentado na Equação 3.4.

$$r(s, a, s') = \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r \in \mathcal{R}} r \frac{p(s', r|s, a)}{p(s'|s, a)} \tag{3.4}$$

O objetivo do agente é maximizar a recompensa acumulada recebida a longo prazo. Formalmente, o que se procura maximizar é o *retorno esperado*, G_t, definido como uma função específica da sequência de recompensas. Na sua forma mais simples, pode ser definida pela Equação 3.5, onde T é um período final.

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T \tag{3.5}$$

Esta definição é utilizada em aplicações nas quais existe uma noção natural de período final, como é o caso do problema abordado nesta tese. A interação agente-

ambiente, se quebra naturalmente em subsequências, chamadas *episódios* (que, no caso do problema abordado nesta tese, termina ao final do horizonte de planejamento). Cada episódio termina em um estado final, seguido de um reinício para um estado inicial padrão (ou poderia ser uma amostra de uma distribuição padrão de estados iniciais). Neste caso, chamamos a tarefa a ser resolvida de tarefa *episódica*. Existem também problemas com tarefas chamadas *contínuas*. Neste caso, o período final T é infinito e o retorno a ser maximizado pode também se tornar infinito. Assim, a definição de retorno é um pouco diferente, pela inclusão de um *fator de desconto*, γ, tal que $0 \leq \gamma \leq 1$. Desta forma, o retorno se dá como apresentado na Equação 3.6.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{3.6}$$

O valor de γ evita que o retorno a longo prazo seja infinito nas tarefas contínuas. Além disso, ele controla o quanto a agente considera que as recompensas mais próximas são mais valiosas que as recompensas a serem obtidas no futuro e, por isso, pode também ser usado para tarefas episódicas, Podemos facilmente demonstrar que G_t pode ser dado por $G_t = R_{t+1} + \gamma G_{t+1}$.

A maioria dos algoritmos de AR utiliza estimativas de uma *função de valor*, que podem ser funções de estado ou de pares estado-ação que indicam quão bom (em termos de retorno esperado) é para o agente estar em um dado estado, ou tomar uma determinada ação estando em um dado estado. As funções de valor dependem da forma que o agente age, ou seja, da *política*. Uma política é um mapeamento de estados para probabilidades de seleção de cada uma das possíveis ações. Se o agente está seguindo uma política π no período t, então $\pi(a|s)$ é a probabilidade de $A_t = a$ se $S_t = s$. Os algoritmos de AR definem como a política do agente muda como resultado de sua experiência. A *função de valor de estado* (*state-value function*) para um estado s sob a política π, chamada $v_\pi(s)$, é o retorno esperado partindo do estado s e seguindo a política π dali em diante. Para um PDM, a função de valor de estado para a política π pode ser definida pela Equação 3.7, onde $\mathbb{E}_\pi[\cdot]$ indica o valor esperado de uma variável aleatória dado que o agente segue a política π, e t é algum período.

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\Big[\sum_{k=0}^{\infty} R_{t+k+1}\Big|S_t = s\Big], \quad \forall\, s \in \mathcal{S} \tag{3.7}$$

De forma similar, podemos definir a *função de valor de ação* (*action-value function*[1]) para a política π, dada pela Equação 3.8. Ela indica o retorno esperado partindo

[1]Essa função é também chamada na literatura de *Q-function* ou *Q-value*.

do estado s, tomando a ação a, e então seguindo a política π.

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\Big[\sum_{k=0}^{\infty} R_{t+k+1}\Big|S_t = s, A_t = a\Big] \tag{3.8}$$

Nós podemos estimar os valores de v_π e q_π através de experiência, fazendo com que o agente siga a política π, de forma a visitar cada estado uma quantidade de vezes tendendo ao infinito. Fazendo isso, as médias dos retornos para cada estado s encontrado convergem para o valor da função de valor de estado, $v_\pi(s)$. O mesmo raciocínio pode ser aplicado para a função de valor de ação.

Para qualquer política π e qualquer estado s, o valor da função para s e para seus estados sucessores se relacionam conforme descrito na Equação 3.9. A última forma apresentada nesta equação, expressando a relação do valor de um estado com o valor dos seus estados sucessores, é chamada *equação de Bellmann* para v_π. O valor verdadeiro da função v_π é a solução única para sua equação de Bellmann.

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\big[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']\big] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\big[r + \gamma v_\pi(s')\big], \quad \forall\, s \in \mathcal{S} \end{aligned} \tag{3.9}$$

Podemos dizer que uma política π é melhor ou igual a outra política π', ou seja $\pi \geq \pi'$, se e somente se $v_\pi(s) \geq v_{\pi'}(s) \,\forall\, s \in \mathcal{S}$. Sempre há pelo menos uma política que é melhor ou igual a todas as outras e, neste caso, dizemos que ela é uma *política ótima* (π_*). Todas as políticas ótimas têm o mesmo valor ótimo da função de valor de estado, dado por $v_*(s) = max_\pi v_\pi(s) \,\forall\, s \in \mathcal{S}$, e também o mesmo valor ótimo da função de valor de ação $q_*(s,a) = max_\pi q_\pi(s,a) \,\forall\, s \in \mathcal{S}, a \in \mathcal{A}$. A partir dessas definições podemos escrever q_* em termos de v_*, como mostrado na Equação 3.10.

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1})|S_t = s, A_t = a] \tag{3.10}$$

A equação de Bellman para v_*, chamada de *equação de otimalidade de Bellman*, dada pela Equação 3.11, expressa o fato de que o valor de um estado sob uma política

ótima deve ser igual ao retorno esperado da melhor ação a partir daquele estado.

$$\begin{aligned} v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\ &= \max_a \mathbb{E}_{\pi_*}[G_t | S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')] \end{aligned} \tag{3.11}$$

Já a equação de otimalidade de Bellman para a função q_* é dada pela Equação 3.12

$$\begin{aligned} q_*(s,a) &= \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') | S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r|s, a)[r + \gamma \max_{a'} q_*(s', a')] \end{aligned} \tag{3.12}$$

As equações de otimalidade para a função v_* são na verdade um sistema de equações com n equações e n incógnitas, sendo n o número de estados. Se a dinâmica p do ambiente é conhecida, em princípio é possível resolver o sistema de equações usando métodos de solução de sistemas de equações não-lineares. Uma vez encontrado v_*, é fácil determinar uma política ótima, seguindo uma estratégia gulosa para determinar a ação em cada estado. Ou seja, basta selecionar a ação que leva a um novo estado com o melhor valor da função de valor. A mesma ideia de estratégia gulosa pode ser seguida para a função q_*. Uma vez encontrada uma função ótima, o agente precisa simplesmente escolher a ação que maximiza $q_*(s,a)$. A vantagem de usar q_* ao invés de v_* é poder determinar ações ótimas sem precisar saber nada sobre seus estados sucessores ou seus valores, ou seja, sem ter que saber nada sobre a dinâmica do ambiente.

Depois do que foi apresentado, podemos dizer que resolver as equações de otimalidade de Bellman é um caminho para encontrar uma política ótima e assim resolver o problema de AR. Isso pode ser feito, por exemplo, utilizando *Programação Dinâmica* para, iterativamente, avaliar a função de valor e melhorar a política, um padrão conhecido como Generalized Policy Iteration (GPI). A fase de avaliação da política calcula os valores para uma dada política, resolvendo o que é chamado problema de predição, enquanto a fase de melhoria da política usa as estimativas da política atual para, agindo de forma gulosa, encontrar uma nova (e melhor) política. A alternância entre avaliação da política e a melhoria da política resolvem o chamado problema de controle, ou seja,

progressivamente geram políticas melhores em direção à otimalidade. Mas esta solução se baseia em três premissas que raramente acontecem juntas na prática: precisamos conhecer precisamente a dinâmica do ambiente, precisamos de poder computacional para calcular a solução e precisamos da propriedade de Markov. Portanto, muitos dos algoritmos de AR podem ser vistos como métodos para resolver as equações de Bellman de forma aproximada, usando transições experienciadas pelo agente no lugar das transições esperadas.

3.3 Métodos Baseados em Valor

Métodos de AR *baseados em valor* têm como foco encontrar uma política que dê o melhor retorno esperado utilizando uma função de valor de estado, $v_\pi(s)$, ou uma função de valor de ação, $q_\pi(s, a)$. A seguir apresentamos as principais ideias desse tipo de método com base em Sutton & Barto [2018].

Uma das possíveis abordagens baseadas em valor é dada pelos chamados métodos de **Monte-Carlo**. A ideia é simples, o agente interage com o ambiente por episódios completos e utiliza a média do retorno observado como uma aproximação do retorno esperado. O retorno empírico é dado por $G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$ e, portanto, o método considera que as tarefas são episódicas. Para aprender, o método segue a mesma ideia do GPI da Programação Dinâmica. Uma possível implementação do método considera que, a partir de alguma política inicial, os seguintes passos são repetidos: (1) a política é melhorada de forma gulosa em relação à função de valor de ação atual, ou seja, $q_\pi(s) = arg\,max_{a \in \mathcal{A}} Q(s, a)$, (2) um novo episódio é gerado com a nova política π e (3) a estimativa da função q é atualizada para cada par estado-ação considerando o retorno obtido.

Na atualização da política feita no primeiro passo, geralmente é utilizado algum método para tratar o *exploration vs. exploitation trade-off*, ou seja, o compromisso apropriado entre a exploração do espaço de busca, ou coleta de informações (*exploration)*, e a exploração das informações já obtidas (*exploitation*). Se a atualização for sempre gulosa, ou seja, se o método sempre explorar as informações já obtidas, ele pode ficar preso em um ótimo local. Por outro lado, se o método sempre coletar novas informações, ele pode demorar muito para convergir. Uma das formas de lidar com essa questão é através da abordagem ϵ-*greedy*. Com ela, a melhor ação é escolhida na maior parte do tempo, com probabilidade $1-\epsilon$, mas ocasionalmente uma ação aleatória é escolhida, com probabilidade ϵ. Os valores de ϵ são geralmente baixos ($\epsilon = 0,1$, por exemplo) de forma a manter um compromisso razoável. Outra possível abordagem é

iniciar ϵ com um determinado valor e, ao longo o treinamento, diminuir aos poucos o valor utilizado, permitindo uma maior exploração do espaço de busca no início e uma maior exploração das informações obtidas no final do treinamento.

Segundo Morales [2020], métodos de Monte-Carlo têm boas propriedades de convergência, por atualizar as estimativas da função de valor (V_t ou Q_t) em direção ao retorno real G_t, que é uma estimativa não-enviesada a função de valor real (v_π ou q_π). Por outro lado, os retornos reais são estimativas de alta variância da função de valor real. Isso acontece devido ao acúmulo de ruído dos eventos aleatórios nos retornos reais (das ações, estados e recompensas, todas variáveis aleatórias). Por isso, o método é ineficiente em relação às amostras, já que é necessário coletar muitos dados para diminuir o efeito do ruído.

Existem outros métodos baseados em valor que são chamados **métodos TD** (de *Temporal Difference*). Tais métodos também aprendem por experiência, mas não dependem de episódios completos para atualizar as estimativas da função de valor. Eles utilizam o conceito de *bootstrapping*, o que significa que as estimativas para um estado (ou par estado-ação) são feitas com base nas estimativas existentes, ao invés de se utilizar apenas as recompensas reais e retornos completos como nos métodos de Monte-Carlo. Isso permite que a atualização seja feita a cada passo, ao custo de incluir um viés em direção à forma como o modelo foi inicializado. A ideia central é atualizar as estimativas V da função de valor de estado v_π em direção a um retorno estimado $R_{t+1} + \gamma V(S_{t+1})$, chamado de alvo-TD (*TD-target*). Note que o alvo-TD é uma estimativa do retorno, já que ele considera a soma da recompensa real R_{t+1} e a estimativa de retorno a partir do próximo estado $\gamma V(S_{t+1})$. O ganho é que podemos agora atualizar as estimativas a cada período, ao invés de fazer uma atualização apenas ao final de cada episódio. Usamos uma taxa de aprendizado α, um hiperparâmetro, para controlar o quanto queremos atualizar a função, conforme apresentado na Equação 3.13. Seguindo uma ideia similar, um método TD pode se basear em estimativas Q da função de valor de ação q_π, como apresentado na Equação 3.14.

$$\begin{aligned} V(S_t) &= (1-\alpha)V(S_t) + \alpha G_t \\ &= V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)) \end{aligned} \tag{3.13}$$

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)) \tag{3.14}$$

A atualização das estimativas das funções de valor podem ser feitas de forma *on-policy* ou *off-policy*. Métodos *on-policy* tentam avaliar e melhorar a política que está

sendo usada para tomar decisões, enquanto métodos *off-policy* avaliam e melhoram um polítca diferente daquela usada para gerar os dados. O algoritmo ***SARSA*** é um método TD *on-policy*, que segue a mesma ideia do GPI e faz atualizações das estimativas Q conforme apresentado pela Equação 3.14. Note que para atualizar o valor da ação atual A_t, é necessário conhecer a próxima ação A_{t+1}, de acordo com a política atual.

Já o método ***Q-Learning***, é um método TD *off-policy* que é muito similar ao SARSA, mas cuja atualização das estimativas Q é dada pela Equação 3.15. Neste caso, o método não segue a política atual para obter a próxima ação. Ao invés disso ele considera o valor da melhor ação para o estado atual, mesmo que o algoritmo não tome esta ação no próximo passo.

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t)) \tag{3.15}$$

O alvo-TD é uma estimativa enviesada da função de valor real (v_π ou q_π), já que ele usa uma estimativa da função de valor para calcular uma nova estimativa da função de valor. Por outro lado, é uma estimativa com variância muito menor do que o retorno real G_t usado nas atualizações de Monte-Carlo. Isso acontece porque o alvo-TD depende de uma única ação, uma única transição e uma única recompensa e, logo, tem menor aleatoriedade acumulada. O resultado é que os métodos TD geralmente aprendem bem mais rápido que os métodos de Monte-Carlo [Morales, 2020].

Existe uma categoria de métodos entre os métodos de Monte-Carlo e os métodos TD apresentados, chamados **métodos TD de n-passos**. Eles têm por objetivo tratar o compromisso entre viés e variância, controlando o quanto de *bootstrapping* é usado. A ideia é seguir a mesma estratégia apresentada para os métodos TD mas, ao invés de fazer a atualização das estimativas da função de valor a cada passo, essa atualização é feita a cada n passos, como apresentado na Equação 3.16. Tais métodos podem ser vistos como uma generalização dos métodos anteriores, uma vez que se n for igual a um, obtemos os mesmos métodos TD já apresentados (e que, justamente por isso, costumam também ser chamados de métodos TD de um passo), já se n for muito grande, maior ou igual ao tamanho do episódio, acabamos obtendo o método de Monte Carlo.

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{v}(S_{t+n}), 0 \leq t \leq T - n \tag{3.16}$$

Há dois pontos importantes sobre os métodos TD de n-passos: as atualizações das estimativas são feitas somente depois de realizados n passos de coleta de dados, e é necessário definir o valor de n. Existe um tipo de método ainda mais geral, que resolve esses pontos, permitindo atualizações a cada passo, chamados **métodos TD(λ)**. A ideia é, ao invés de se definir o valor de n, utilizamos uma média ponderada dos retornos de n-passos para diferentes valores de n. O algoritmo TD(λ) tem todos os retornos de n-passos, cada um ponderado proporcionalmente a λ^{n-1}, com $\lambda \in [0, 1]$, e normalizado por um fator $1 - \lambda$ para garantir que os pesos somem um. O retorno de um passo tem o maior peso $1 - \lambda$, o segundo retorno tem peso $(1 - \lambda)\lambda$, e assim sucessivamente, cada passo decaindo por λ. Depois do estado terminal, todos os passos seguintes têm o retorno convencional G_t. A atualização resultante é em direção a um retorno chamado retorno-λ, definido como apresentado na Equação 3.17.

$$G_t^{\lambda} = (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} \lambda^{T-t-1} G_t \tag{3.17}$$

Vale notar que se $\lambda = 1$, o primeiro termo da Equação 3.17 se torna zero e o que sobra é retorno convencional G_t. Portanto, para $\lambda = 1$ temos o retorno do algoritmo de Monte Carlo. Já se $\lambda = 0$, obtemos o retorno de um passo $G_{t:t+1}$ dos métodos TD. Na prática, λ é um hiperparâmetro do algoritmo que precisa ser ajustado para o problema em questão. Maiores detalhes sobre tais métodos, como estratégia de implementação usando *eligibility traces*, podem ser encontrados em Sutton & Barto [2018].

3.3.1 Métodos de Solução Aproximada

Os métodos baseados em valor apresentados utilizam retorno exaustivo, ou seja, os agentes precisam ter acesso a todas as amostras possíveis (todos os estados e todas as ações) e, inclusive, acessá-los várias vezes para se obter uma boa estimativa da função de valor. Mas muitos problemas têm espaços de estados ou de ações com muitas dimensões (ou eles podem ser espaços contínuos), e, em tais casos, não é possível lidar com uma tabela para a função de valor (valores de estado ou de pares estado-ação). O problema não é apenas com as restrições de tempo e armazenamento da tabela, mas, principalmente, porque muitos estados provavelmente nunca serão visitados durante o processo de aprendizado. Assim, é necessário usar métodos que possam lidar com retornos amostrados e generalizar a partir de estados similares.

Uma possível abordagem para tratar essa questão é usar *métodos de solução aproximada*, lançando mão de funções paramétricas v_ϕ ou q_ϕ, ao invés de tabelas, para estimar o valor real das funções de valor. Os pesos (parâmetros) da função aproximada

podem ser dados por diferentes formas, como uma combinação linear dos pesos de cada *feature* ou com pesos de uma RNA, por exemplo. Geralmente a dimensionalidade dos pesos é muito menor que o número de estados. Assim, quando os pesos da função são atualizados para um estado (ou par estado-ação), a atualização também impacta os valores para outros estados (ou pares estado-ação), o que nos permite generalizar para estados (ou pares estado-ação) não observados. Muitos dos métodos aproximados seguem o mesmo mecanismo iterativo básico dos métodos tabulares, mas usando uma função paramétrica ao invés de uma tabela. O algoritmo DQN, por exemplo, é uma abordagem aproximada baseada no *Q-Learning*.

Mas este tipo de abordagem é limitada para problemas com espaços de ação contínuos devido à necessidade de se calcular o valor máximo sobre as ações. Como o problema tratado neste trabalho tem espaços de ação e de estado contínuos, não aprofundamos neste texto nos métodos aproximados baseados em valor. Ao invés disso, damos maior ênfase aos métodos baseados em política, que podem lidar melhor com este tipo de questão.

3.4 Métodos *Policy Gradient*

Os métodos de AR *baseados em política* aprendem uma política parametrizada, que seleciona ações sem a necessidade de consultar uma função de valor. Há métodos que até utilizam uma função de valor para aprender a política, mas ela não é necessária para a escolha da ação. Assim como nas seções anteriores, os conceitos apresentados a seguir se baseiam principalmente em Sutton & Barto [2018].

Sendo $\theta \in R^d$ o vetor d-dimensional de parâmetros de uma política parametrizada, esta pode ser dada por $\pi(a|s,\theta) = \pi_\theta(a|s) = Pr\{A_t = a|S_t = s, \theta_t = \theta\}$. Ou seja, ela define a probabilidade da ação a ser tomada no período t dado que o ambiente se encontra no estado s no período t e os parâmetros da política são θ. Esta seção apresenta um tipo de métodos baseados em política, chamados *Policy Gradient*. Tais métodos usam o gradiente de alguma medida escalar de desempenho $J(\theta)$ em relação aos parâmetros da política. Como o objetivo é maximizar o desempenho, suas atualizações aproximam o gradiente (*ascent*) em J, dadas pela Equação 3.18, onde $\widehat{\nabla J(\theta_t)}$ é uma estimativa estocástica cuja expectativa aproxima o gradiente da medida de desempenho em relação a seu argumento θ_t.

$$\theta_{t+1} = \theta_t + \alpha\widehat{\nabla J(\theta_t)} \tag{3.18}$$

Vale notar que, em termos práticos, os métodos *Policy Gradient* caminham no

sentido de aumentar a probabilidade das ações que levam a um maior retorno e a diminuir a probabilidade daquelas que trazem um retorno menor [Achiam, 2018]. Tais métodos podem parametrizar políticas de quaisquer formas desde que $\pi_\theta(a|s)$ seja diferenciável em relação a seus parâmetros, ou seja, $\nabla\pi_\theta(a|s)$, o vetor coluna das derivadas parciais de $\pi_\theta(a|s)$ em relação aos componentes de θ, exista e seja finito para todo estado, ação e parâmetro θ. Para garantir a generalização, usualmente definimos que a política nunca pode se tornar determinística, ou seja, $\pi_\theta(a|s) \in (0, 1)$.

Na descrição dos métodos *Policy Gradient* é comum o uso do termo *loss function* para se referir à função de custo utilizada pelos algoritmos. Mas, segundo Achiam [2018], é importante notar que tal função de custo não tem o mesmo sentido utilizado em aprendizado supervisionado, por duas razões. A primeira é que nos métodos *Policy Gradient* a distribuição dos dados depende dos parâmetros que estão sendo otimizados, diferente do aprendizado supervisionado no qual o conjunto de dados é fixo e independente dos parâmetros a serem otimizados. A segunda razão é que a função de custo não mede realmente o desempenho. O objetivo dos métodos *Policy Gradient* é maximizar o retorno esperado $J(\theta)$, mas a função de custo não aproxima este valor. Ela é útil na verdade porque, quando avaliada com os parâmetros atuais, com os dados gerados por tais parâmetros, seu valor é o negativo do gradiente do desempenho. Mas a minimização da função de custo em si não garante que o retorno esperado será maximizado. Em termos práticos isso é importante porque nos métodos *Policy Gradient* a diminuição do valor função de custo não é um indicador da qualidade dos resultados e, portanto, deve-se utilizar outras medidas, como o retorno médio, para avaliar a qualidade dos resultados obtidos por tais métodos.

Nas próximas seções apresentamos algumas formas comuns de parametrização da política, o Teorema *Policy Gradient* e o algoritmo REINFORCE. Em seguida, apresentamos o conceito de métodos Ator-Crítico e de Função de Vantagem. E, por fim, apresentamos os algoritmos A2C, DDPG, PPO, SAC e TD3, utilizados nos experimentos da presente tese, e um sumário de comparação entre os algoritmos.

3.4.1 Formas Comuns de Parametrização

Nesta seção apresentamos formas comuns de parametrização de políticas usadas em métodos *Policy Gradient*, para espaços de ação discretos e contínuos, conforme apresentado por Sutton & Barto [2018].

3.4.1.1 Para espaço de ações discreto

Se o espaço de ações é discreto e não muito grande, uma parametrização comum e natural é formar preferências numéricas parametrizadas $h_\theta(s,a) \in \mathbb{R}$ para cada par estado-ação, nas quais, as ações com maior preferência têm mais probabilidade de ser selecionadas. Isto pode ser feito, por exemplo, usando a distribuição exponencial *soft-max*, dada pela Equação 3.19, onde o denominador garante que a soma de probabilidades em cada estado seja um.

$$\pi_\theta(a|s) = \frac{e^{h_\theta(s,a)}}{\sum_b e^{h_\theta(s,b)}} \tag{3.19}$$

Políticas parametrizadas com preferência de ações *soft-max* têm vantagens importantes sobre as políticas ϵ-*greedy*, dos algoritmos baseados em valor. Com a abordagem ϵ-*greedy* a melhor ação tem probabilidade muito alta, enquanto as demais (mesmo que próximas da melhor em preferência) dividem a mesma pequena probabilidade ϵ. Já na abordagem *soft-max* a segunda ação em preferência, terá probabilidade similar à primeira, se tiver valor próximo a ela. Além disso, uma ação com preferência muito ruim, terá probabilidade bem mais baixa no *soft-max* que as demais, ou seja, *soft-max* distribui melhor as probabilidades entre as ações. As preferências em si podem ser parametrizadas de qualquer forma, podendo ser simples combinações lineares das *features*, ou os pesos de todas as conexões de uma RNA, por exemplo.

Uma outra diferença da abordagem *soft-max* é que a política aproximada se aproxima de uma política determinística. À medida que as preferências das ações se aproximam de valores específicos, a política se torna gulosa automaticamente, tendendo a se tornar políticas estocásticas ótimas, com os valores se aproximando dos valores verdadeiros que são sempre diferentes de 0 e 1 por uma pequena quantidade finita. Se o ótimo é determinístico, a preferência das ações ótimas será infinitamente maior que as sub-ótimas (se a parametrização permitir). Ao passo que, na abordagem ϵ-greedy há sempre uma probabilidade ϵ de uma escolha aleatória. Por fim, a abordagem *soft-max* permite a seleção de ações com probabilidade arbitrária. Em muitos problemas as melhores políticas aproximadas podem ser estocásticas, mas os métodos baseados em valor não têm uma forma natural de encontrar políticas estocásticas ótimas como métodos baseados em políticas aproximadas têm.

3.4.1.2 Para espaço de ações contínuo

É impraticável usar função de preferência de ações para definir as probabilidades das ações de grandes espaços de ação, ou de espaços de ação contínuos (já que, neste caso,

as ações são infinitas). Uma alternativa é aprender estatísticas da distribuição de probabilidades dessas preferências. Isso pode ser feito definindo a política como uma função densidade de probabilidade de uma distribuição Normal sobre um valor escalar real da ação, com a média e o desvio padrão dados pelos aproximadores da função paramétrica que depende do estado. Podemos definir a política conforme apresentado na Equação 3.20, onde $\mu : \mathcal{S} \times \mathbb{R}^{d'} \rightarrow \mathbb{R}$ e $\sigma : \mathcal{S} \times \mathbb{R}^{d'} \rightarrow \mathbb{R}^{+}$ são dois aproximadores de função parametrizados (e π do lado direito da equação é a constante matemática).

$$\pi_\theta(a|s) = \frac{1}{\sigma_\theta(s)\sqrt{2\pi}} exp\Big(- \frac{(a - \mu_\theta(s))^2}{2\sigma_\theta(s)^2} \Big) \tag{3.20}$$

Os aproximadores da política podem ser dados por diversas técnicas, como uma combinação linear das *features* ou pelos pesos de uma RNA. A Figura 3.2 apresenta uma ideia esquemática de um possível método *Policy Gradient* que usa uma RNA para aproximar a política, $\pi_\theta(a|s)$, para um problema com espaços de estados e ações contínuos (como é problema tratado nesta tese). O estado s é representado por um vetor de valores contínuos, e a camada de entrada da RNA tem um nó para cada dimensão do estado. A camada de saída da RNA consiste em um nó para cada dimensão de ação e fornece valores médios $\mu_\theta(s)$ para as ações. Neste método de exemplo, o agente teria um vetor σ_θ com os valores de desvio padrão para cada valor de ação (independente do estado). A ação $a(s)$, retornada pelo agente, seria dada então por $a(s) = \mu_\theta(s) + \sigma_\theta(s) \odot z$, onde $z \sim \mathcal{N}(0, 1)$ e $\odot$ representa o produto elemento a elemento de dois vetores [Achiam, 2018]. Em relação ao vetor σ_θ, ele pode ser usado para controlar a exploração do algoritmo. No começo do treinamento, os valores do vetor são maiores permitindo uma maior exploração, e durante o processo de aprendizado, os valores são diminuídos lentamente para tirar melhor proveito do conhecimento já adquirido pelo agente. Uma outra possível abordagem é ter uma rede neural separada para aprender os valores de desvio padrão. Neste caso, o vetor é dado por $\sigma_\theta(s)$, já que ele depende do estado. Os pesos das RNAs (θ) podem ser atualizados usando métodos de Gradiente Estocástico, usando a recompensa recebida pelo agente.

As políticas encontradas por essa forma de aproximação, são chamadas políticas Gaussianas diagonais [Achiam, 2018]. Isto porque uma distribuição Normal (Gaussiana) multivariada é descrita por um vetor de médias, μ, e uma matriz de covariância, Σ. Uma distribuição Gaussiana diagonal é um caso especial no qual a matriz de covariância tem valores não-nulos apenas na diagonal, e que, portanto, pode ser representada por um vetor. Vale notar que nos métodos *Policy Gradient* é muito comum encontrarmos a expressão de log-probabilidade $log\, \pi_\theta(a|s)$. Para uma ação a de k dimensões, a log-probabilidade de uma Gaussiana diagonal com $\mu = \mu_\theta(s)$ e desvio padrão $\sigma = \sigma_\theta(s)$, é

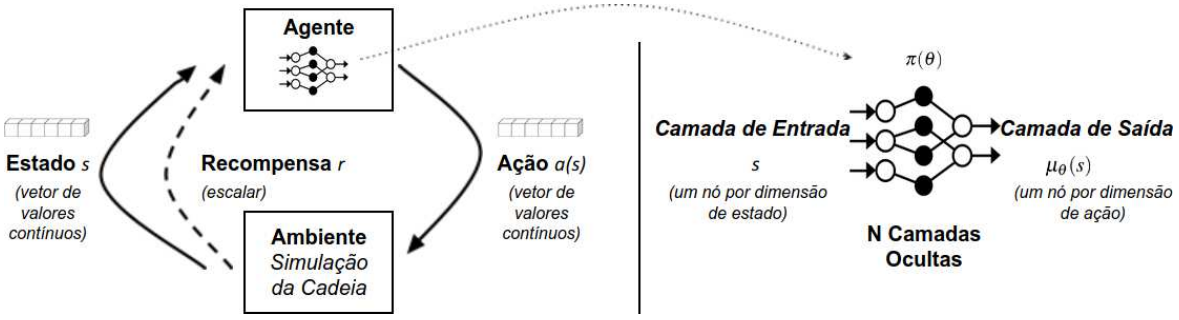

Figura 3.2. Representação esquemática de um possível método *Policy Gradient* simples que aprende políticas estocásticas para problemas com espaços de estados e de ações contínuos. A política é parametrizada por uma RNA que recebe os valores do estado na camada de entrada, e tem como saída os valores médios para cada valor de ação. Os valores de ação realmente retornados são amostrados de uma distribuição Normal usando os valores médios da saída da RNA e os desvios padrões de um vetor usado para controlar a exploração do algoritmo.

dada pela Equação 3.21 (onde π do lado direito da equação é a constante matemática).

$$log\,\pi_\theta(a|s) = -\frac{1}{2}\left(\sum_{i=1}^{k}\left(\frac{(a_i-\mu_i)^2}{\sigma_i^2} + 2\,log\,\sigma_i\right) + k\,log\,2\pi\right) \tag{3.21}$$

Mesmo para problemas com espaço de ações discreto, se as ações forem muitas, pode valer a pena trabalhar também dessa forma, já que o uso da distribuição normal fornece uma forma natural de se obter exploração.

3.4.2 O Teorema *Policy Gradient*

Nas seções anteriores foram apresentadas vantagens práticas de usar uma política parametrizada ao invés de métodos baseados em valor com seleção de ações usando ϵ-greedy. Nesta seção, veremos que políticas parametrizadas têm também uma vantagem teórica importante. Quando usamos uma política parametrizada contínua as probabilidades das ações mudam suavemente como uma função dos parâmetros aprendidos. Já quando usamos ϵ-greedy as probabilidades das ações podem mudar de forma abrupta, mesmo com uma mudança pequena nos valores das ações (quando leva a uma troca da melhor ação). Devido a isso, existem garantias de convergência mais fortes para métodos baseados em política do que para métodos baseados em valor [Sutton & Barto, 2018].

Para tarefas episódicas, como o problema tratado neste trabalho, podemos definir o desempenho como $J(\theta) = v_{\pi_\theta}(s_0)$, onde v_{π_θ} é o valor verdadeiro da função para a política parametrizada π_θ e s_0 é o estado inicial (determinístico). O desafio aqui é como atualizar os parâmetros da política de forma a obter uma melhoria da mesma.

Isso ocorre pelo fato do desempenho depender tanto da seleção das ações, quanto da distribuição dos estados nos quais as seleções são feitas, já que ambos são afetados pelos parâmetros da política. Dado um estado, podemos calcular de forma razoavelmente direta o efeito dos parâmetros conhecidos da política nas ações e, consequentemente, nas recompensas. Mas o efeito da política na distribuição dos estados é uma função do ambiente, e em métodos *model-free* tal distribuição não é conhecida, o que inviabilizaria estimar o gradiente do desempenho em relação aos parâmetros da política.

É nesse contexto que se insere a importância do *Teorema Policy Gradient*. Ele fornece uma expressão analítica para o gradiente do desempenho com relação aos parâmetros da política (que é justamente o que precisamos para aproximar o gradiente, Equação 3.18) que não envolve a derivada da distribuição dos estados. A Equação 3.22 apresenta o resultado do teorema, onde os gradientes são vetores-coluna das derivadas parciais em relação aos componentes θ, π é a política correspondente ao parâmetro θ, $\propto$ significa proporcional a, o que no caso episódico significa proporcional ao tamanho médio dos episódios, e μ é a distribuição de estados *on-policy* sob π. Para o leitor interessado, a prova do teorema, para o caso episódico, pode ser encontrada em Sutton & Barto [2018].

$$\nabla J(\theta) \propto \sum_{s} \mu(s) \sum_{a} q_{\pi}(s,a) \nabla \pi_{\theta}(a \mid s) \tag{3.22}$$

3.4.3 O Método REINFORCE

A partir do teorema apresentado na seção anterior, podemos derivar um primeiro método *Policy Gradient*. Para isso, precisamos obter amostras de tal forma que a expectativa do gradiente amostrado seja proporcional ao gradiente real da medida de desempenho como uma função dos parâmetros de aproximação. Basta ser proporcional porque é utilizado um parâmetro α que acaba por absorver a constante de proporção. Como o teorema já dá uma forma exata para o gradiente, precisamos apenas de uma forma de amostragem que aproxime essa expressão. Se a política π for seguida, podemos substituir a soma das variáveis aleatórias referentes aos estados pela expectativa sob a política π, chegando à Equação 3.23. A partir dessa definição, poderíamos definir um algoritmo de gradiente estocástico (*ascent*), como apresentado na Equação 3.24,

onde $\hat{q}$ é uma aproximação aprendida de q_π, parametrizada por ϕ.

$$\begin{aligned}\nabla J(\theta) &\propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi_\theta(a \mid s) \\ &= \mathbb{E}_\pi \Big[\sum_a q_\pi(S_t, a) \nabla \pi_\theta(a \mid S_t) \Big]\end{aligned} \tag{3.23}$$

$$\theta_{t+1} = \theta_t + \alpha \sum_a \hat{q_\pi}(S_t, a, \phi) \nabla \pi_\theta(a \mid S_t) \tag{3.24}$$

Um problema desta abordagem é que ela depende de todas as ações a partir do estado atual. Para tratar isso, a mesma ideia de substituir a soma das variáveis aleatórias por uma expectativa sob a política π pode também ser feita para as ações. Como demonstrado em Sutton & Barto [2018], ao fazer isso, obtemos a Equação 3.25, onde G_t é o retorno já mencionado anteriormente. A quantidade definida entre colchetes pode ser amostrada a cada período e sua expectativa é igual à do gradiente. A partir disso, podemos definir um método de gradiente estocástico genérico, cuja atualização, mostrada na Equação 3.26, define o algoritmo **REINFORCE**.

$$\nabla J(\theta) = \mathbb{E}_\pi \Big[G_t \frac{\nabla \pi_\theta(A_t \mid S_t)}{\pi_\theta(A_t \mid S_t)} \Big] \tag{3.25}$$

$$\theta_{t+1} = \theta_t + \alpha G_t \frac{\nabla \pi_\theta(A_t \mid S_t)}{\pi_\theta(A_t \mid S_t)} \tag{3.26}$$

Intuitivamente, podemos notar que o incremento é proporcional ao produto de um retorno G_t e um vetor definido por uma fração. Tal vetor é a direção no espaço de parâmetros que mais aumenta a probabilidade de repetir a ação A_t em uma futura visita ao estado S_t. A atualização aumenta o vetor de parâmetros nesta direção, proporcionalmente ao retorno (para caminhar no sentido das ações de maior retorno) e inversamente proporcional à probabilidade da ação (para evitar que ações mais visitadas levem vantagem).

O pseudocódigo do método REINFORCE é apresentado no Algoritmo 1, e vale observar que ele inclui também o fator de desconto γ. O método usa o retorno completo a partir do tempo t, o que exige calcular a recompensa até o final do episódio, sendo portanto um método de Monte-Carlo. Outro detalhe é que ele usa a expressão $\nabla ln \pi_\theta(A_t \mid S_t)$ que é o mesmo que a fração $\frac{\nabla \pi_\theta(A_t|S_t)}{\pi_\theta(A_t|S_t)}$, devido à identidade $\nabla lnx = \frac{\nabla x}{x}$. O REINFORCE possui boas propriedades de convergência, podendo convergir para um ótimo local sob condições apropriadas. Mas, sendo um método de Monte-Carlo sofre com alta variância, o que torna o aprendizado lento.

Algoritmo 1: REINFORCE

Inicializar parâmetros θ (da política)
for *cada episódio* **do**
    Gera um episódio completo, seguindo $\pi_\theta(\cdot|\cdot)$
    for *cada passo t dado no episódio* **do**
        $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
        $\theta \leftarrow \theta + \alpha \gamma^t G \nabla ln \pi_\theta(A_t \mid S_t)$

Para diminuir a variância do método, podemos usar um *baseline* arbitrário $b(s)$, como mostrado na Equação 3.27, desde que $b(s)$ não varie com a ação (já que, desta forma, pode-se demonstrar algebricamente que ele não altera a expressão).

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a \big(q_\pi(s,a) - b(s)\big) \nabla \pi_\theta(a \mid s) \tag{3.27}$$

Com esta alteração, podemos modificar a regra de atualização do algoritmo, como apresentado na Equação 3.28, e assim obter uma versão mais geral do REINFORCE que inclui o *baseline*.

$$\theta_{t+1} = \theta_t + \alpha(G_t - b(S_t)) \frac{\nabla \pi_\theta(A_t \mid S_t}{\pi_\theta(A_t \mid S_t)} \tag{3.28}$$

Uma escolha natural para o *baseline* é uma aproximação da função de valor de estado $\hat{v}_\phi(S_t)$ onde $\phi \in \mathbb{R}^d$ é aprendido por algum método. O Algoritmo 2 apresenta o pseudocódigo do método **REINFORCE com Baseline**.

Algoritmo 2: REINFORCE com Baseline

Inicializar parâmetros θ (da política) e ϕ (da função de valor de estado)
for *cada episódio* **do**
    Gera um episódio completo, seguindo $\pi_\theta(\cdot|\cdot)$
    for *cada passo t dado no episódio* **do**
        $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
        $\delta \leftarrow G - \hat{v}_\phi(S_t)$
        $\phi \leftarrow \phi + \alpha^\phi \, \delta \nabla \hat{v}_\phi(S_t)$
        $\theta \leftarrow \theta + \alpha^\theta \, \gamma^t \, \delta \nabla ln \, \pi_\theta(A_t \mid S_t)$

3.4.4 Métodos Ator-Crítico

Alguns dos métodos *Policy Gradient* são chamados métodos Ator-Crítico. Tais métodos possuem um ator que aprende a política e um crítico que aprende uma função de

valor com o objetivo de avaliar a política aprendida. Segundo Sutton & Barto [2018], o REINFORCE com baseline, apresentado na seção anterior, não é considerado um método Ator-Crítico uma vez que, apesar de aprender a política e a função de valor de estado, esta não é usada como crítico. Em outras palavras, para que um método seja considerado Ator-Crítico, é necessário que o crítico inclua o viés de *bootstrapping*, como nos métodos TD. Este viés é geralmente benéfico porque reduz a variância, o que acelera o aprendizado. O REINFORCE, mesmo com baseline, é não-enviesado e converge assintoticamente para um ótimo local, mas, como qualquer método de Monte-Carlo, tende a aprender lentamente por gerar estimativas com alta variância, não sendo viável para implementação *online* ou em problemas contínuos. Métodos Ator-Crítico podem fazer atualizações a cada passo, similares ao Q-Learning e, desta forma, são totalmente *online* e incrementais. Por essas características, eles podem ser aplicados também a tarefas contínuas.

Um método Ator-Crítico de um passo pode ser definido de forma análoga aos métodos TD, como o Q-Learning ou o SARSA. O método troca o retorno completo do REINFORCE pelo retorno de um passo, além de usar uma função de valor de estado aprendida como baseline, como apresentado na Equação 3.29. O pseudocódigo do método é apresentado no Algoritmo 3.

$$\begin{aligned}\theta_{t+1} &= \theta_t + \alpha(G_{t+1} - \hat{v}_\phi(S_t))\frac{\nabla \pi_{\theta_t}(A_t \mid S_t)}{\pi_{\theta_t}(A_t \mid S_t)} \\ &= \theta_t + \alpha(R_{t+1} + \gamma\hat{v}_\phi(S_{t+1}) - \hat{v}_\phi(S_t))\nabla ln\, \pi_{\theta_t}(A_t \mid S_t) \\ &= \theta_t + \alpha\, \delta\, \nabla ln\, \pi_{\theta_t}(A_t \mid S_t)\end{aligned} \tag{3.29}$$

3.4.5 Função de Vantagem

Os passos de atualização dos algoritmos *Policy Gradient* apresentados nas seções anteriores podem ser dados por uma forma geral representada pela Equação 3.30, onde ψ_t pode assumir diferentes formas:

1. $\psi_t = G_t$ no caso do método REINFORCE,
2. $\psi_t = G_t - \hat{v}_\phi(S_t)$ no caso do REINFORCE com baseline,
3. e $\psi_t = R_{t+1} + \gamma\hat{v}_\phi(S_{t+1}) - \hat{v}_\phi(S_t)$ no caso do método Ator-Crítico de um passo.

$$\theta_{t+1} = \theta_t + \alpha\, \psi_t\, \nabla ln\, \pi_\theta(A_t \mid S_t) \tag{3.30}$$

Algoritmo 3: Ator-Crítico de um passo

Inicializar parâmetros θ (da política) e ϕ (da função de valor de estado)
for *para cada episódio* **do**
  Inicializar s (primeiro estado)
  $I \leftarrow 1$
  while S *não é terminal* **do**
    a é selecionado da política π $a \leftarrow \pi_\theta(. \mid s)$
    Toma ação a e observa retornos do ambiente s' e r
    $\delta = r + \gamma \hat{v}_\phi(s') - \hat{v}_\phi(s)$ ▷ $\hat{v}_\phi(s') = 0$ se s' é terminal
    $\phi = \phi + \alpha^\phi \, \delta \, \nabla \hat{v}_\phi(s)$
    $\theta = \theta + \alpha^\theta \, I \, \delta \, \nabla ln \pi_\theta(a \mid s)$
    $I = \gamma I$
    $s = s'$

Já vimos que os algoritmos de AR podem ser baseados na função de valor de estado, $v_\pi(s)$ ou na função de valor de ação, $q_\pi(s, a)$. Mas eles também podem ser baseados na chamada *função de vantagem* apresentada na Equação 3.31. Com esta função, ao invés de termos uma indicação de desempenho de cada ação para um dado estado s sob a política π (como é o caso da função q_π), temos uma medida de comparação de cada ação em relação ao retorno esperado no estado s, dado por v_π. Portanto, a função de vantagem A_π pode ser mais uma opção de valor de ψ_t, mas que tem ordens de magnitude menores que a função q_π

$$A_\pi(s, a) = q_\pi(s, a) - v_\pi(s) \tag{3.31}$$

Segundo François-Lavet et al. [2018], em termos práticos, quando utilizamos uma RNA para aproximar uma política π_θ, a função de vantagem leva a uma redução da variância do estimador do gradiente $\nabla v_{\pi_\theta}(s_0)$, sem modificar a expectativa do gradiente (já que caímos no caso de um baseline que não depende da ação e portanto, não muda o resultado do gradiente). Portanto, o uso da função de vantagem melhora a eficiência numérica por permitir a utilização de taxas de aprendizados maiores, possibilitando que se alcance um certo nível de desempenho com menos atualizações. Vale notar que existem diferentes formas de estimar a função de vantagem. Uma forma comumente usada nos algoritmos *Policy Gradient* é a **Generalized Advantage Estimation (GAE)** [Schulman et al., 2018]. Tal abordagem usa a mesma ideia dos métodos TD(λ) apresentados anteriormente (Seção 3.3), mas para estimar a função de vantagem. Ou seja, GAE usa uma combinação exponencialmente ponderada de n-passos dos alvos da função de vantagem, o que geralmente reduz de forma substancial

a variância das estimativas dos algoritmos ao custo de algum viés [Morales, 2020].

3.4.6 A3C e A2C

O algoritmo **Asynchronous Advantage Actor-Critic (A3C)**, proposto por Mnih et al. [2016], procura reduzir o viés do REINFORCE com baseline, utilizando retornos de n-passos com *bootstrapping*, tanto para aprender a política quanto para aprender a função de valor, e usando atores concorrentes para gerar um conjunto de amostras de experiência em paralelo.

Segundo Morales [2020], o fato das amostras serem correlacionadas e não estacionárias são a principal causa da variância em algoritmos de AR. No caso de métodos aproximados baseados em valor, usa-se geralmente um *buffer* de repetição (*replay buffer*) para amostrar minilotes dos dados de maneira uniforme, a partir de dados que são, em sua maior parte, independentes e identicamente distribuídos. Mas *buffer* de repetição só pode ser usado em métodos *off-policy*, já que agentes *on-policy* não podem reutilizar dados gerados por políticas anteriores. Dessa forma, cada passo de otimização depende de um novo conjunto de experiências *on-policy*. O que o A3C faz para tratar isso é ter atores múltiplos gerando experiência em paralelo, a partir de múltiplas instâncias do ambiente, e atualizando, de forma assíncrona, a política e a função de valor. Tal estratégia remove a correlação dos dados usados para o treinamento, reduzindo a variância do algoritmo. A variância também é reduzida pelo uso dos retornos de n-passos, quando comparado aos retornos de Monte-Carlo. Como os retornos usados fazem uso de *bootstrapping*, o método adiciona viés e é um método Ator-Crítico.

A Equação 3.32 apresenta a função de vantagem usada no algoritmo que faz uso dos retornos de n-passos. Já a Equação 3.33 apresenta a função de custo (*loss*) utilizada pelo algoritmo. Além do uso da função de vantagem, tal equação pode acrescentar um termo de entropia para encorajar soluções com ações uniformemente distribuídas, o que permite uma maior exploração durante o treinamento. Por fim, a função de valor é ajustada utilizando a função de custo apresentada na Equação 3.34.

$$A_\phi(S_t, A_t) = R_t + \gamma R_{t+1} + \cdots + \gamma^n R_{t+n} + \gamma^{n+1} V_\phi(S_{t+n+1}) - V_\phi(S_t) \tag{3.32}$$

$$L_\pi(\theta) = -\frac{1}{N} \sum_{n=0}^{N} \left[A_\phi(S_t, A_t)\, ln\, \pi_\theta(A_t|S_t) + \beta H(\pi_\phi(S_t)) \right] \tag{3.33}$$

$$L_v(\phi) = \frac{1}{N}\sum_{n=0}^{N}\left[\left((R_t + \gamma R_{t+1} + \cdots + \gamma^n R_{t+n} + \gamma^{n+1} V_\phi(S_{t+n+1}) - V_\phi(S_t)\right)^2\right] \quad (3.34)$$

O método A3C, apresentado no Algoritmo 4, utiliza duas RNAs, uma para aproximar a política (o ator) e outra para aproximar a função de valor (o crítico). Os dados são coletados por múltiplos atores em paralelo e as atualizações dos pesos das RNAs são feitas assincronamente, sem semáforos.

Algoritmo 4: A3C (CÓDIGO PARA THREAD DE CADA ATOR)

▷ Dados parâmetros compartilhados globais θ (da política), ϕ (da função de valor) e contador T

▷ Considerando parâmetros específicos da thread: θ' e ϕ'

$t \leftarrow 1$ ▷ Contador de passos da thread

while $T \leq T_{max}$ **do**

  Reinicia gradientes: $d_\theta \leftarrow 0$ e $d_\phi \leftarrow 0$

  Sincroniza parâmetros da thread: $\theta' = \theta$ e $\phi' = \phi$

  $t_{inicio} = t$

  Obtém estado s

  repeat

    a é selecionado da política π $\quad a \leftarrow \pi_\theta(. \mid s)$

    Toma ação a e observa retornos do ambiente s' e r

    $t \leftarrow t + 1$

    $T \leftarrow T + 1$

  until *s é terminal ou* $t - t_{inicio} == t_{max}$

  $R = \hat{v}_{\phi'}(s)$ ▷ $\hat{v}_{\phi'}(s) = 0$ se s é terminal

  for $i \in \{t-1, \cdots, t_{inicio}\}$ **do**

    $R \leftarrow r_i + \gamma R$

    $d_\theta \leftarrow d_\theta + \nabla_{\theta'} log \pi_{\theta'}(a_i|s_i)(R - \hat{v}_{\phi'}(s_i))$ ▷ Acumula os gradientes

    $d_\phi \leftarrow d_\phi + \frac{\partial (R - \hat{v}_{\phi'}(s_i))^2}{\partial \phi'}$

  Atualiza de forma assíncrona: θ usando d_θ e ϕ usando d_ϕ

O algoritmo **Advantage Actor-Critic (A2C)**, foi proposto posteriormente como uma versão síncrona do A3C que alcança o mesmo nível de desempenho. Nesta versão, os atores coletam dados, potencialmente em paralelo, mas eles são agrupados em um único lote de dados utilizados nas atualizações das RNAs. Em termos práticos, o A2C traz a vantagem de poder se beneficiar do uso de GPUS, enquanto que para o A3C a CPU é o recurso mais importante [Morales, 2020].

3.4.7 PPO

O algoritmo **Proximal Policy Optimization (PPO)**[2], foi proposto por Schulman et al. [2017] com o objetivo de encontrar um equilíbrio entre facilidade de implementação e parametrização e complexidade amostral. PPO é um algoritmo *on-policy* que tem uma arquitetura similar a algoritmos Ator-Crítico anteriores, como o A2C, mas inova com duas contribuições principais. A primeira é uma função objetiva substituta (*surrogate*) que permite o uso de múltiplas etapas do método de gradiente no mesmo minilote de experiências. A segunda contribuição é uma limitação no tamanho dos passos de atualização. O objetivo é atualizar a política com uma nova que não seja tão diferente da atual. Isso já havia sido proposto por Schulman et al. [2015] em um algoritmo anterior (*Trust Region Policy Optimization*, TRPO); mas tal algoritmo utiliza uma função objetivo quadrática restrita, exigindo o cálculo de derivadas de segunda ordem, o que é difícil de parametrizar. Já o PPO usa o que os autores chamam de função cortada (*clipped*[3]) que precisa somente de derivadas de primeira ordem e, ao mesmo tempo, mantém a eficiência das amostras e desempenho confiável do TRPO. Esta abordagem conservadora de atualização da política evita quedas de desempenho e permite o reuso dos minilotes de experiência. Assim, em relação aos métodos *on-policy* anteriores, PPO é mais eficiente em relação às amostras e tem menor variância, alcançando melhor desempenho em diversos problemas.

Uma visão geral do PPO é apresentada na Figura 3.3. O algoritmo usa vários atores para coletar trajetórias (experiências) com a política atual e os agrupa em um lote. Por k épocas, este lote é aleatoriamente divido em minilotes que, por sua vez, são usados para atualizar os pesos das RNAs do ator e do crítico. Depois da atualização, o processo é repetido usando a nova política. Estes passos são repetidamente executados até que um critério de parada seja alcançado.

Conforme apresentado por Achiam [2018], o PPO atualiza a política como apresentado na Equação 3.35. Já a função objetivo é dada pela Equação 3.36, onde $\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}$ é a razão de probabilidade entre a nova política e a atual, ϵ é um hiperparâmetro de valor baixo que indica, intuitivamente, o quanto a nova política pode se distanciar da atual, e a função $clip(x, min, max)$ retorna x para $min \leq x \leq max$, min para $x < min$

[2]Na literatura, podemos encontrar o termo PPO2, se referindo à segunda versão do algoritmo, projetada para execução paralela em ambientes com GPU. Mas, recentemente, os textos têm usado o termo PPO para se referir ao algoritmo independentemente de sua forma de uso. Adotamos essa abordagem no presente texto.

[3]Há outra variante do PPO que, ao invés de usar a função cortada, penaliza a divergência Kullback-Leibler (uma medida de diferença entre duas distribuições de probabilidade) na função objetivo, ajustando de forma automática o peso da penalidade.

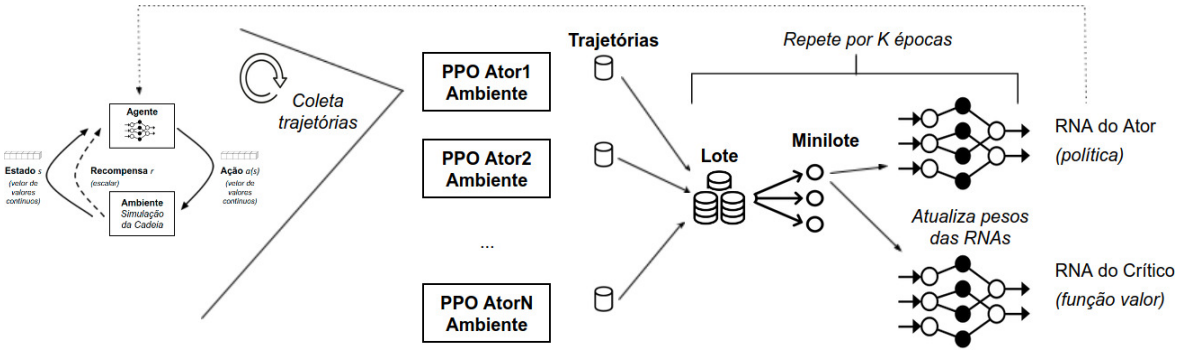

Figura 3.3. Ideia geral do algoritmo PPO. Em cada passo, os agentes coletam trajetórias usando a política atual. Os dados coletados são agrupados em um lote de experiências. O lote é aleatoriamente dividido em minilotes que são usados para atualizar os pesos das RNAs do ator e do crítico e esse processo é repetido k vezes. O processo é repetido com a política atualizada, e isto é feito até o fim do treinamento.

e max para $x > max$.

$$\theta_{k+1} = \arg\max_{\theta} \mathop{\mathbb{E}}_{s,a\sim\pi_{\theta_k}} [L(s, a, \theta_k, \theta)] \tag{3.35}$$

$$L^{CLIP}(s, a, \theta_k, \theta) = min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A_{\pi_{\theta_k}}(s, a), clip\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon\right) A_{\pi_{\theta_k}}(s, a)\right) \tag{3.36}$$

Para entender o efeito da função objetivo utilizada, vale analisar o comportamento da mesma quando a estimativa de vantagem é positiva ou negativa. No caso de vantagem positiva, se a probabilidade da ação a aumentar, o objetivo também aumentará. Mas a função min limitará o quanto o objetivo pode aumentar, podendo ser no máximo igual a $(1 + \epsilon)A_{\pi_{\theta_k}}(s, a)$. Já no caso de vantagem negativa, se a probabilidade da ação diminuir, o objetivo aumentará. Mas, da mesma forma, a função min limitará o quanto o objetivo pode aumentar, podendo agora ser no máximo igual a $(1 - \epsilon)A_{\pi_{\theta_k}}(s, a)$. Portanto, o corte nos valores funciona como um regularizador que incentiva a política a mudar mais suavemente.

Quando PPO é implementado com uma única RNA para aproximar a política (ator) e a função de valor (crítico), a função objetivo é dada pela Equação 3.37, que inclui um termo de erro da estimativa da função de valor ($L^{VF}(s, \theta)$) e um termo de entropia para encorajar exploração suficiente ($H(s, \pi_\theta(\cdot))$; e c_1 e c_2 são hiperparâmetros

do algoritmo.

$$L(\theta) = L^{CLIP}(s, a, \theta_k, \theta) - c_1 L_\theta^{VF}(s) + c_2 H(s, \pi_\theta(\cdot)) \tag{3.37}$$

O pseudocódigo do PPO é apresentado no Algoritmo 5. Na linha 1 os parâmetros da política, θ_0, e da função de valor, ϕ_0, são inicializados (aleatoriamente, por exemplo). O laço iniciado na linha 2 se refere ao número total de passos que o algoritmo será executado, que é um parâmetro a ser definido para o problema. O laço mostrado nas linhas 3-6 é responsável por coletar o *buffer* de experiências (trajetórias). Cada ator (potencialmente em paralelo) executa a política atual por τ períodos e coleta um conjunto de experiências (linha 4), são calculadas as somas das recompensas após cada ação (linha 5) e, então, na linha 6, a função de de vantagem é calculada para cada experiência. PPO usa uma versão truncada da estimativa de vantagem generalizada (GAE), dada por $\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + ... + (\gamma\lambda)^{\tau-t+1}\delta_{\tau-1}$, onde: t é o período na faixa $[0, \tau]$; $\delta_t = r_t + \gamma v(s_{t+1}) - v(s_t)$; v é a função de valor; e γ e λ são parâmetros do algoritmo. Os *buffers* de experiências coletados por todos os atores são agrupados em um lote na linha 7. As linhas 8-12 realizam múltiplos passos de gradiente com as experiências coletadas. O lote de experiências é aleatoriamente dividido em minilotes (linha 9) e cada minilote é usado para atualizar os pesos das RNAs tanto da política (linha 11) quando da função de valor (linha 12). Este processo dividir o lote aleatoriamente e atualizar as RNAs é repetido k vezes. Por fim, os parâmetros da política e da função de valor são atualizados na linha 13 para a próxima iteração do laço principal.

Além dos parâmetros apresentados, o número e o tamanho das camadas ocultas das RNAs e o tamanho do passo de atualização (taxa de aprendizado) usados nos métodos de gradiente são outros parâmetros do PPO.

3.4.8 DDPG

O algoritmo **Deep Deterministic Policy Gradient (DDPG)**, proposto por Lillicrap et al. [2015], pode ser visto com um método Deep Q-Learning para espaços de ação contínuos. Ele usa dados *off-policy* e as equações de Belmann para aprender a função de valor de ação (*Q-function*), e usa a função para aprender a política. Os detalhes do algoritmo são apresentados a seguir conforme descrito por Achiam [2018].

A motivação do algoritmo é a mesma do Q-Learning, ou seja, se conhecermos a função de valor de ação ótima $q_*(s, a)$, então, em um dado estado, a ação ótima $a_*(s)$ pode ser encontrada usando $a_*(s) = arg\,max_a q_*(s, a)$. O algoritmo intercala o aprendizado de um aproximador para $q_*(s, a)$ com o aprendizado de um aproximador para $a_*(s)$, de uma forma que é adaptada para ambientes com espaços de ação contínuos.

Algoritmo 5: PPO

Inicializar parâmetros θ_0 (da política) e ϕ_0 (da função de valor)
for *i=0,1,2,...* **do**
  for *ator=1,2,...,N* **do**
    Executar política π_{θ_0} por τ períodos e coletar as trajetórias $\mathcal{D}_k$
    Calcular as recompensas $\hat{R}$ ▷ *rewards-to-go*
    Calcular estimativas de vantagem $\hat{A}_1, ..., \hat{A}_\tau$ baseadas na função de valor V_ϕ
  Formar um *batch*, de tamanho $N\tau$, com as trajetórias e vantagens coletadas
  for *k=1,2,...,K* **do**
    Embaralha *batch* e divide-o em *minibatches*
    for *cada minibatch* **do**
      Atualiza a política maximizando o objetivo $\theta_{k+1} = \arg\max_{\theta} \underset{s,a\sim\pi_{\theta_k}}{\mathrm{E}} [L(s, a, \theta_k, \theta)]$ via gradiente estocástico (*ascent*)
      Ajusta a função de valor por regressão no erro quadrado médio via gradiente (*descent*) $\phi_{k+1} = \arg\min_{\phi} \mathbb{E}[(v_\phi(s) - \hat{R})^2]$
  $\theta_0 \leftarrow \theta_K$ $\phi_0 \leftarrow \phi_K$

Devemos lembrar que no caso de espaços de ação contínuos não é simples encontrar a melhor ação, pois isso significaria resolver um subproblema de otimização. E tal problema precisaria ser resolvido a cada transição, o que é inviável na prática. Mas como o espaço de ações é contínuo, assumimos que a função $q_*(s, a)$ é diferenciável com respeito às ações, o que nos permite criar uma regra de aprendizado baseada em gradiente para uma política $\pi(s)$. Assim, ao invés de utilizar um subproblema de otimização para encontrar $max_a\, q(s, a)$, podemos aproximar a solução como $\hat{q}(s, \pi(s))$.

Para aprender a função de valor de ação, o algoritmo se baseia na Equação de Bellman (Equação 3.12). Uma função $q_\phi(s, a)$, parametrizada por ϕ, pode ser aprendida utilizando-se o *erro médio quadrado de Bellman*, que nos indica de forma aproximada o quão próximo a função satisfaz a equação de Bellman. A função de custo usada para aprender a função parametrizada pode ser dada pela Equação 3.38, onde (s, a, r, s', d) é um conjunto $\mathcal{D}$ de amostras de transição coletadas, sendo $d = 1$ se s' é um estado terminal e 0, em caso contrário.

$$L_q(\phi, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathbb{E}} \left[\Big(q_\phi(s, a) - \big(r + \gamma\,(1 - d)\, \underset{a'}{max}\, q_\phi(s', a') \big) \Big)^2 \right] \tag{3.38}$$

Na prática, o algoritmo DDPG utiliza três artifícios para obter melhor desempenho. O primeiro deles é o uso de *buffer* de repetição (o conjunto $\mathcal{D}$ mencionado anteriormente), como é comum nos algoritmos que aproximam a função $q_*(s, a)$ utilizando uma RNA. É isso que torna o algoritmo DDPG um método *off-policy*, já que o *buffer* inclui transições coletadas com diferentes versões da política (não apenas a atual). Isto é possível porque a Equação de Bellman considera todas as possíveis transições, não importando como elas foram geradas. Para que o algoritmo tenha comportamento estável, é importante que o *buffer* tenha tamanho suficiente para conter uma grande variedade de experiências. Mas o tamanho do *buffer* é um parâmetro que precisa ser ajustado com cuidado, porque se o *buffer* tiver apenas os dados mais recentes, ele vai causar *overfitting* e comprometerá o desempenho, por outro lado, se ele for muito grande pode tornar o aprendizado lento.

O segundo artifício utilizado pelo DDPG é a utilização de redes-alvo (*target networks*). O nome vem do termo $r + \gamma\,(1 - d)\,max_{a'} q_\phi(s', a')$ que é chamado *alvo*, porque quando minimizamos a função de erro médio quadrado de Bellman, estamos tentando fazer com que a função de valor de ação aproximada seja mais próxima deste alvo. Mas existe um problema de dependência, uma vez que o alvo depende dos mesmos parâmetros ϕ que estamos tentando treinar, e isso causa instabilidades na minimização da função de custo. A solução adotada pelo algoritmo é usar um conjunto de parâmetros que é próximo de ϕ, mas com um certo atraso. Isto é alcançado através do uso de uma segunda RNA, a rede-alvo. Em uma das formas de uso, essa rede é atualizada com os valores da RNA principal a cada certo número fixo de passos. Outra possível abordagem é atualizar a rede-alvo a cada atualização da rede principal, mas usando média de Polyak dada por $\phi_{alvo} = \rho\phi_{alvo} + (1-\rho)\phi$, onde $\rho \in [0, 1]$ é um hiperparâmento a ser ajustado e que, usualmente, tem valores próximos de um.

O terceiro artifício usado pelo algoritmo DDPG é usar uma rede-alvo para a política, o que possibilita calcular uma ação que maximiza de forma aproximada $q_{\phi_{alvo}}$. A rede-alvo para a política é atualizada da mesma forma que a rede-alvo para a função de valor de ação, ou seja, usando média de Polyak para os parâmetros da política ao longo do treinamento.

Uma vez explicados os três artifícios, podemos chegar na função de custo realmente utilizada pelo DDPG para minimizar o erro quadrado de Bellman com um método de gradiente (*descent*), dada pela Equação 3.39, onde $\pi_{\theta_{alvo}}(s)$ é a política da

rede-alvo.

$$L_q(\phi, \mathcal{D}) = \mathop{\mathbb{E}}_{(s,a,r,s',d)\sim\mathcal{D}}\left[\left(q_\phi(s,a) - \left(r + \gamma\,(1-d)\,q_\phi(s', \pi_{\theta_{alvo}}(s'))\right)\right)^2\right] \tag{3.39}$$

Em relação à política, o objetivo do algoritmo é aprender uma política $\pi_\theta(s)$ que retorne a ação que maximiza $q_\phi(s,a)$. Como o espaço de ações é contínuo, assumimos que a função q é diferenciável com relação às ações, e usamos o método de gradiente (*ascent*) para resolver o problema dado na Equação 3.40 em relação aos parâmetros da política, considerando os parâmetros da função q são constantes.

$$\max_\theta \mathop{\mathbb{E}}_{s\sim\mathcal{D}}[q_\phi(s, \pi_\theta(s))] \tag{3.40}$$

Como a política aprendida é determinística, para melhorar a exploração do algoritmo, adiciona-se um ruído nas ações retornadas durante a fase de treinamento. No algoritmo original, os autores recomendam o uso do ruído de Ornstein–Uhlenbeck correlacionado ao período, mas há também estudos que mostram bons resultados com um ruído Gaussiano não-correlacionado de média zero. Nas implementações do algoritmo, é comum também o uso de estratégias de redução da escala do ruído ao longo do treinamento, de forma a se obter uma maior exploração no início, mas que é diminuída ao longo do processo. Por fim, um último artifício comumente encontrado nas implementações para melhorar a exploração, é o uso de ações amostradas de uma distribuição aleatória uniforme nos primeiros passos do algoritmo. A quantidade de passos é um hiperparâmetro a ser ajustado e, depois disso, é seguida a estratégia normal de exploração do DDPG.

O pseudocódigo do método DDPG é apresentado no Algoritmo 6. Na linha 1 os parâmetros da política, θ, e da função q, ϕ, são inicializados (aleatoriamente, por exemplo). Na linha 2 o *buffer* de repetição é inicializado vazio e na linha 3 os parâmetros θ_{alvo} e ϕ_{alvo} são inicializados com os valores de θ e ϕ, respectivamente. Um primeiro estado s é obtido do ambiente na linha 4. O laço iniciado na linha 5 é repetido até que uma condição de parada (por exemplo, alguma medida de convergência) seja atingida. Dentro do laço, na linha 6, uma ação a é obtida da política atual, adicionando-se um ruído Gaussiano e limitando os valores à faixa de ações válidas para o problema. A ação é executada no ambiente, obtendo-se o próximo estado s', a recompensa r e uma indicação se o estado é final d (linha 7). Na linha 8, a experiência coletada é adicionada ao *buffer* de repetição. A condição da linha 9, indica que o ambiente precisa ser reiniciado (voltar ao estado inicial) se o episódio estiver terminado. O bloco

das linhas 11-18 é executado de acordo com um hiperparâmetro do algoritmo que geralmente indica a cada quantas iterações do laço principal a condição será satisfeita. O laço iniciado na linha 12 é executado com um número de iterações dado por outro hiperparâmetro do algoritmo. Dentro do laço, na linha 13, uma amostra (conjunto de transições) é obtida aleatoriamente do *buffer* de repetição. Na linha 14, são calculados os alvos δ, que são utilizados na atualização da função q, realizada na linha 15, através de um passo do método de gradiente (*descent*). Na linha 16, a política é atualizada com um passo do método do gradiente (*ascent*) considerando a função q atualizada. Por fim, nas linhas 17 e 18, os parâmetros alvo θ_{alvo} e ϕ_{alvo} são atualizados de acordo com a média de Polyak.

Algoritmo 6: DDPG

1 Inicializar parâmetros θ (da política) e ϕ (da função q)
2 Inicializar *buffer* de repetição vazio $\mathcal{D}$
3 $\theta_{alvo} \leftarrow \theta$, $\phi_{alvo} \leftarrow \phi$
4 Inicializa o ambiente e obtém s
5 **repeat**
6 $a \leftarrow clip(\pi_\theta(s) + \epsilon, a_{min}, a_{max})$ com $\epsilon \sim \mathcal{N}$
7 Toma a ação a no ambiente e obtém s', r, d
8 Armazena (s, a, r, s', d) no *buffer* de repetição $\mathcal{D}$
9 **if** *s' é terminal* **then**
10 Reinicia o ambiente
11 **if** *é hora de atualizar* **then**
12 **for** *número de atualizações* **do**
13 Amostra aleatoriamente minilote $B = \{(s, a, r, s', d)\}$ de $\mathcal{D}$
14 $\delta(r, s', d) = r + \gamma(1 - d)q_{\phi_{alvo}}(s', \pi_{\theta_{alvo}}(s'))$ ▷ alvos
15 Atualiza a função q com um passo do método de gradiente (*descent*) usando $\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} (q_\phi(s, a) - \delta(r, s', d))^2$
16 Atualiza a política com um passo do método de gradiente (*ascent*) usando $\nabla_\theta \frac{1}{|B|} \sum_{s \in B} q_\phi(s, \pi_\theta(s))$
17 $\phi_{alvo} = \rho\phi_{alvo} + (1 - \rho)\phi$
18 $\theta_{alvo} = \rho\theta_{alvo} + (1 - \rho)\theta$
19 **until** *condição de parada*

3.4.9 TD3

O algoritmo Twin Delayed DDPG (TD3) foi proposto por Fujimoto et al. [2018], com o objetivo de resolver problemas comumente encontrados no DDPG. Apesar do DDPG poder alcançar bons resultados em alguns casos, seu desempenho pode também dete-

riorar em outros, e um dos motivos é que a função q aprendida pode superestimar os valores q. As características do algoritmo são apresentados a seguir conforme descrito por Achiam [2018] e Morales [2020].

O algoritmo TD3 utiliza três artifícios para melhorar o desempenho do algoritmo DDPG. O primeiro deles é a suavização da política (*target policy smoothing*). A motivação é usar um regularizador para tentar evitar que o algoritmo aprenda uma função q com um pico agudo incorreto para algumas ações. Pois, caso isso aconteça, o algoritmo começará explorar tal pico, prejudicando seu desempenho. A solução adotada é adicionar ruído não só na ação retornada mas também na ação usada para formar os alvos. Treinar a política com alvos com ruído pode ser visto como uma forma de regularização porque a rede é forçada a generalizar sobre ações similares. Isso ajuda a evitar que a política convirja para ações incorretas no início do treinamento. Basicamente, a ideia é incluir um ruído Gaussiano limitado para cada dimensão das ações (antes do valor da ação ser limitado para se manter na faixa válida, $a_{min} \leq a \leq a_{max}$). Assim, a ação usada para formar os alvos é dada pela Equação 3.41.

$$a'(s') = clip(\pi_{\theta_{alvo}}(s') + clip(\epsilon, \epsilon_{min}, \epsilon_{max}), a_{min}, a_{max}), \quad \epsilon \sim \mathcal{N}(0, \sigma) \tag{3.41}$$

O segundo artifício utilizado pelo TD3 é o chamado *clipped double-Q learning*, que significa que o algoritmo aprende duas funções q ao invés de uma, daí o termo *twin*, e usa o menor dos dois valores q para formar os alvos na função de erro médio quadrado de Bellman. Os alvos são calculados como mostrado na Equação 3.42, e são usados na hora de aprender as duas funções q por regressão, como mostrado nas Equações 3.43. O uso do menor dos valores q para o alvo e a regressão em direção a tal alvo ajudam a evitar que a função q seja superestimada. Vale notar que a função é aprendida como no DDPG, considerando apenas a primeira das funções q (ou seja $q(s, \pi(s, \theta), \phi_1)$).

$$\delta(r, s', d) = r + \gamma\,(1 - d)\,\min_{i=1,2} q_{\phi_i}(s', a'(s')) \tag{3.42}$$

$$\begin{aligned} L_q(\phi_1, \mathcal{D}) &= \mathop{\mathbb{E}}_{(s,a,r,s',d)\sim\mathcal{D}}\left[\left(q_{\phi_1}(s,a) - \delta(r,s',d)\right)^2\right] \\ L_q(\phi_2, \mathcal{D}) &= \mathop{\mathbb{E}}_{(s,a,r,s',d)\sim\mathcal{D}}\left[\left(q_{\phi_2}(s,a) - \delta(r,s',d)\right)^2\right] \end{aligned} \tag{3.43}$$

O terceiro artifício usado pelo TD3 consiste em atualizar a política (e as redes-

alvo) com menos frequência que a função q. O artigo original do algoritmo propõe uma atualização a cada duas da função q. A ideia é ajudar a amortecer a volatilidade que usualmente aparece no DDPG, causada pela forma com a atualização da política altera os alvos. Portanto a ideia é esperar um pouco para que se tenha uma informação mais correta para atualizar a política.

Em relação à exploração do algoritmo, as implementações do TD3 costumam trazer as mesmas estratégias do DDPG de adicionar um ruído Gaussiano nas ações retornadas pelo algoritmo (com decaimento ao longo do treinamento) e o uso de ações aleatórias no início do treinamento para alimentar o *buffer* de repetição.

O pseudocódigo do método TD3 é apresentado no Algoritmo 7. Na linha 1 os parâmetros da política, θ, e das duas funções q, ϕ_1 e ϕ_2, são inicializados (aleatoriamente, por exemplo). Na linha 2 o *buffer* de repetição é inicializado vazio e na linha 3 os parâmetros θ_{alvo}, $\phi_{alvo,1}$ e $\phi_{alvo,2}$ são inicializados com os valores de θ, ϕ_1 e ϕ_2, respectivamente. Um primeiro estado s é obtido do ambiente na linha 4. O laço iniciado na linha 5 é repetido até que uma condição de parada (por exemplo, alguma medida de convergência) seja atingida. Dentro do laço, na linha 6, uma ação é a é obtida da política atual, adicionando-se um ruído Gaussiano e limitando os valores à faixa de ações válidas para o problema. A ação é executada no ambiente, obtendo-se o próximo estado s', a recompensa r e uma indicação se o estado é final d (linha 7). Na linha 8, a experiência coletada é adicionada ao *buffer* de repetição. A condição da linha 9, indica que o ambiente precisa ser reiniciado (voltar ao estado inicial) se o episódio estiver terminado. O bloco das linhas 11-20 é executado de acordo com um hiperparâmetro do algoritmo que geralmente indica a cada quantas iterações do laço principal a condição será satisfeita. O laço iniciado na linha 12 é executado com um número de iterações dado por outro hiperparâmetro do algoritmo. Dentro do laço, na linha 13, uma amostra (conjunto de transições) é obtida aleatoriamente do *buffer* de repetição. Na linha 14, as ações a serem usadas para calcular os alvos são obtidas da política alvo, considerando um ruído Gaussiano limitado e também a limitação dentro da faixa de ações válidas para o problema. Na linha 15, são calculados os alvos δ, que são utilizados na atualização das duas funções q, realizadas na linha 16, através de um passo do método de gradiente (*descent*). A condição da linha 17 indica o atraso na atualização da política e a frequência de atualização é mais um hiperparâmetro do algoritmo. Quando é hora de atualizar a política, isso é feito na linha 18 com um passo do método do gradiente (*ascent*) considerando a primeira função q. Por fim, nas linhas 19 e 20, os parâmetros alvo θ_{alvo}, $\phi_{alvo,1}$ e $\phi_{alvo,2}$ são atualizados de acordo com a média de Polyak.

Algoritmo 7: TD3

Inicializar parâmetros θ (da política) e ϕ_1 e ϕ_2 (das funções q)
Inicializar *buffer* de repetição vazio $\mathcal{D}$
$\theta_{alvo} \leftarrow \theta$, $\phi_{alvo,1} \leftarrow \phi_1$, $\phi_{alvo,2} \leftarrow \phi_2$
Inicializa o ambiente e obtém s
repeat
    $a \leftarrow clip(\pi_\theta(s) + \epsilon, a_{min}, a_{max})$ com $e \sim \mathcal{N}$
    Toma a ação a no ambiente e obtém s', r, d
    Armazena (s, a, r, s', d) no *buffer* de repetição $\mathcal{D}$
    if s' *é terminal* **then**
        Reinicia o ambiente
    if *é hora de atualizar* **then**
        for *número de atualizações* **do**
            Amostra aleatoriamente minilote $B = \{(s, a, r, s', d)\}$ de $\mathcal{D}$
            $a'(s') = clip(\pi_{\theta_{alvo}}(s') + clip(\epsilon, \epsilon_{min}, \epsilon_{max}), a_{min}, a_{max})$, $\epsilon \sim \mathcal{N}(0, \sigma)$
             $\delta(r, s', d) = r + \gamma(1 - d)\, min_{i=1,2}\, q_{\phi_{alvo,i}}(s', a'(s'))$ ▷ alvos
            Atualiza as funções q com um passo do método de gradiente (*descent*) usando $\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} (q_{\phi_i}(s, a) - \delta(r, s', d))^2$ para $i = 1, 2$
            if j *mod atraso_politica* $= 0$ **then**
                Atualiza a política com um passo do método de gradiente (*ascent*) usando $\nabla_\theta \frac{1}{|B|} \sum_{s \in B} q_{\phi_1}(s, \pi(s, \theta))$
                $\phi_{alvo,i} = \rho\phi_{alvo,i} + (1 - \rho)\phi_i$ para $i = 1, 2$
                $\theta_{alvo} = \rho\theta_{alvo} + (1 - \rho)\theta$
until *condição de parada*

3.4.10 SAC

O método **Soft Actor-Critic (SAC)**, foi proposto por Haarnoja et al. [2018], e tem como ideia principal a regularização da entropia (uma medida da aleatoriedade da política). É um método que aprende de maneira *off-policy*, como o DDPG ou TD3, mas, diferente destes algoritmos, otimiza uma política estocástica. A versão original do algoritmo trata espaços de ação contínuos, mas ele pode ser adaptado também para espaços de ação discretos. Como nos algoritmos anteriores, as características do algoritmo são apresentadas a seguir conforme descrito por Achiam [2018] e Morales [2020].

Como a ideia principal do algoritmo se refere à regularização da entropia, vale a pena detalharmos o que isso significa em um problema de AR. Dada uma variável aleatória x com função de densidade P, a entropia H de x é calculada a partir da distribuição P como $H(P) = \mathbb{E}_{x \sim P}[-logP(x)]$. O problema de encontrar uma polí-

tica ótima no AR regularizado por entropia pode ser definido como apresentado na Equação 3.44, considerando configuração descontada de horizonte infinito, onde $\alpha > 0$ é um coeficiente de *trade-off*. Note que neste problema o agente recebe um bônus de recompensa a cada período, proporcional à entropia da política naquele período. Neste contexto, a função de valor de ação q_π é também alterada para incluir o bônus de entropia, conforme apresentado na Equação 3.45. Veja que a entropia tem uma conexão com o *exploration-exploitation trade-off*, uma vez que aumentar a entropia resulta em uma maior exploração, o que pode acelerar o treinamento mais adiante, e evitar que a política convirja prematuramente para um ótimo local ruim.

$$\pi^* = \arg\max_{\pi} \mathop{\mathbb{E}}_{\tau\sim\pi}\Big[\sum_{t=0}^{\infty} \gamma^t \big(R(S_t, A_t, S_{t+1}) + \alpha H(\pi(\cdot|S_t))\big)\Big] \tag{3.44}$$

$$q_\pi(s,a) = \mathop{\mathbb{E}}_{r,s'\sim P(s,a), a'\sim\pi(s')}\Big[r + \gamma\big(q_\pi(s',a') + \alpha H(\pi(\cdot|s'))\big)\Big] \tag{3.45}$$

O algoritmo SAC aprende de forma concorrente uma política π_θ e duas funções de valor de ação, q_{ϕ_1} e q_{ϕ_2}. Em uma variante do algoritmo o coeficiente de regularização de entropia α é fixo, já em outra o valor de α é ajustado automaticamente por uma método de otimização baseado em gradiente em direção a uma entropia heurística esperada. O método SAC, apesar de não ser um sucessor direto do TD3, compartilha com ele diversas características propostas na versão original do algoritmo, ou acrescentadas depois. Entre as características compartilhadas estão: (1) o fato das funções q serem aprendidas com a minimização do erro médio quadrado de Bellman através de regressão para um único alvo, (2) o cálculo do alvo compartilhado usar as redes-alvo da função q (usando média de Polyak), e (3) o uso do artifício *clipped double-Q learning*. Mas, diferente do TD3, no SAC: (1) o alvo inclui o termo de entropia, (2) as ações do próximo estado usadas no alvo vêm da política atual ao invés da política alvo, e (3) não há uma estratégia explícita de suavização da política. Contudo, em relação a este último ponto, como o SAC aprende uma política estocástica, o ruído da estocasticidade é suficiente para obter um efeito similar de suavização.

Dadas a definição de entropia e a Equação 3.45, e considerando amostras da recompensa e do próximo estado vindas do *buffer* de repetição e amostras das próximas ações vindas da política atual, podemos aproximar a função q, em expectativa, conforme apresentado na Equação 3.46, onde $\hat{a}'$ enfatiza que as próximas ações vêm da política atual e não do *buffer* de repetição.

$$q_\pi(s,a) \approx r + \gamma\big(q_\pi(s',\hat{a}') + \alpha\, log\, \pi(\hat{a}', s')\big), \quad \hat{a}' \sim \pi(\cdot|s') \tag{3.46}$$

Os alvos são dados conforme apresentado na Equação 3.47 e a função de custo para as redes das funções q é dada pela Equação 3.48.

$$\delta(r, s', d) = r + \gamma(1 - d)\Big(\min_{i=1,2} q_{\phi_{alvo,i}}(s', \hat{a}') - \alpha \log \pi_\theta(\hat{a}'|s')\Big), \quad \hat{a}' \sim \pi(\cdot|s') \tag{3.47}$$

$$L(\phi_i, \mathcal{D}) = \mathop{\mathbb{E}}_{(s,a,r,s',d)\sim\mathcal{D}}\left[\left(q_{\phi_i}(s, a) - \delta(r, s', d)\right)^2\right], \quad i = 1, 2 \tag{3.48}$$

Para aprender a política, devemos nos lembrar que precisamos tomar, em cada estado, a ação que maximiza o retorno esperado mais a entropia futura esperada. Portanto, a política deve maximizar a função de valor de estado, como apresentada na Equação 3.49.

$$v_\pi(s) = \mathop{\mathbb{E}}_{a\sim\pi}[q_\pi(s, a) - \alpha \log \pi(a|s)] \tag{3.49}$$

A política é otimizada através do chamado *reparameterization trick*, no qual uma amostra de $\pi_\theta(\cdot|s)$ é obtida através de uma função determinística do estado, dos parâmetros da política e de um ruído independente. No artigo original do SAC, as amostras são obtidas de acordo com a Equação 3.50, usando uma política Gaussiana *achatada* (*squashed*). O uso da tangente hiperbólica (*tanh*) garante que as ações serão limitadas a uma faixa finita. Vale notar que, no caso do SAC, a parametrização da política deve necessariamente usar desvios padrões dependentes do estado e, portanto, eles são dados pela saída das RNAs (ao invés da abordagem de se utilizar um vetor independente de estado).

$$\tilde{a}_\theta(s, \xi) = tanh\big(\pi_\theta(s) + \sigma_\theta(s) \odot \xi\big), \quad \xi \sim \mathcal{N}(0, I) \tag{3.50}$$

O *reparameterization trick* permite substituir a expectativa sobre a ações, que depende dos parâmetros da política, por uma expectativa sobre o ruído, que não depende de parâmetros, como mostrado na Equação 3.51. A função de custo da política, apresentada na Equação 3.52, é dada pelo mínimo entre os dois aproximadores da função q, diferente do TD3 que usa apenas o primeiro aproximador.

$$\mathop{\mathbb{E}}_{a\sim\pi_\theta}[q_{\pi_\theta}(s, a) - \alpha \log \pi_\theta(a|s)] = \mathop{\mathbb{E}}_{\xi\sim\mathcal{N}}[q_{\pi_\theta}(s, \tilde{a}_\theta(s, \xi)) - \alpha \log \pi_\theta(\tilde{a}_\theta(s, \xi)|s)] \tag{3.51}$$

$$\max_{\theta} \mathop{\mathbb{E}}_{s\sim D,\xi\sim\mathcal{N}}\big[\min_{i=1,2} q_{\phi_i}(s,\tilde{a}_\theta(s,\xi)) - \alpha\, log\, \pi_\theta(\tilde{a}_\theta(s,\xi)|s)\big] \tag{3.52}$$

Assim como no TD3, implementações do SAC costumam usar ações aleatórias para alimentar o *buffer* de repetição no início do treinamento. Já em relação à exploração do algoritmo, como já mencionado, o parâmetro α controla o peso da entropia e, portanto, o nível de exploração (quanto maior o valor de α, maior a exploração). Desta forma, este parâmetro deve ser ajustado cuidadosamente para cada ambiente, ou pode-se utilizar a estratégia de otimização automática de seu valor.

O pseudocódigo do método SAC é apresentado no Algoritmo 8. Na linha 1 os parâmetros da política, θ, e das duas funções q, ϕ_1 e ϕ_2, são inicializados (aleatoriamente, por exemplo). Na linha 2 o *buffer* de repetição é inicializado vazio e na linha 3 os parâmetros $\phi_{alvo,1}$ e $\phi_{alvo,2}$ são inicializados com os valores de ϕ_1 e ϕ_2, respectivamente. Um primeiro estado s é obtido do ambiente na linha 4. O laço iniciado na linha 5 é repetido até que uma condição de parada (por exemplo, alguma medida de convergência) seja atingida. Dentro do laço, na linha 6, uma ação é a é obtida da política atual. A ação é executada no ambiente, obtendo-se o próximo estado s', a recompensa r e uma indicação se o estado é final d (linha 7). Na linha 8, a experiência coletada é adicionada ao *buffer* de repetição. A condição da linha 9, indica que o ambiente precisa ser reiniciado (voltar ao estado inicial) se o episódio estiver terminado. O bloco das linhas 11-20 é executado de acordo com um hiperparâmetro do algoritmo que geralmente indica a cada quantas iterações do laço principal a condição será satisfeita. O laço iniciado na linha 12 é executado com um número de iterações dado por outro hiperparâmetro do algoritmo. Dentro do laço, na linha 13, uma amostra (conjunto de transições) é obtida aleatoriamente do *buffer* de repetição. Na linha 14, são calculados os alvos δ usando o estado e a recompensa amostrados do *buffer* de experiências e a ação obtida da política atual. As duas funções q são atualizadas na linha 15, através de um passo do método de gradiente (*descent*). Na linha 16 a política é atualizada com um passo do método do gradiente (*ascent*) considerando a menor das funções q e a obtendo-se a ação através do *reparemeterization trick*. Por fim, na linha 17, os parâmetros alvo $\phi_{alvo,1}$ e $\phi_{alvo,2}$ são atualizados de acordo com a média de Polyak.

3.4.11 Resumo sobre os Algoritmos

Nas últimas seções foram tratados os cinco algoritmos que utilizamos nos experimentos da presente tese: A2C, DDPG, PPO, SAC e TD3, e, nesta seção, apresentamos um pequeno sumário sobre as características destes algoritmos.

Algoritmo 8: SAC

Inicializar parâmetros θ (da política) e ϕ_1 e ϕ_2 (das funções q)
Inicializar *buffer* de repetição vazio $\mathcal{D}$
$\phi_{alvo,1} \leftarrow \phi_1$, $\phi_{alvo,2} \leftarrow \phi_2$
Inicializa o ambiente e obtém s
repeat
  Seleciona a ação $a \sim \pi_\theta(\cdot|s)$
  Toma a ação a no ambiente e obtém s', r, d
  Armazena (s, a, r, s', d) no *buffer* de repetição $\mathcal{D}$
  if *s' é terminal* **then**
    Reinicia o ambiente
  if *é hora de atualizar* **then**
    for *número de atualizações* **do**
      Amostra aleatoriamente minilote $B = \{(s, a, r, s', d)\}$ de $\mathcal{D}$
      $\delta(r, s', d) =$
       $r + \gamma\,(1-d)\,\big(\min_{i=1,2} q_{\phi_{alvo,i}}(s', \tilde{a}') - \alpha\, log\, \pi_\theta(\tilde{a}', s')\big), \quad \tilde{a}' \sim \pi_\theta(\cdot|s')$
      Atualiza as funções q com um passo do método de gradiente (*descent*) usando $\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d)\in B}(q_{\phi_i}(s, a) - \delta(r, s', d))^2$ para $i = 1, 2$
      Atualiza a política com um passo do método de gradiente (*ascent*) usando $\nabla_\theta \frac{1}{|B|} \sum_{s\in B} \big(\min_{i=1,2} q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha\, log\, \pi_\theta(\tilde{a}_\theta(s)|s)\big)$, onde $\tilde{a}_\theta(s)$ é amostrado de $\pi_\theta(\cdot|s)$ via *reparameterization trick*
      $\phi_{alvo,i} = \rho\phi_{alvo,i} + (1-\rho)\phi_i$ para $i = 1, 2$
until *condição de parada*

Os algoritmos A2C e PPO são métodos *on-policy* que aprendem uma política estocástica. Por serem *on-policy*, tais métodos não podem fazer uso de um *buffer* de repetição, sendo, portanto, menos eficientes em relação às amostras (ou seja, precisam coletar mais dados para melhorar a política). Por outro lado, por aprenderem políticas estocásticas, eles costumam apresentar treinamentos mais estáveis, o que pode acelerar o aprendizado. Já os métodos DDPG e TD3 são métodos *off-policy* que aprendem uma política determinística. Dessa forma, eles fazem uso de um *buffer* de repetição, o que costuma levar a uma maior eficiência em relação às amostras. Mas, por outro lado, tais métodos possuem treinamento mais instável, e artifícios são utilizados por cada algoritmo para amenizar esse problema. Já o algoritmo SAC tenta unir a vantagem da eficiência em relação às amostras, sendo um método *off-policy* como o DDPG ou TD3, com a estabilidade ao longo do treinamento, por aprender uma política estocástica como o A2C ou PPO.

Comparando especificamente o A2C com o PPO, ambos usam a ideia de vários

atores coletando dados a partir de múltiplas instâncias do ambiente. A principal diferença é que o PPO utiliza uma função substituta que evita que uma nova política seja muito diferente da anterior. Dadas as características dos dois algoritmos, e a partir de resultados publicados para diversos problemas, espera-se que o PPO tenha um desempenho melhor que o A2C, pois a limitação do passo na atualização da política tende a gerar treinamentos mais estáveis, o que costuma acelerar o aprendizado. Já se compararmos o DDPG com o TD3, o segundo é uma evolução do primeiro, utilizando artifícios para resolver problemas que comumente acontecem no DDPG, como as superestimativas da função q. Desta forma, é esperado que, em geral, o TD3 tenha melhor desempenho que o DDPG. Por fim, em relação ao SAC, pelo fato dele maximizar a entropia junto com as recompensas, ele tende a explorar o espaço de busca de forma diferente, podendo, potencialmente, encontrar soluções diferentes dos demais métodos.

De toda forma, vale ressaltar que tais avaliações se referem ao comportamento ou desempenho dos algoritmos de forma geral, mas características específicas de um problema podem favorecer um ou outro algoritmo. Por isso, nesta tese experimentamos os cinco algoritmos para a solução do problema abordado.

3.5 RNAs e Métodos de Gradiente Estocástico

Os métodos *Policy Gradient* apresentados nas seções anteriores, e utilizados nos experimentos desta tese, usam RNAs para aproximação da política e da função de valor, e tais redes têm seus pesos atualizados através de métodos de gradiente estocástico. Esta seção tem então o objetivo apresentar uma visão geral sobre RNAs (Seção 3.5.1), conforme apresentado por Sutton & Barto [2018], e sobre métodos de otimização baseados em gradiente utilizados no contexto do AM (Seção 3.5.2), conforme apresentado por Goodfellow et al. [2016].

3.5.1 Redes Neurais Artificiais

RNAs são amplamente utilizadas para aproximação de funções não-lineares. Elas possuem diversos usos na área de AM, como para classificação no Aprendizado Supervisionado, ou para aproximar uma política no AR. As RNAs são formadas por três tipos de camadas: a camada de entrada, que tem uma unidade para cada informação de entrada, a camada de saída, que tem uma unidade para cada informação de saída, e as camadas ocultas, que são as demais camadas que ficam entre a entrada e a saída. Existem ligações entre todas as unidades de uma camada e as unidades da camada seguinte. Cada ligação dessas possui um peso (valor real) associado, que corresponde,

intuitivamente, à eficácia da conexão sináptica em uma rede neural. As unidades são geralmente semi-lineares, ou seja, calculam a soma ponderada de seus sinais de entrada, e então aplicam uma função não-linear, chamada função de ativação, para produzir a ativação, ou seja, a saída da unidade. Diferentes funções de ativação são utilizadas, sendo uma das mais comuns a função sigmoide dada pela Equação 3.53. É comum também o uso do retificador de não-linearidade dado por $f(x) = max(0, x)$. As ativações da camada de entrada são dadas externamente, pelos valores de entrada da função que está sendo aproximada. Pode-se dizer então que a ativação de cada unidade de saída de RNAs do tipo *feedforward* (sem loops) é uma função não-linear dos padrões de ativação sobre as unidades de entrada da rede.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.53}$$

RNAs geralmente aprendem usando algum método de gradiente estocástico. Cada peso é ajustado em uma direção visando melhorar o desempenho geral da rede, medido por uma função a ser minimizada ou maximizada. No caso do Aprendizado Supervisionado, a função é o erro sobre o conjunto de exemplos rotulados de treinamento. Já no caso de AR, as RNAs podem usar erros TD para aprender valores de função, ou podem maximizar a recompensa esperada como em métodos *Policy Gradient*. Em todos os casos é necessário estimar como uma mudança em cada conexão (peso) influenciaria o desempenho geral da rede, ou seja, estimar a derivada parcial de uma função objetivo em relação a cada peso, dados os valores atuais de todos os pesos da rede. A maneira de maior sucesso para fazer isso em RNAs com camadas ocultas (considerando que as unidades têm funções de ativação diferenciáveis) é através do algoritmo *Backpropagation* [Rumelhart et al., 1986]. Tal algoritmo consiste basicamente em passadas alternadas pra frente e pra trás em toda a rede, de forma que: cada passada pra frente calcula a ativação de cada unidade, dadas as ativações atuais das unidades de entrada e, após cada passada pra frente, uma passada pra trás calcula de forma eficiente as derivadas parciais para cada peso. É importante notar que, como é um algoritmo de aprendizado de gradiente estocástico, o vetor dessas derivadas parciais é uma estimativa do gradiente verdadeiro.

Segundo Sutton & Barto [2018], o uso de RNAs com camadas ocultas profundas tem sido responsável pelos grandes avanços recentes na área de AM. Uma rede neural com uma única camada oculta contendo um número finito e suficientemente grande de unidades de função sigmoide pode aproximar qualquer função contínua sobre uma região compacta do espaço de entrada com qualquer grau de precisão. Apesar dessa propriedade de "aproximação universal" das RNAs com uma única camada oculta,

teoria e prática mostram que a aproximação de funções complexas para muitas tarefas de Inteligência Artificial são mais fáceis (e na verdade precisam) de abstrações que são composições hierárquicas de muitas camadas de abstrações de baixo nível. Ou seja, abstrações produzidas por arquiteturas profundas tais como RNAs com muitas camadas ocultas. As camadas sucessivas de uma rede profunda computam de forma crescente representações abstratas da entrada bruta, com cada unidade fornecendo um atributo que contribui para a representação hierárquica da função geral de entrada-saída da rede. Portanto, treinar as camadas ocultas de uma RNA é uma forma de criar automaticamente atributos apropriados para um dado problema, de forma que representações hierárquicas podem ser obtidas sem depender exclusivamente de *features* projetadas à mão. Por isso, RNAs com camadas ocultas têm recebido tanta atenção.

O algoritmo *Backpropagation* tem bons resultados para redes rasas com uma ou duas camadas ocultas, mas pode não funcionar em RNAs profundas. De fato, treinar com mais camadas pode produzir resultados piores do que com menos e uma das explicações é que redes mais profundas podem causar *overfitting*, falhando em generalizar para casos não treinados. Outro fator é que as derivadas parciais podem cair rapidamente no sentido das entradas, gerando treinamento muito lento; ou subir rapidamente gerando aprendizado instável. A criação de métodos para lidar com esses problemas são largamente responsáveis por muitos dos bons resultados das RNAs profundas (Sutton & Barto [2018]).

Em relação ao problema de *overfitting*, diferentes técnicas são utilizadas para amenizar o problema. Entre elas podem ser destacadas: parar o treinamento quando o resultado começa a piorar em dados de validação diferentes dos de treinamento (*cross validation*), modificar a função objetivo para desencorajar complexidade na aproximação (regularização), introduzir dependências entre os pesos para reduzir o número de graus de liberdade (ex: *weight sharing*) e remoção aleatória de unidades e suas conexões durante o treinamento (método de *dropout*). Outro problema recorrente com as RNAs é que, como o método de atualização dos pesos é iterativo, o ponto de onde começamos faz diferença. Uma forma de amenizar essa questão é iniciar os pesos aleatoriamente com uma função de valores baixos, e baixa variância (ex: $\mathcal{N}(0,1)$). É importante ter valores diferentes na mesma camada, caso contrário, eles produzirão as mesmas saídas. Além disso, pode ser interessante escalar de acordo com o número de entradas (ex: $w = \frac{N(0,1)}{\sqrt{\#entradas}}$. Sutton & Barto [2018] citam ainda outras técnicas para melhorar o desempenho do treinamento em RNs profundas, como uma técnica de encontrar bons valores iniciais dos pesos através de treinamento não supervisionado de uma camada oculta por vez, ou o uso de *batch normalization* (normalização de dados para média zero e variância um) também entre camadas ocultas.

Um tipo de RNA que obteve resultados expressivos, inclusive no AR, foram as rede convolucionais profundas. Tais redes são especializadas em processar dados de muitas dimensões arranjados em *arrays* espaciais, como imagens. Elas se baseiam em como funciona o início do processamento visual no cérebro e, devido à sua arquitetura especial, elas podem ser treinadas usando *backpropagation* padrão, sem as técnicas citadas anteriormente para melhorar os resultados. Tais redes possuem camadas de convolução e subamostragem. A ideia da camada convolucional é usar o mesmo peso para diferentes unidades de forma que elas capturem a mesma *feature* em diferentes partes da imagem, formando um *feature map*. A camada pode ter vários *features maps*. A ideia da camada de subamostragem é reduzir a resolução da entrada, de forma a direcionar que as *features* tenham os mesmos resultados independente de onde se encontram as porções da imagem. Tais características levam a resultados bem melhores quando as redes convolucionais são utilizadas em espaços de estados que representam imagens.

3.5.2 Métodos de Gradiente Estocástico

Nesta seção tratamos de métodos de otimização baseados em gradiente. O objetivo é apresentar uma visão geral sobre tais métodos, especialmente métodos de gradiente estocástico, no contexto de AM. O texto desta seção é baseado em Goodfellow et al. [2016].

3.5.2.1 Otimização Baseada em Gradiente

Muitos dos métodos de aprendizado profundo envolvem algum tipo de otimização. O objetivo de uma tarefa de otimização é maximizar ou minimizar uma função objetivo $f(x)$ alterando o valor de x. Podemos simplificar e considerar que o problema é sempre de minimização, uma vez que maximizar $f(x)$ é o mesmo que minimizar $-f(x)$. O método *gradient descent* pode ser usado para reduzir $f(x)$ movendo x por pequenos passos no sentido oposto ao sinal da derivada da função. Um mínimo local é um ponto no qual $f(x)$ é menor que todos seus pontos vizinhos, de forma que não é mais possível diminuir $f(x)$ por passos infinitesimais. Já um mínimo global é um ponto para o qual se obtém o menor valor de $f(x)$. No contexto de aprendizado profundo, é comum que as funções a serem otimizadas tenham muitos mínimos locais que não são ótimos globais e muitos pontos de sela rodeados por grandes platôs, o que dificulta a tarefa de otimização, especialmente no caso de funções multidimensionais. Portanto, em termos práticos, métodos de aprendizado profundo geralmente buscam um (bom) valor baixo para $f(x)$ mas não necessariamente mínimo.

Os problemas de interesse em aprendizado profundo geralmente consideram uma função com múltiplas entradas, $f : \mathbb{R}^n \to \mathbb{R}$. Para tais funções usamos derivadas parciais $\frac{\partial}{\partial x_i} f(x)$, que medem o quanto a função f muda se somente a variável x_i aumenta no ponto x. O gradiente é uma generalização da noção de derivada para o caso onde a derivada se refere a um vetor, ou seja, o gradiente de f (escrito como $\nabla_x f(x)$) é o vetor contendo todas as derivadas parciais. Neste contexto, o método *gradient descent* (ou *steepest descent*), consiste em reduzir o valor de f movendo-se na direção do negativo do gradiente. Ou seja, o método define um novo ponto $x' = x - \epsilon \nabla_x (fx)$, onde ϵ é a taxa de aprendizado, um escalar positivo que define o tamanho do passo. O método converge quando todos os elementos do gradiente são iguais a zero (ou, em termos práticos, bem próximos de zero). O método é limitado a otimização em espaços contínuos, mas o conceito geral de fazer pequenos movimentos repetidamente, que aproximam o melhor pequeno movimento, em direção a melhores valores pode ser também generalizado para espaços discretos.

Muitos dos algoritmos de otimização usados em aprendizado profundo são aplicáveis a uma grande variedade de funções, mas ao preço de não oferecer quase nenhuma garantia de otimalidade. No caso de funções convexas existem algoritmos que fornecem muitas garantias, mas muitos dos problemas de aprendizado profundo são difíceis de serem expressos em termos de otimização convexa.

3.5.2.2 Stochastic Gradient Descent (SGD)

Quase todos os métodos de aprendizado profundo usam alguma variante do algoritmo **Stochastic Gradient Descent (SGD)**, que é uma extensão do algoritmo de *gradient descent* apresentado anteriormente. Em AM é muito comum que se utilize grandes conjuntos de dados para se obter uma boa generalização, mas o problema é que tratar todos estes dados é computacionalmente caro. A função de custo usada em métodos de aprendizado muitas vezes pode ser decomposta com uma soma sobre exemplos de treinamento de alguma função de custo (*loss function*), $\mathcal{L}$, dada para cada exemplo. Considerando uma função de custo aproximada por pesos θ, o cálculo do gradiente exige computar uma função como a apresentada na Equação 3.54. Mas o custo computacional de tal operação é $O(m)$, onde m é a quantidade de exemplos. Portanto, quanto maior o tamanho da base de dados mais inviável se torna o tempo necessário para se calcular um único passo de gradiente.

$$\nabla_\theta J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \mathcal{L}(x^{(i)}, y^{(i)}, \theta) \tag{3.54}$$

A ideia central do SGD é que o gradiente é um valor esperado e, portanto, pode ser aproximado utilizando-se uma pequena quantidade de amostras. Assim, a cada passo do algoritmo, podemos utilizar um minilote de exemplos $\mathcal{D} = \{x^{(1)}, \cdots, x^{(m')}\}$ amostrados uniformemente do conjunto de treinamento. O tamanho do minilote, m', é geralmente um valor pequeno entre um e algumas centenas e o gradiente pode ser então estimado conforme apresentado na Equação 3.55, usando exemplos do minilote $\mathcal{D}$. O algoritmo SGD segue então o gradiente estimado no sentido que decresce a função objetivo, usando a Equação 3.56, onde ϵ é a taxa de aprendizado.

$$g = \frac{1}{m'} \nabla_\theta \sum_{i=1}^{m'} \mathcal{L}(x^{(i)}, y^{(i)}, \theta) \tag{3.55}$$

$$\theta \leftarrow \theta - \epsilon g \tag{3.56}$$

No passado, a aplicação de SGD em problemas de otimização não convexa não era vista com bons olhos, mas hoje sabemos que os modelos de AM funcionam muito bem quando treinados com tal método. Embora o algoritmo não tenha garantia de se chegar nem mesmo a um mínimo local em um tempo razoável, ele geralmente encontra valores muito baixos para a função de custo rápido o suficiente para ser útil. Vale ressaltar que os algoritmos de otimização usados para treinamento de modelos de aprendizado são diferentes dos algoritmos de otimização tradicionais em vários aspectos. Primeiramente, enquanto na otimização "pura", minimizar uma função de custo J é o próprio objetivo em si, no caso de muitos cenários de AM, a minimização da função de custo é uma forma indireta de se tentar melhorar uma medida de desempenho. Ou seja, reduzimos o valor da função de custo, na esperança que isso melhore o desempenho. Além disso, algoritmos de aprendizado geralmente incluem alguma especialização na estrutura das funções objetivo utilizadas. Por fim, vale notar que enquanto na otimização pura o objetivo é executar o algoritmo até se alcançar pelo menos um mínimo local, no caso do aprendizado a execução do algoritmo é feita até que algum critério de parada (*early stopping*) é alcançado. Isso é feito para evitar o problema de *overfitting* e, assim, o método é interrompido mesmo quando a função ainda tem valores não-nulos e razoavelmente grandes para as derivadas. O que é diferente da otimização, a qual considera que o algoritmo converge quando o gradiente se torna nulo (ou muito pequeno).

3.5.2.3 SGD com *Momentum*

O algoritmo SGD apresentado na seção anterior é muito utilizado, mas pode ser lento em alguns casos. O método de ***momentum*** foi criado para acelerar o aprendizado, se movendo na direção de um acúmulo da média móvel, com queda exponencial, dos gradientes anteriores. O algoritmo utiliza uma variável v que funciona como uma "velocidade", ou seja, é a direção e a rapidez com que os parâmetros se movem através do espaço de parâmetros. A velocidade é ajustada como uma média exponencialmente decrescente do negativo do gradiente. Um parâmetro $\alpha \in [0, 1)$ indica o quão rapidamente as contribuições dos gradientes anteriores caem exponencialmente. A regra de atualização do algoritmo é dada pela Equação 3.57.

$$v \leftarrow \alpha v - \epsilon \nabla_\theta \Big(\frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(f(x^{(i)}, \theta), y^{(i)}) \Big), \theta \leftarrow \theta + v \tag{3.57}$$

A velocidade v acumula os elementos do gradiente e quanto maior o valor de α em relação a ϵ, mais os gradientes anteriores afetam a direção atual. O Algoritmo 9 apresenta o método SGD com *momentum*.

Algoritmo 9: SGD com *momentum*

Parâmetros:
    Taxa de aprendizado ϵ, *momentum* α, pesos θ e velocidade inicial v
while *critério de parada não é atendido* **do**
    Amostra um minilote de m exemplos do conjunto de treinamento $\{x^{(i)}, \cdots, x^{(m)}\}$ e os alvos correspondentes $y^{(i)}$
    $g \leftarrow \frac{1}{m} \nabla_\theta \sum_i \mathcal{L}(f(x^{(i)}, \theta), y^{(i)})$ ▷ Calcula a estimativa do gradiente
    $v \leftarrow \alpha v - \epsilon g$ ▷ Calcula a velocidade da atualização
    $\theta \leftarrow \theta + v$ ▷ Aplica a atualização

No SGD original, o tamanho do passo era simplesmente a norma do gradiente multiplicada pela taxa de aprendizado. Já no SGD com *momentum*, o tamanho do passo depende do quanto uma sequência de gradientes é grande e alinhada. Dessa forma, se muitos gradientes sucessivos apontam exatamente na mesma direção, o tamanho do passo é maior. Já se a direção muda, é como se o algoritmo voltasse ao funcionamento normal do SGD.

No caso de métodos de gradiente estocásticos utilizados para atualização de RNAs, taxa de aprendizado é um dos hiperparâmetros mais difíceis de se ajustar, devido ao quanto ele afeta o desempenho do modelo aprendido. Isso porque geralmente o custo é muito sensível a algumas direções no espaço de parâmetros mas insensível a outras. Apesar do algoritmo de *momentum* conseguir mitigar um pouco essa questão, ele

faz isso acrescentado um novo hiperparâmetro a ser ajustado. Alguns métodos foram então propostos, partindo da ideia de se utilizar taxas de aprendizado diferentes para cada parâmetro, adaptando-as automaticamente durante o processo de aprendizado. Alguns desses métodos são apresentados nas seções a seguir.

3.5.2.4 AdaGrad

O método **AdaGrad** adapta a taxa de aprendizado de cada parâmetro do modelo escalando-os de forma inversamente proporcional à raiz quadrada da soma dos valores quadrados de todos os gradientes anteriores. As taxas de aprendizado referentes aos parâmetros com os maiores valores de derivadas parciais da função de custo caem mais rapidamente do que aquelas referentes a parâmetros com menores valores de derivadas parciais. O efeito geral disso é caminhar mais nas direções mais suavemente inclinadas do espaço de parâmetros. O Algoritmo 10 apresenta tal método.

Algoritmo 10: AdaGrad

Parâmetros:
    Taxa de aprendizado global ϵ e pesos θ
    Constante δ para estabilidade numérica ▷ valor sugerido: 10^{-7}
Inicializa as variáveis de acumulação do gradiente $r = 0$
while *critério de parada não é atendido* **do**
    Amostra um minilote de m exemplos do conjunto de treinamento $\{x^{(i)}, \cdots, x^{(m)}\}$ e os alvos correspondentes $y^{(i)}$
    $g \leftarrow \frac{1}{m} \nabla_\theta \sum_i \mathcal{L}(f(x^{(i)}, \theta), y^{(i)})$ ▷ Calcula a estimativa do gradiente
    $r \leftarrow r + g \odot g$ ▷ Acumula os gradientes quadrados
    $\nabla\theta \leftarrow \frac{\epsilon}{\delta + \sqrt{r}} \odot g$ ▷ Calcula a atualização (element-wise)
    $\theta \leftarrow \theta + \nabla\theta$ ▷ Aplica a atualização

O algoritmo tem propriedades teóricas interessantes no contexto de otimização convexa, mas quando usado para treinar modelos de RNAs, em termos práticos, pode resultar em decréscimo prematuro da taxa de aprendizado efetiva. Dessa forma, ele alcança bons resultados em alguns, mas não todos, modelos de aprendizado.

3.5.2.5 RMSProp

O algoritmo **RMSProp** parte da mesma ideia do AdaGrad, mas utiliza uma média móvel exponencialmente ponderada para tratar a acumulação do gradiente. O método AdaGrad foi projetado para convergir rapidamente no caso de funções convexas. No caso do treinamento de uma RNA, apesar da função ser não convexa, a trajetória de

treinamento pode passar por diferentes estruturas e, eventualmente, alcançar uma região que é localmente convexa (*convex bowl*). O problema com o AdaGrad é que ele diminui a taxa de aprendizado considerando todo o histórico de gradientes quadrados e assim a taxa pode ter diminuído muito antes de se alcançar tal estrutura convexa. O método RMSProp, apresentado no Algoritmo 11, usa uma média que cai exponencialmente para descartar os pontos mais antigos de forma que ele consiga convergir rapidamente ao alcançar uma região convexa. De certa forma, podemos dizer que ele funciona como uma instância do AdaGrad inicializada dentro da região convexa. Uma desvantagem é que ele utiliza um novo hiperparâmetro ρ, que controla a escala da média móvel. De todo modo, o algoritmo RMSProp tem se mostrado efetivo como um algoritmo de otimização de RNAs profundas em problemas práticos.

Algoritmo 11: RMSProp

Parâmetros:
    Taxa de aprendizado global ϵ, taxa de decaimento ρ e pesos θ
    Constante δ para estabilidade numérica ▷ valor sugerido: 10^{-6}
Inicializa as variáveis de acumulação do gradiente $r = 0$
while *critério de parada não é atendido* **do**
    Amostra um minilote de m exemplos do conjunto de treinamento $\{x^{(i)}, \cdots, x^{(m)}\}$ e os alvos correspondentes $y^{(i)}$
    $g \leftarrow \frac{1}{m} \nabla_\theta \sum_i \mathcal{L}(f(x^{(i)}, \theta), y^{(i)})$ ▷ Calcula a estimativa do gradiente
    $r \leftarrow \rho r + (1 - \rho) g \odot g$ ▷ Acumula os gradientes quadrados
    $\nabla\theta \leftarrow \frac{\epsilon}{\sqrt{\delta + r}} \odot g$ ▷ Calcula a atualização (element-wise)
    $\theta \leftarrow \theta + \nabla\theta$ ▷ Aplica a atualização

3.5.2.6 Adam

O método **Adam**, apresentado no Algoritmo 12, também usa a ideia de uma taxa de aprendizado adaptativa, e seu nome vem da expressão "*adaptive moments*". Ele pode ser visto com uma espécie de RMSProp com *momentum*, mas com algumas diferenças importantes. No algoritmo Adam, o *momentum* é incorporado diretamente como uma estimativa do momento de primeira ordem do gradiente, ponderado exponencialmente. O algoritmo aplica o *momentum* aos gradientes reescolonados, mesmo que não exista uma motivação teórica clara sobre isso. Outra diferença importante é que o algoritmo inclui correções de viés para as estimativas tanto dos momentos de primeira ordem (o próprio termo de *momentum*) quanto dos momentos de segunda ordem para se levar em conta como eles foram inicializados. Uma das vantagens do Adam é que ele costuma

ser bem robusto em relação à escolha padrão dos valores dos hiperparâmetros, embora a taxa de aprendizado precise ser, às vezes, modificada.

Algoritmo 12: ADAM

Parâmetros:
    Tamanho de passo ϵ ▷ valor sugerido: 0.001
    Taxas de decaimento $\rho_1, \rho_2 \in [0, 1)$ ▷ valores sugeridos: 0.9 e 0.999
    Pesos θ
    Constante δ para estabilidade numérica ▷ valor sugerido: 10^{-8}
Inicializa as variáveis $s = 0, r = 0$
while *critério de parada não é atendido* **do**
    Amostra um minilote de m exemplos do conjunto de treinamento $\{x^{(i)}, \cdots, x^{(m)}\}$ e os alvos correspondentes $y^{(i)}$
    $g \leftarrow \frac{1}{m} \nabla_\theta \sum_i \mathcal{L}(f(x^{(i)}, \theta), y^{(i)})$ ▷ Calcula a estimativa do gradiente
    $t \leftarrow t = 1$
    $s \leftarrow \rho_1 s + (1 - \rho_1) g$ ▷ Atualiza estimativa enviesada do primeiro momento
    $r \leftarrow \rho_2 r + (1 - \rho_2) g \odot g$ ▷ Atualiza estimativa enviesada do segundo momento
    $\hat{s} \leftarrow \frac{s}{1 - \rho_1^t}$ ▷ Corrige viés do primeiro momento
    $\hat{r} \leftarrow \frac{r}{1 - \rho_2^t}$ ▷ Corrige viés do segundo momento
    $\nabla\theta \leftarrow -\epsilon \frac{\hat{s}}{\delta + \sqrt{\hat{r}}} \odot g$ ▷ Calcula a atualização (element-wise)
    $\theta \leftarrow \theta + \nabla\theta$ ▷ Aplica a atualização

Como apresentado por Goodfellow et al. [2016], não há uma resposta clara sobre qual dos algoritmos apresentados é o melhor. Embora existam artigos comparando tais algoritmos, em termos práticos a escolha costuma depender mais da familiaridade do usuário com o algoritmo, o que costuma facilitar o ajuste dos hiperparâmetros.

Como é apresentado mais adiante, nós utilizamos o algoritmo Adam nos experimentos da presente tese.

Capítulo 4

Solução Usando Algoritmo PPO

4.1 Introdução

Neste capítulo apresentamos como o problema abordado nesta tese foi primeiramente resolvido utilizando-se o algoritmo PPO. O algoritmo foi escolhido por alcançar bons resultados em diversos problemas, especialmente aqueles com espaços de ação contínuos. Uma primeira aplicação do algoritmo foi feita para resolver uma versão mais simples do problema abordado. Nesta versão, as demandas são sempre regulares (não-sazonais) e os tempos de espera constantes, e, além disso, o algoritmo foi aplicado em um único cenário utilizado como estudo de caso. Tal aplicação foi descrita em um artigo científico [Alves & Mateus, 2020] apresentado na ICCL 2020, *International Conference on Computational Logistics*, e é também apresentada no Anexo A da presente tese. Preferimos focar este capítulo nos experimentos feitos posteriormente por duas razões principais. A primeira porque eles foram feitos considerando a versão completa do problema, como ele foi apresentado no Capítulo 2. E a segunda razão é que experimentamos o algoritmo em 17 cenários, o que permite fazer uma avaliação mais completa do desempenho do algoritmo. Os resultados de tais experimentos foram descritos em um artigo científico [Alves & Mateus, 2021] submetido para a revista *Annals of Operations Research*, e que encontra-se, no momento da escrita deste texto, em processo de revisão. Depois de termos aplicado o PPO com sucesso para resolver o problema, partimos para a comparação com outros algoritmos *Policy Gradient*, e tal comparação é apresentada no Capítulo 5.

Este capítulo é organizado como se segue. Na Seção 4.2 apresentamos a metodologia utilizada para aplicar o algoritmo PPO para resolver o problema. São apresentados os cenários experimentais, as adaptações na formulação PDM (especialmente a normalização de espaços e ações), o agente PL usado como *baseline* nos experimen-

tos e a metodologia experimental composta por três fases (ajuste de hiperparâmetros, treinamentos e avaliação dos melhores modelos). Já na Seção 4.3 apresentamos os experimentos realizados e discutimos os resultados encontrados. São apresentados os resultados gerais, a avaliação da construção dos estoques com sazonalidade, os tipos de custo, e as curvas de aprendizado do algoritmo. Por fim, na Seção 4.4, são apresentadas as conclusões do presente capítulo.

4.2 Metodologia

Nesta seção, descrevemos a metodologia utilizada para resolver o problema de cadeia de suprimentos com demandas sazonais e tempos de espera incertos que tratamos neste capítulo. Na Seção 4.2.1, apresentamos os 17 cenários usados nos experimentos para avaliar se o algoritmo PPO é adequado para resolver o problema. A forma de aplicação do algoritmo é apresentada na Seção 4.2.2. O objetivo principal é apresentar a normalização dos valores dos estados e ações que usamos para obter melhores resultados com o algoritmo. Na Seção 4.2.3, descrevemos como o agente PL é construído e usado como um *baseline* nos experimentos. Na Seção 4.2.4, apresentamos como o modelo PL com informação perfeita é usado para calcular limites inferiores para cada experimento. Finalmente, as três fases da metodologia experimental (ajuste de hiperparâmetros, treinamento e avaliação) são apresentadas na Seção 4.2.5.

4.2.1 Cenários Experimentais

Nós consideramos vários cenários para avaliar se o algoritmo PPO é adequado para resolver o problema proposto. Os parâmetros que são comuns a todos os cenários são apresentados na Tabela 4.1, seguindo a notação do modelo PL (Seção 2.4). Todos os custos são aplicados por unidade de matéria-prima ou produto. Os valores iniciais de material e as capacidades foram definidos de forma que fizessem sentido com a faixa de valores de demanda.

Os cenários foram projetados para que tivéssemos variedade em termos de tipos de demanda (sazonal ou regular), nível de incerteza das demandas, e tipos de tempo de espera (estocástico ou constante). Nos cenários com demanda sazonal, os valores de demanda para cada varejista são gerados usando funções senoidais e de perturbação. A função senoidal S, apresentada na Equação 4.1, gera dados com comportamento sazonal, onde min e max representam, respectivamente, os valores mínimo e máximo da curva, z é o número de picos da função e t é o período em questão. Adiciona-se ao valor da função senoidal o valor de uma função de perturbação P (definida

Tabela 4.1. Parâmetros comuns a todos os cenários

Grupo	**Param.**	**Valor**	**Detalhes**
Cadeia	q	8	2 fornecedores, 2 fábricas, 2 distribuidores e 2 varejistas
	f_n	1	para fábricas (0 para os outros nós)
	r_n	3	razão de processamento para fábricas (1 para os demais nós)
Horizonte	h	360	tamanho do episódio
Custos	c_n^s	1	custos de estoque para todos os nós
	c_n^p	6,4	custo de produção para cada fornecedor, respectivamente
	c_n^f	12,10	custo de processamento para cada fábrica, respectivamente
	c^t	2	custo de transporte em toda a cadeia
Custos Pen.	c^e	10	custo por material descartado por exceder capacidade de estoque
	c^d	216	custo incorrido por demanda não atendida
Capacidades	b_n^p	600, 840	capacidade de produção para cada fornecedor, respectivamente
	b_n^f	840, 960	capacidade de processamento para cada fábrica, respectivamente
	b_n^s	6400, 7200	capacidade de estoque para cada fábrica, respectivamente
	b_n^s	1600, 1800	capacidade de estoque para cada par de nó no mesmo estágio
Valores iniciais	s_n	800	nível de estoque para todos os nós
	p_{in}	600,840	material que estará disponível nos períodos $1, ..., l^{avg}$ em cada fornecedor
	t_{inm}	600,840	material que chegará nos períodos $1, ..., l^{avg}$ em cada fábrica
	t_{inm}	240,240	idem, mas para distribuidores e varejistas

para cada cenário) parametrizada por um nível de incerteza p, como mostrado na Equação 4.2. Nesta equação, foram utilizados os valores $min^{sin} = 100$ e $max^{sin} = 300$ como mínimo e máximo da função senoidal, e $min = 0$ e $max = 400$ como os menores e maiores valores possíveis de demanda, respectivamente. Se removermos o termo de perturbação da equação, obtemos demandas sazonais determinísticas, que poderiam ser vistas como valores de demanda previstos. No caso de cenários com demandas regulares (não-sazonais), os valores de demanda são gerados por $d^{avg} + P(p)$, onde $d^{avg} = 200$.

$$S(min, max, z, t) = min + \frac{max - min}{2}\Big[1 + \sin\Big(\frac{2.z.t.\pi}{h}\Big)\Big] \tag{4.1}$$

$$D = \text{clip}\Big(S(min^{sin}, max^{sim}, z, t) + P(p), min, max\Big) \tag{4.2}$$

Em relação aos cenários com tempos de espera estocásticos, os valores de tal parâmetro para cada nó são amostrados de uma distribuição de Poisson, dada por $min(Poisson(l^{avg} - 1) + 1, l^{max})$, onde $l^{avg} = 2$ é o tempo de espera médio e $l^{max} = 4$ o tempo de espera máximo. Esta forma foi utilizada para se evitar tempos de espera com valor zero e para manter os valores de tempos de espera no intervalo $[1, 4]$. Em cenários com tempos de espera constantes, usamos o valor médio $l^{avg} = 2$.

A Tabela 4.2 apresenta os cenários experimentais propostos, destacando suas diferenças. Os cenários do conjunto A foram projetados para verificar o comportamento do agente PPO considerando tempos de espera estocásticos e diferentes níveis de incerteza para as demandas sazonais. O conjunto B é similar ao primeiro conjunto, mas considerando tempos de espera constantes. Em ambos os conjuntos a incerteza dos valores de demanda (perturbação) é dada por uma distribuição Gaussiana (Normal), com média zero e desvio padrão p. Os valores de p, de 0 a 60, foram escolhidos para representar diferentes níveis de incerteza, desde nenhuma até incertezas mais altas. O valor de p foi limitado a 60 para assegurar que os valores de demanda, embora incertos, continuassem tendo comportamento sazonal. A Figura 4.1 apresenta exemplos de demandas para os cenários *N20* e *N60*. As linhas sólidas pretas são os valores da função senoidal, sem a perturbação. As linhas pontilhadas em cinza representam os desvio padrão da perturbação ($\mathcal{N}(0, 20)$ e $\mathcal{N}(0, 60)$, respectivamente). Os pontos azuis mostram exemplos de valores de demanda para um varejista.

Tabela 4.2. Cenários experimentais e suas diferenças. O cenário *N20stc* é igual ao *N20* exceto que os custos de estoque são [1,2,1,2,5,6,5,6].

Conj.	**Cenário**	**Demandas Sazonais**	**Função de Pert.**	**Pert.** p	**Tempos de Espera Estocásticos**
	`N0`	✓	*nenhuma*		✓
`A`	`N20`	✓	$\mathcal{N}$	20	✓
	`N40`	✓	$\mathcal{N}$	40	✓
	`N60`	✓	$\mathcal{N}$	60	✓
	`N0cl`	✓	*nenhuma*		
`B`	`N20cl`	✓	$\mathcal{N}$	20	
	`N40cl`	✓	$\mathcal{N}$	40	
	`N60cl`	✓	$\mathcal{N}$	60	
	`rN0`		*nenhuma*		✓
`C`	`rN50`		$\mathcal{N}$	50	✓
	`rN100`		$\mathcal{N}$	100	✓
	`rU200`		$\mathcal{U}$	[-200,200]	✓
	`rN0cl`		*nenhuma*		
`D`	`rN50cl`		$\mathcal{N}$	50	
	`rN100cl`		$\mathcal{N}$	100	
	`rU200cl`		$\mathcal{U}$	[-200,200]	
E	`N20stc`*	✓	$\mathcal{N}$	20	✓

A Tabela 4.2 também apresenta os conjuntos C e D, que contêm cenários com demandas regulares, considerando tempos de espera estocásticos e constantes, respec-

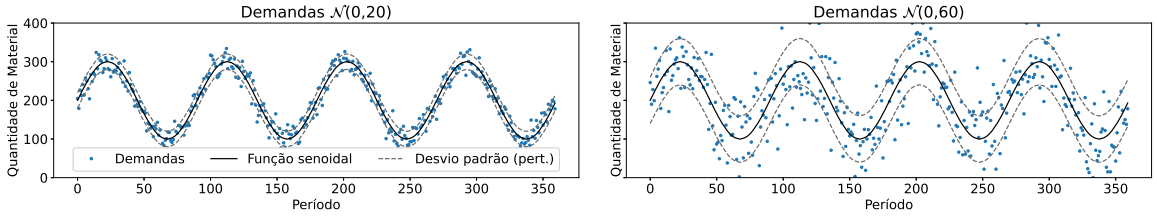

Figura 4.1. Exemplo de demandas para os cenários `N20` e `N60`: linhas sólidas pretas são a função senoidal (representando valores esperados), linhas pontilhadas cinza representam o desvio padrão da perturbação ($\mathcal{N}(0, 20)$ e $\mathcal{N}(0, 60)$), e os pontos azuis mostram exemplos de demanda para um varejista.

tivamente. As funções de perturbação podem ser dadas por distribuição Gaussiana ou Uniforme. No caso de cenários nos quais as demandas são geradas por distribuição Uniforme, os valores de demanda são dados por $200 + \mathcal{U}([-200, 200])$, o que seria o mesmo que dizer que elas são amostradas uniformemente a partir do intervalo $[0, 400]$. Desta forma, os cenários dos conjuntos C e D foram projetados considerando um nível crescente de incerteza para as demandas. Finalmente, uma variação nos custos de estoque é avaliada com o cenário do grupo E.

4.2.2 Aplicação do Algoritmo PPO

Nós escolhemos inicialmente o algoritmo PPO para resolver o problema pelo fato dele alcançar bom desempenho em problemas com espaços de ação contínuos de muitas dimensões. O primeiro passo para aplicar o algoritmo é implementar a simulação da operação da cadeia de suprimentos, ou seja, implementar o ambiente. Uma possível abordagem seria implementar o ambiente seguindo exatamente a formulação PDM apresentada na Seção 2.3. Mas o PPO, e outros algoritmos *Policy Gradient*, geralmente obtêm melhores resultados se os valores de estado e ação são normalizados ou reescalonados para o intervalo $[-1, 1]$. Isto acontece porque muitos destes algoritmos utilizam uma distribuição Normal que inicialmente tem média zero e desvio padrão igual a um. Desta forma, no caso das ações, por exemplo, se a faixa de valores considerada for muito grande, as ações geradas inicialmente pelos algoritmo serão todas ao redor de zero e o espaço de busca não será explorado suficientemente. Já se os valores forem muito próximos de zero, por exemplo entre -0,02 e 0,02, a maior parte das ações será maior do que os limites. Portanto, o uso de uma faixa com mesma magnitude da distribuição Normal utilizada inicialmente pelos algoritmos tende a encontrar melhores resultados [Raffin et al., 2019]. De fato, em experimentos preliminares, nós tentamos aplicar o PPO sem considerar a normalização, e também usando normalização

automática (considerando médias móveis em tempo de execução), mas a metodologia apresentada a seguir obteve melhores resultados. Além desta questão, a metodologia aqui proposta mostra como nós tratamos as ações de forma a gerarem sempre estados viáveis. À primeira vista isso pode parecer um mero detalhamento, mas na verdade aqui estão descritas decisões que foram fundamentais para a obtenção de bons resultados com o algoritmo utilizado (como, por exemplo, a forma utilizada para garantir que a capacidade de processamento será sempre respeitada).

Os valores de estado são divididos por um limite máximo e então reescalonados do intervalo $[0, 1]$ para o intervalo $[-1, 1]$. Seguindo a notação usada para o modelo PL, seja b_n^s a capacidade de estoque do nó n, b_n^p a capacidade de produção de um fornecedor n, d^{max} o maior valor possível de demanda, l^{max} o maior valor possível de tempo de espera e h o horizonte de planejamento. A partir da definição dos estados apresentada na Seção 2.3, os limites máximos usados na normalização são:

- b_n^s para o nível de estoque atual de cada nó.
- Para cada fornecedor:
  - b_n^p para matéria-prima sendo produzida e que estará disponível no próximo período.
  - $b_n^p * (l^{max} - 1)$ para a soma da matéria-prima que está sendo produzida e que estará disponível depois do próximo período.
- Para cada outro nó:
  - $b_l^s + b_m^s$ para o material em transporte que chegará no nó n em questão no próximo período e que foi enviado pelos nós antecessores l e m.
  - $(b_l^s + b_m^s) * (l^{max} - 1)$ para a soma das quantidades de material que chegará no nó n em questão depois do próximo período e que foi enviado pelos nós antecessores l e m.
- d^{max} para demandas de cliente de cada varejista.
- h para o número de períodos restantes até o final do episódio.

A Tabela 4.3 apresenta a normalização para alguns dos valores do exemplo de estado apresentado na Figura 2.1. Neste exemplo, consideramos $b_n^s = 500$, $b_n^p = 400$, $l^{max} = 4$ e $h = 360$. A primeira coluna indica o tipo de valor e a segunda e a terceira colunas mostram o período e os nós relacionados, respectivamente. A coluna *Valor* mostra o valor real na simulação da cadeia de suprimentos, a coluna *Máx* o maior valor

possível, a coluna *N[0,1]* apresenta os valores divididos pelo maior valor, e, finalmente, a última coluna mostra os valores reescalonados para o intervalo $[-1, 1]$. Os valores da última coluna são usados como entrada para o algoritmo PPO.

Tabela 4.3. Exemplo parcial de normalização de um estado para alguns dos valores apresentados na Figura 2.1. Os valores reais da simulação da cadeia de suprimentos são apresentados na coluna *Valor* e os valores usados como entrada para o algoritmo PPO são mostrados na última coluna.

Tipo	**Período**	**Nó**	**Valor**	**Máx.**	**N[0,1]**	**S[-1,1]**
Estoque	t	Fornecedor1	400	500	0,800	0,600
Matéria-prima	$t+1$	Fornecedor2	330	400	0,825	0,650
sendo produzida	após $t+1$	Fornecedor2	105	1200	0,088	-0,825
Material	$t+1$	Fábrica1	280	1000	0,280	-0,440
em transporte	após $t+1$	Fábrica1	420	3000	0,140	-0,720
Demandas de clientes	$t+1$	Varejista1	138	400	0,345	-0,310
Períodos restantes	t		330	360	0,916	0,833

Em relação aos valores das ações, a saída do algoritmo PPO é um vetor cujos valores estão no intervalo $[-1, 1]$. Estes valores são reescalonados para o intervalo $[0, 1]$ e então transformados nos valores realmente usados na simulação da cadeia de suprimentos. A partir da definição das ações apresentada na Seção 2.3, a mudança do intervalo $[0, 1]$ para os valores realmente utilizados é feita como se segue. Para as decisões sobre quanto produzir em cada fornecedor, o valor da ação é multiplicado pela capacidade do fornecedor (b_n^p). Já para as decisões sobre quanto transportar, seja a_{nm} e a_{no} os valores de ação representando a quantidade de material a ser transportada pelo nó n para seus nós sucessores m e o. Se o nó n não for uma fábrica, os valores de ação são primeiramente multiplicados pelo nível atual de estoque do nó n (dado por S_{in}), de forma que obtemos $a'_{nm} = a_{nm}S_{in}$ e $a'_{no} = a_{no}S_{in}$. Já se n for uma fábrica, é necessário um tratamento para garantir que a capacidade de processamento da fábrica seja respeitada. Para isso, os valores de ação são primeiramente multiplicados pelo mínimo entre o estoque atual da fábrica S_{in} e sua capacidade de processamento b_n^f, de forma que obtemos $a'_{nm} = a_{nm}min(S_{in}, b_n^f)$ e $a'_{no} = a_{no}min(S_{in}, b_n^f)$. Os valores a'_{nm} e a'_{no} calculados são usados para definir os cortes mínimo e máximo no material em estoque no nó n, dados, respectivamente, por $c^{min} = min(a'_{nm}, a'_{no})$ e $c^{max} = max(a'_{nm}, a'_{no})$. O valor c^{min} indica a quantidade de material a ser entregue ao nó k, tal que $c^{min} = a'_{nk}$. Para o outro nó, a quantidade de material é dada por $c^{max} - c^{min}$. O

material restante $S_{in} - c^{max}$ é mantido no estoque do nó n. Com esta abordagem[1], todos os possíveis valores de saída gerados pelo PPO representam valores de ação viáveis na simulação da cadeia de suprimentos.

Para ilustrar como os valores de ação são tratados, vamos usar algumas das decisões do exemplo de ação apresentado na Figura 2.2, considerando que elas foram tomadas após se observar o estado de exemplo apresentado na Figura 2.1. No exemplo, a decisão sobre quanto produzir no `Fornecedor1` é 210. Este valor teria vindo de um valor de saída $a = 0.050$ do PPO, o qual seria reescalonado para o intervalo $[0, 1]$ como $a' = \frac{a+1}{2} = 0.525$. E, finalmente, o valor da decisão seria calculado como $a' b_n^p = 0.525 * 400 = 210$. A Figura 4.2 mostra como lidar com as decisões de transporte a partir do nó `Fábrica1`. Os valores de saída do PPO teriam sido 0.492 e -0.864 para `Distribuidor1` e `Distribuidor2`, respectivamente. Estes valores seriam reescalonados para o intervalo $[0, 1]$ como 0.746 e 0.068, e então multiplicados pelo nível de estoque atual ($b_n^s = 15 + 280 = 295$) da `Fábrica1`, chegando-se assim aos valores 220 e 20, respectivamente. O valor mínimo $c^{min} = 20$ indica que a quantidade de material a ser entregue para o `Distribuidor2` é 20. Para o `Distribuidor1` a quantidade é $c^{max} - c^{min} = 220 - 20 = 200$ unidades. O material restante, $b_n^s - c^{max} = 295 - 220 = 75$, seria mantido em estoque.

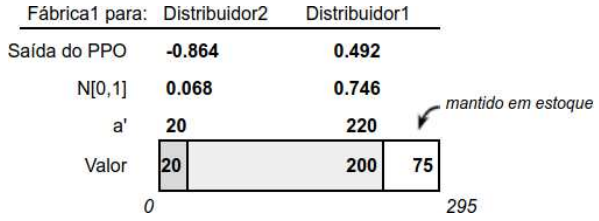

Figura 4.2. Exemplo de valores de ação relacionados a quanto transportar da `Fábrica1` para os distribuidores. Os valores de saída do PPO são reescalonados para o intervalo $[0, 1]$ e então multiplicados pelo nível de estoque da `Fábrica1`. Os valores resultados são ordenados e vistos como cortes no estoque da `Fábrica1`.

Foram realizados experimentos exploratórios para tomar algumas outras decisões sobre a forma com usamos o algoritmo PPO. Primeiro, nós decidimos usar normalização automática de recompensas (considerando médias móveis em tempo de execução). Como mencionado para os espaços de estados e de ações, é recomendável normalizar também as recompensas [Raffin et al., 2019], e nossos experimentos preliminares confirmaram que esta abordagem obtém melhores resultados com o PPO no problema

[1]Poderíamos ter feito uma definição mais direta, sem necessidade de ordenar os cortes nos estoques se incluíssemos a decisão de quanto estocar nas ações. Mas isso aumentaria o espaço de ações de 14 para 20 dimensões nos cenários experimentados, o que tornaria o problema mais complexo.

abordado. Nós também experimentamos fornecer retorno para o agente somente no final do episódio, ou seja, considerar custos iguais a zero em todos os períodos, exceto no último (que teria como recompensa o custo acumulado total). A motivação é que o objetivo é minimizar o custo total de operação, não importando como os custos são alocados ao longo do horizonte de planejamento. A desvantagem é que as recompensas se tornam muito esparsas, podendo comprometer o aprendizado do agente. Os resultados não foram melhores com esta abordagem, e, portanto, mantivemos as recompensas como os custos incorridos a cada período.

A metodologia proposta poderia ser adaptada para funcionar em problemas com espaços de estado e de ação discretos ou com capacidades ilimitadas. No caso de problemas discretos, uma possível abordagem seria manter os estados e ações como valores contínuos na visão do agente, mas arredondar os valores de ação antes de usá-los na simulação da cadeia de suprimentos. Uma outra possível abordagem seria usar espaços de estado e de ação discretos, já que o PPO também funciona nessa configuração, mas acreditamos que a primeira abordagem obteria melhores resultados. No caso de capacidades ilimitadas, uma simulação poderia ser usada para encontrar limites superiores para os valores de estado e ação que fizessem sentido com a faixa de valores de demandas dos clientes. E, assim, tais limites superiores poderiam ser usados para normalizar os valores em uma forma similar a que fazemos em nossa metodologia.

4.2.3 O *Baseline*: Agente PL

Nós decidimos usar um agente baseado em PL como um *baseline* para nossos experimentos. Embora trabalhos relacionados geralmente usem heurísticas, como (r, Q) ou estoque-base, como *baseline*, acreditamos que elas são mais adequadas para abordagens baseada em pedidos, ou quando a cadeia é linear. No nosso caso, com uma cadeia de suprimentos não-linear, não é tão simples adaptar tais heurísticas, uma vez que seria complexo definir como combinar as decisões de dois nós em cada estágio. De fato, o único trabalho relacionado que também lida com cadeias de suprimentos não-lineares [Perez et al., 2021] também usa PL como *baseline*. Além disso, como nós tratamos o problema com uma abordagem de planejamento de produção, acreditamos que um *baseline* baseado em PL, usando demandas previstas e tempos de espera médios, é uma abordagem mais usada na prática.

Como mencionado na Seção 2.4, o modelo PL pode ser resolvido de forma eficiente se nós consideramos os valores de demanda previstos e tempos de espera médios. Nos experimentos, as demandas sazonais são gerados a partir de uma função senoidal e uma perturbação, como apresentado na Seção 4.2.1. Portanto, podemos considerar a

função senoidal sem a perturbação como um valor previsto para as demandas. Nos cenários com demanda regular, o valor previsto pode ser definido como a demanda média. Assim, podemos resolver o modelo PL considerando tais valores de demanda previstos e tempos de espera médios com um *solver* de modelo PL. A solução do modelo PL é então usada para codificar o agente PL utilizado como um *baseline* nos experimentos.

O agente PL é construído a partir das variáveis de decisão P_{ijn} e T_{ijnm} do modelo (produção nos fornecedores e transporte de material, respectivamente). As outras variáveis de decisão (estoque, material processado, excesso de material e produto faltante para atender à demanda) não precisam ser codificadas no agente, já que elas serão tratadas pela própria simulação da cadeia. Os valores das variáveis de decisão utilizadas são normalizados e reescalonados para o intervalo $[-1, 1]$. Assim o agente PL pode interagir com o ambiente da mesma forma que o agente PPO.

4.2.4 Limites Inferiores: PL com Informação Perfeita

O modelo PL pode também ser usado para encontrar limites inferiores para cada cenário experimental, resolvendo o problema depois dos parâmetros terem sido realizados, ou seja, com informação perfeita. Se resolvermos o modelo usando as demandas e os tempos de espera verdadeiros, como se eles fossem conhecidos antecipadamente, obtemos então limites inferiores para os custos totais de operação. A solução do modelo é ótima considerando a realização dos valores de demanda e tempos de espera, mas ela tem um custo total menor que o valor ótimo do problema original com demandas e tempos de espera estocásticos. De qualquer forma, ela pode ser vista como um oráculo, uma referência, para os valores encontrados.

4.2.5 Metodologia Experimental

A metodologia experimental consiste em três partes: ajuste de hiperparâmetros, treinamento, e avaliação, e é ilustrada na Figura 4.3.

A primeira parte, ajuste de hiperparâmetros, é essencial para se obter melhores resultados com algoritmos *Policy Gradient*. Ela é apresentada na primeira parte da Figura 4.3. Na metodologia proposta, usamos 100 diferentes combinações (tentativas) de valores dos hiperparâmetros. Nas primeiras 20 tentativas, os valores são escolhidos aleatoriamente a partir de intervalos pré-definidos. As 80 tentativas restantes[2] usam o

[2]Em experimentos exploratórios iniciais, começamos usando valores aleatórios nas 100 tentativas. Depois vimos que o algoritmo TPE melhorava bastante os resultados. Usamos então tais experimentos exploratórios para definir a proporção 20/80 citada.

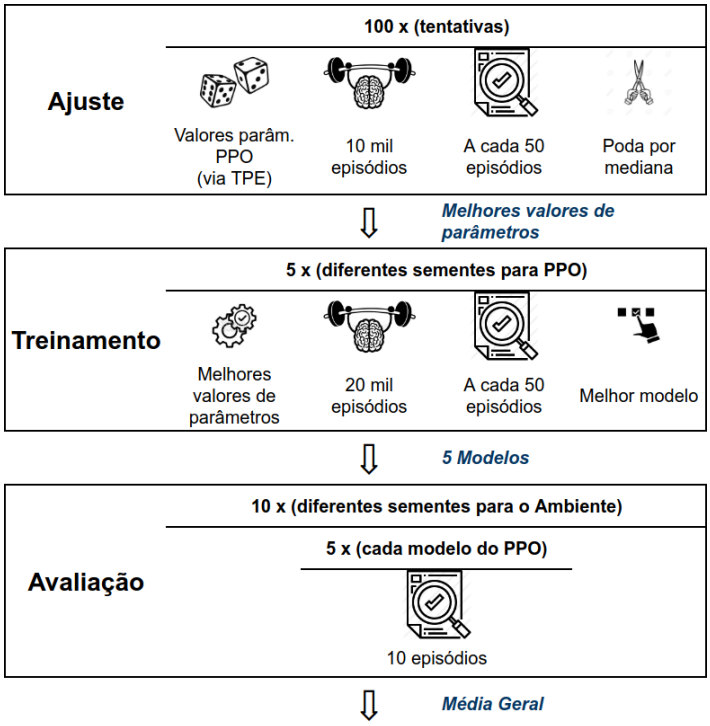

Figura 4.3. A metodologia experimental consiste em três fases: ajuste dos hiperparâmetros do PPO, treinamento usando os melhores valores encontrados para os parâmetros, e avaliação dos resultados usando os melhores modelos do PPO.

algoritmo Tree-structured Parzen Estimator (TPE) [Bergstra et al., 2011] para escolher os valores dos parâmetros. Para cada combinação de valores de hiperparâmetros, ou seja, para cada tentativa, o agente é treinado por 3,6 milhões de períodos (equivalente a 10 mil episódios), com avaliações do modelo a cada 50 episódios (ou 18 mil períodos). Cada avaliação consiste em cinco episódios, e a média do retorno (recompensa acumulada) é usada para definir a qualidade do modelo. É importante notar que a realização das demandas e dos tempos de espera é diferente em cada avaliação feita. Fazemos desta forma para evitar o *overfitting*, ou seja, para evitar que o agente aprenda a resolver bem apenas um conjunto específico de realizações de demandas e de tempos de espera. O melhor modelo encontrado em todas as avaliações é considerado como o resultado da tentativa. Um mecanismo de poda por mediana é usado nas últimas 80 tentativas. Tentativas que não parecem promissoras são interrompidas usando uma regra de poda por mediana, isto é, uma tentativa é abortada se seu resultado intermediário é pior que a mediana das tentativas anteriores. Tal processo de poda é utilizado para poupar tempo de máquina com tentativas que se mostram ruins em relação às anteriores. Finalmente, os valores dos parâmetros usados na tentativa com os melhores resultados são escolhidos para os experimentos. O ajuste de hiperparâmetros é feito

considerando-se um cenário, e os valores encontrados para os parâmetros são usados para todos os cenários experimentados.

Depois de selecionar os valores dos hiperparâmetros do PPO, a segunda parte da metodologia se refere a treinar o agente PPO (parte do meio da Figura 4.3). É importante repetir o treinamento considerando diferentes sementes de números aleatórios para assegurar a robustez dos resultados alcançados por algoritmos de AR [Henderson et al., 2018]. Portanto, nós treinamos o agente PPO cinco vezes, com diferentes valores pré-definidos de sementes, para cada cenário experimental. Cada treinamento consiste em executar o algoritmo por 7,2 milhões de períodos[3] (ou 20 mil episódios), avaliando o modelo a cada 50 episódios (ou 18 mil períodos), considerando 10 episódios em cada passo de avaliação. O melhor modelo encontrado em cada treinamento, isto é, para cada semente de números aleatórios, é usado para avaliar os resultados.

Finalmente, a avaliação dos resultados é a terceira parte da metodologia experimental. Ela é apresentada na parte debaixo da Figura 4.3. A avaliação consiste em simular 100 episódios do ambiente para cada modelo PPO encontrado no processo de treinamento. Os 100 episódios são gerados usando 10 diferentes sementes pré-definidas de números aleatórios para o ambiente. Com esta abordagem, um total de 500 episódios de avaliação é utilizado para cada cenário, e a métrica resultante é a média e o desvio padrão das recompensas acumuladas de todos os episódios.

O agente PL, apresentado na Seção 4.2.3, é usado como um *baseline* para ser comparado com o PPO. O agente é construído para cada cenário a partir da solução do modelo PL. O modelo é resolvido considerando tempos de espera médios e demandas esperadas (previstas). Como já apresentado, demandas previstas significa gerar demandas sem o termo de perturbação. Para avaliar o agente PL, usamos os mesmos 100 episódios do ambiente usados com o PPO, isto é, a mesma sequência de demandas e tempos de espera para cada episódio.

4.3 Experimentos e Resultados

Os experimentos foram realizados usando Python 3.6.10 em um computador com processador 2.9 GHz x 6, 32 GB de RAM, GPU de 6 GB, e Ubuntu Linux 20.04. A simulação da cadeia de suprimentos foi implementada em Python seguindo o padrão OpenAI Gym [Brockman et al., 2016] e o modelo PL foi resolvido usando o *solver*

[3]Nos experimentos preliminares, tentamos treinamentos com 3,6 milhões de períodos mas, para vários cenários, o modelo ainda estava sendo melhorado até o final do treinamento. Decidimos, então, treinar por 7,2 milhões de passos, e verificamos que o modelo parava de ser melhorado antes do final do treinamento.

CPLEX 12.10 via interface Python. Foi utilizada a versão do PPO da biblioteca Stable Baselines (SB3) [Raffin et al., 2019], e a biblioteca RL Baselines3 Zoo[4] [Raffin, 2020] foi usada no ajuste de hiperparâmetros. Os experimentos foram feitos inicialmente com a biblioteca Stable Baselines 2 [Hill et al., 2018], mas obtivemos melhores resultados com a SB3 nos experimentos preliminares. Embora tenhamos verificado que os valores padrões dos hiperparâmetros tenham sido a principal razão para a diferença entre os resultados das duas versões da biblioteca, escolhemos a biblioteca SB3 pelo fato de seus autores afirmarem que a implementação do PPO nesta versão é mais próxima da versão original do algoritmo.

O restante desta seção é organizado como se segue. Na Seção 4.3.1 apresentamos a fase de ajuste dos hiperparâmetros. A Seção 4.3.2 apresenta os resultados gerais para todos os cenários experimentados. A Seção 4.3.3 mostra que, para os cenários com demandas sazonais, o agente PPO consegue construir estoques, antecipadamente, com sazonalidade. A seguir, na Seção 4.3.4, a análise dos resultados é detalhada para verificar o desempenho do agente PPO em relação aos tipos de custo. As curvas de aprendizado do PPO são discutidas na Seção 4.3.5. Finalmente, na Seção 4.3.6, apresentamos um resumo dos resultados.

4.3.1 Ajuste dos Hiperparâmetros

Seguindo a metodologia proposta (Seção 4.2.5), começamos com o ajuste de hiperparâmetros usando o cenário `N20`. A Tabela 4.4 mostra os melhores valores encontrados para os hiperparâmetros do PPO. Para cada hiperparâmetro é apresentado: o tipo de amostragem (categórica, uniforme, ou log-uniforme), os possíveis (intervalos de) valores pré-definidos, o melhor valor encontrado no final do processo de ajuste, e descrição do hiperparâmetro. Em relação aos parâmetros com valores fixos, realizamos os experimentos com quatro atores (cada um com uma instância da simulação da cadeia) e o parâmetro `ent_coef` foi mantido fixo com valor zero, já que ele costuma ser utilizado apenas em problemas com espaços de ação discretos [Raffin et al., 2019]. Os possíveis valores pré-definidos foram escolhidos em experimentos preliminares. Começamos utilizando os valores padrões da biblioteca RL Baselines3 Zoo e então reduzimos as opções de alguns parâmetros para os valores que alcançaram os melhores resultados. O código da biblioteca RL Baselines 3 Zoo foi modificado para usar tais valores e também para fixar os valores da primeira tentativa[5]. Em relação à arquitetura das RNAs, utiliza-

[4]A biblioteca RL Baselines3 Zoo foi utilizada como um *wrapper* para usar a biblioteca Optuna [Akiba et al., 2019] com a SB3.

[5]A primeira das 100 tentativas foi fixada para usar os valores padrões da biblioteca SB3 dos hiperparâmetros do PPO, já que a documentação da biblioteca afirma que eles foram otimizados para

mos redes da classe *Multilayer Perceptron (MLP)* e, quanto ao algoritmo de gradiente, usamos o algoritmo *Adam* [Kingma & Ba, 2017]. Os treinamentos (segundo passo da metodologia) de todos os cenários propostos foram feitos usando os melhores valores dos hiperparâmetros encontrados no processo de ajuste.

4.3.2 Resultados Gerais

A Tabela 4.5 sumariza os resultados dos experimentos comparando os agente PPO e PL para todos os cenários. A primeira coluna mostra o conjunto de cenários, a segunda indica se as demandas são sazonais ou não, e a terceira coluna mostra se os tempos de espera são estocásticos. A quarta coluna apresenta os nomes dos cenários, a quinta e a sexta colunas os limites inferiores para os custos totais de operação (ou seja, a solução ótima se as demandas e os tempos de espera fossem conhecidos antecipadamente). As quatro colunas seguintes apresentam a média e o desvio padrão dos custos totais de operação referentes aos agentes PL e PPO, respectivamente. Por fim, as últimas duas colunas mostram o ganho do PPO sobre o agente PL (diferença e porcentagem, respectivamente).

Cenários do conjunto `A` têm tempos de espera estocásticos e demandas sazonais. Como pode ser visto na Tabela 4.5, o ganho do agente PPO sobre o agente PL vai de 7,3% a 11,2%. Em relação aos cenários do conjunto `B`, eles têm tempos de espera constantes e demandas sazonais. Neste caso, os agentes PPO e PL alcançam resultados bem próximos, com uma diferença entre -1,0% e 0,3%. É interessante notar que no cenário *N0cl*, que não tem incerteza, o resultado do agente PL é, de fato, um valor ótimo. Portanto, neste cenário o agente PPO alcança um *gap* de otimalidade de 1.0%. O agente PPO tem melhor desempenho que o *baseline* nos cenários do conjunto `C`, com demandas regulares e tempos de espera estocásticos, com uma diferença de 7,8% a 9,1%. PPO também é melhor nos cenários do conjunto `D`, que tem demandas regulares com tempos de espera constantes, com uma diferença de 2,2% a 4,7% (exceto no cenário `rN0cl`, que não tem incerteza, e portanto o agente PL alcança um valor ótimo). Finalmente, PPO é também melhor no cenário do conjunto `E`, que tem custos de estoque diferentes. Este cenário é similar ao `N20`, logo as demandas são sazonais e os tempos de espera estocásticos, e o ganho do PPO é de 8,0%.

A Figura 4.4 mostra os resultados obtidos pelos agentes PPO e PL com um intervalo de confiança de 95%. Nós usamos amostragem *bootstrapped* [Efron & Tibshirani, 1994], com 10 mil iterações e método pivotal, para gerar intervalos de confiança com relevância estatística, como sugerido por Henderson et al. [2018]. Os resultados mostram

problemas contínuos [Raffin et al., 2019].

Tabela 4.4. Melhores valores encontrados no ajuste de hiperparâmetros do PPO. A coluna S indica as opções de amostragem, que são C: categórica, U: amostrada a partir de um intervalo no domínio linear, L: amostrada a partir de um intervalo no domínio logarítmico, e -: fixo

Parâmetro	S.	Valores possíveis	Melhor valor	Descrição
`n_steps`	C	$2^5; 2^6; ...; 2^{11}$	1024	Tamanho da trajetória ao coletar dados (τ no Algoritmo 5))
`n_epochs`	C	[3; 5; 10; 20]	20	Épocas de atualização do gradiente (K no Algoritmo 5)
`batch_size`	C	[64; 128; 256; 512]	64	Tamanho do minilote para cada atualização do gradiente
`vf_coef`	U	[0; 1]	0,88331	Coeficiente da função valor (c_1 na Equação 3.37)
`clip_range`	C	[0,1; 0,2; 0,3]	0.2	Coeficiente da função objetivo cortada (ϵ na Equação 3.36)
`gae_lambda`	C	[0,9; 0,92; 0,95; 0,98; 1,0]	0,95	Fator de desconto λ para calcular GAE
`gamma`	C	[0,95; 0,98; 0,99; 0,995; 0,999; 0,9999]	0,999	Valor de γ usado para calcular GAE
`net_arch`	C	[(64,64), (128,128), (256,256)]	(64,64)	Unidades nas camadas ocultas das RNAs
`lr_schedule`	C	[constante, linear]	constante	Agenda da taxa de aprendizado
`learning_rate`	L	[0,00001; 0,01]	0,0001	Taxa de aprendizado do otimizador Adam
`activation_fn`	C	[ReLU, TanH]	TanH	Função de ativação das RNAs
`max_grad_norm`	C	[0,3; 0,5; 0,6; 0,7; 0,8; 0,9; 1; 2; 5]	0,5	Para cortar gradientes normalizados
`n_actors`	-	4	4	Número de atores (N no Algoritmo 5)
`ent_coef`	-	0	0	Coeficiente de entropia na função objetivo (c_2 na Equação 3.37)

Tabela 4.5. Resultados para todos os cenários considerados: cada conjunto de cenários indica se as demandas são sazonais ou não e se as demandas são estocásticas ou não. A tabela apresenta, para cada cenário, a média e o desvio padrão (em milhares) dos custos totais de operação para os limites inferiores, o agente PL (o *baseline*) e o agente PPO. As últimas duas colunas apresentam o ganho do PPO sobre o PL (diferença e porcentagem, respectivamente). Os números nos nomes dos cenários indicam o nível de perturbação da demanda.

Conj.	**Dem. Saz.**	**T.Esp. Estoc.**	**Cenário**	**Lim. inferior**		**Agente PL**		**Agente PPO**		**Ganho**	
				Média	σ	*Média*	σ	*Média*	σ	valor	%
	✓	✓	`N0`	8.004	27	10.298	195	9.147	125	1.151	11,2 %
A	✓	✓	`N20`	8.005	49	10.316	207	9.252	157	1.065	10,3 %
	✓	✓	`N40`	8.008	88	10.393	237	9.492	196	901	8,7 %
	✓	✓	`N60`	8.010	128	10.503	276	9.737	206	766	7,3 %
	✓		`N0cl`	7.941	0	7.941	0	8.017	7	-76	-1,0 %
B	✓		`N20cl`	7.944	42	8.226	61	8.231	80	-6	-0.1 %
	✓		`N40cl`	7.951	84	8.501	118k	8.478	162	22	0,3 %
	✓		`N60cl`	7.958	124	8.740	171	8.720	201	20	0,2 %
		✓	`rN0`	7.806	8	9.405	142	8.565	49	840	8,9 %
C		✓	`rN50`	7.804	91	9.557	257	8.811	124	746	7,8 %
		✓	`rN100`	7.808	174	9.941	388	9.104	235	837	8,4 %
		✓	`rU200`	7.817	262	10.143	486	9.219	303	924	9,1 %
			`rN0cl`	7.652	0	7.652	0	7.778	3	-126	-1,6 %
D			`rN50cl`	7.647	89	8.283	130	8.098	93	185	2,2 %
			`rN100cl`	7.666	173	8.747	240	8.402	180	345	3,9 %
			`rU200cl`	7.714	256	8.985	308	8.565	198	420	4,7 %
E	✓	✓	`N20stc`	8.706	101	12.685	249	11.673	243	1.012	8,0 %

que o PPO tem intervalos de confiança pequenos com essa metodologia, mostrando que o valor médio é representativo do desempenho do algoritmo. Os intervalos de confiança do PPO são menores que os do agente PL, mas isto é esperado, já que usamos mais dados para o PPO (uma vez que avaliamos os mesmos 100 episódios para ambos os agentes, mas para o PPO isso é feito para cada um dos 5 modelos resultantes). Em relação à distância entre os intervalos dos agentes PPO e PL, podemos notar que somente nos cenários do conjunto `B` (com tempos de espera constantes e demandas sazonais) os intervalos de confiança dos agentes têm sobreposição. Nos cenários dos outros conjuntos, a diferença é significativa, especialmente nos cenários dos conjuntos `A` e `C`. Considerando o conjunto `D` (exceto o cenário `rN0cl` que não tem incerteza), embora a diferença entre os agentes seja relativamente pequena (2,2 a 4,7%), os intervalos de confiança não se sobrepõem, logo podemos dizer que o agente PPO é melhor que o agente PL com confiança estatística.

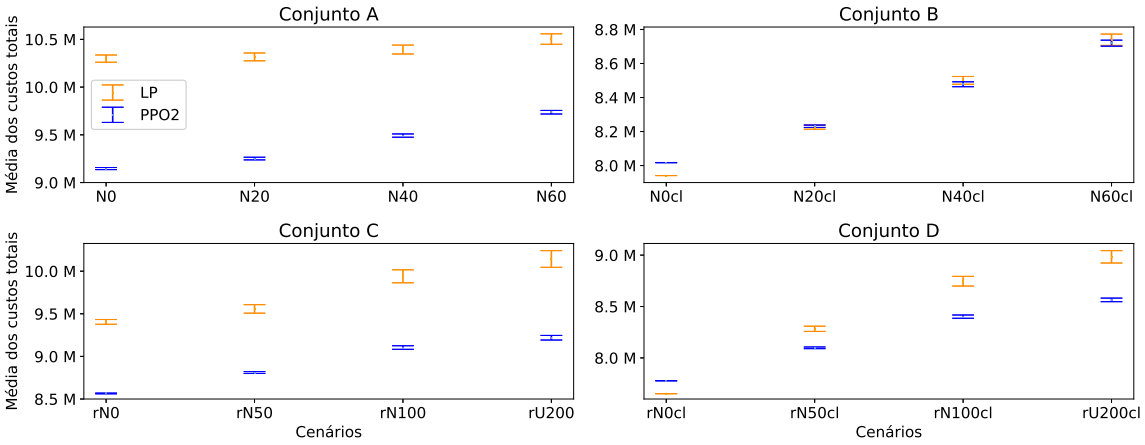

Figura 4.4. Intervalos de confiança de 95% para os resultados médios dos agentes PPO e PL, obtidos via amostragem *bootstrapped* (com 10 mil iterações e método pivotal). Os intervalos se sobrepõe somente nos cenários do conjunto `B`. Nos cenários dos outros conjuntos, a diferença entre os resultados alcançados pelo PPO e pelo agente PL são significativas.

Se olharmos os resultados com foco na comparação entre tempos de espera constantes e estocásticos, podemos notar que o PPO é uma boa ferramenta para usar com tempos de espera estocásticos. O algoritmo é melhor que o *baseline* em 9 cenários com tempos de espera incertos (com diferenças entre 7,3% e 11,2%), independentemente das demandas serem sazonais ou não. O PPO é também melhor considerando tempos de espera constantes se as demandas são regulares e com maior incerteza (diferenças entre 2,2% to 4,7%). Considerando tempos de espera constantes e demandas sazonais,

PPO alcança basicamente o mesmo nível de desempenho do *baseline* (diferenças entre -1,0% e 0,3%).

Focando nas demandas, podemos notar que os custos (e a variância) dos resultados obtidos com o agente PPO crescem com o nível de incerteza, como seria esperado. Nos cenários com demandas regulares ou tempos de espera constantes, a diferença entre os agentes PPO e PL, em geral, também cresce com a incerteza das demandas. Em cenários com demandas sazonais e tempos de espera estocásticos, a diferença entre os dois agentes cai com o aumento da incerteza das demandas.

4.3.3 Estoques com Sazonalidade

Como usamos o algoritmo PPO para resolver o problema abordado considerando cenários com demandas sazonais, é interessante avaliar se o agente consegue construir estoques com sazonalidade. A Figura 4.5 mostra diferentes tipos de custo ao longo do horizonte de planejamento referentes ao cenário `N20`. Os valores são as quantidades médias de material de todos os episódios de avaliação e se referem à soma de todos os nós da cadeia. O primeiro gráfico mostra as quantidades de material produzidas nos fornecedores, estocadas e transportadas, e o segundo gráfico as quantidades referentes às demandas não atendidas. A soma das demandas para ambos os varejistas também foi incluída no segundo gráfico para referência. Podemos notar que os estoques foram construídos seguindo o padrão senoidal da demanda com um pequeno deslocamento, mostrando que o agente PPO começa a construir os estoques antes das demandas começarem a subir. Os gráficos mostram também que as demandas não atendidas ocorrem quando os estoques atingem seus níveis mais baixos. Podemos observar que a produção nos fornecedores e o transporte de material também seguem o padrão senoidal da demanda. Uma outra avaliação é que o PPO foi capaz de diminuir a produção no final do episódio, já que as quantidades de material em produção nos fornecedores, em estoque e em transporte caem nos últimos períodos. A Figura 4.6 mostra os mesmos tipos de dados para o cenário `N20cl` e o comportamento é similar ao observado para o cenário `N20`. Comparando os cenários, podemos notar que, para tratar a incerteza dos tempos de espera, os níveis de estoque no cenário `N20` são mais altos que no cenário `N20cl`. Uma outra diferença é que a variância das curvas é maior no cenário com tempos de espera estocásticos. Finalmente, o PPO pode atender melhor às demandas dos clientes no cenário `N20cl`, com tempos de espera constantes. Outros cenários com demandas sazonais têm comportamentos similares, mas não são incluídos aqui por questão de espaço.

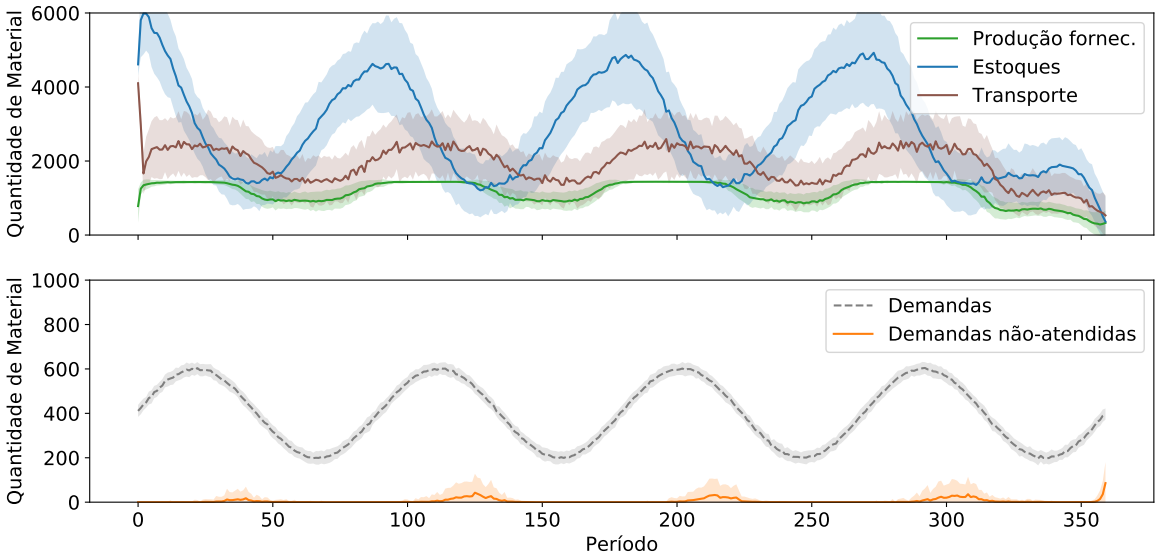

Figura 4.5. Quantidades médias de material por tipo e por período considerando todos os episódios avaliados para o cenário `N20`. Áreas sombreadas representam ± 1 desvio padrão, e os valores se referem à soma de todos os nós da cadeia. Os estoques seguem o padrão senoidal das demandas.

4.3.4 Tipos de Custo

Uma outra análise interessante é investigar quais tipos de custos são responsáveis pela diferença entre os agentes. Primeiramente, avaliamos os cenários com demandas sazonais dos conjuntos A e B, cujos resultados finais são apresentados na Figura 4.7. Como já mencionado, PPO é melhor que o agente PL nos cenários com tempos de espera estocásticos e tem basicamente o mesmo nível de desempenho quando os tempos de espera são constantes. A Figura 4.8 mostra a composição dos custos finais para cada cenário, considerando cada tipo de custo. Os valores se referem à média de todos os episódios avaliados. É importante notar que o eixo vertical de cada gráfico tem faixas de valores diferentes. Considerando tempos de espera estocásticos, podemos ver que a principal razão pela qual o PPO tem custos menores é que é mais eficiente em atender às demandas dos clientes. PPO pode atender melhor às demandas por duas razões: por produzir mais material (levando a custos maiores de operação relacionados à produção nos fornecedores, ao processamento e ao transporte de materiais); e por ter menos material descartado por excesso de materiais nos estoques (ou, dizendo de outra forma, por respeitar melhor as capacidades de estoque). Podemos também notar que, no caso do agente PPO, os níveis de estoque crescem com o aumento da perturbação da demanda.

Vamos analisar agora os cenários com tempos de espera constantes. À medida que

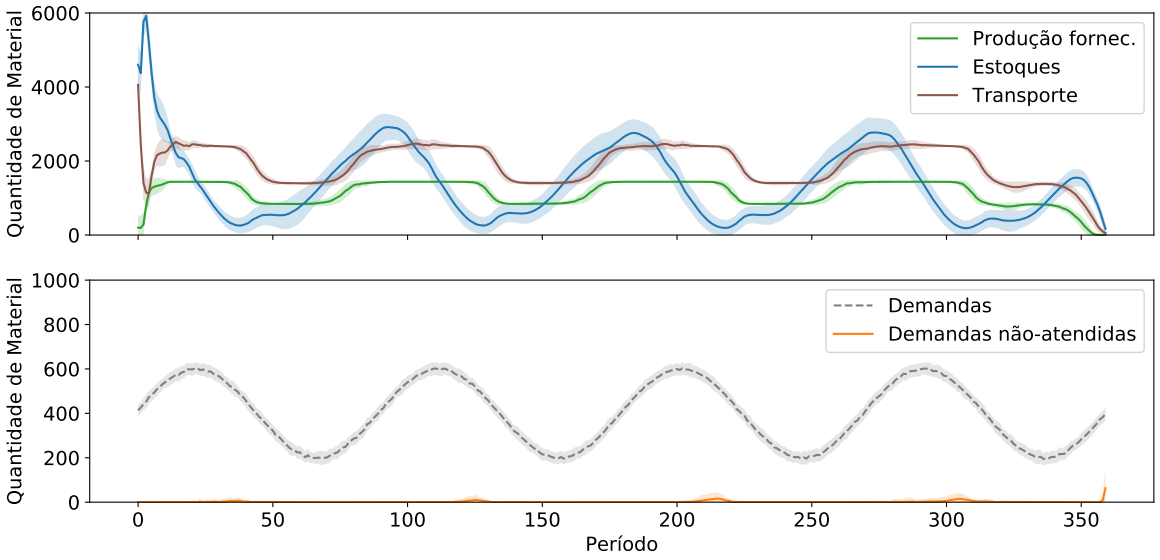

Figura 4.6. Quantidades médias de material por tipo e por período considerando todos os episódios avaliados para o cenário `N20c1`. Áreas sombreadas representam ± 1 desvio padrão, e os valores se referem à soma de todos os nós da cadeia. O comportamento é similar ao observado para o cenário `N20` na Figura 4.5, mas com níveis mais baixos de estoque e menor variância.

a incerteza da demanda cresce, ambos os agentes perdem mais demandas dos clientes, mas o PPO se torna mais eficiente que o agente PL (o PPO é pior somente no cenário no qual não há perturbação de demanda, e o agente PL tem um valor ótimo). Entretanto, para atender melhor às demandas, o agente PPO precisa construir estoques maiores, enquanto as outras operações têm custos similares. Em cenários do mundo real, com demandas incertas sazonais e tempos de espera constantes, o PPO é uma opção mais viável se o nível de incerteza das demandas for alto, ou se for difícil modelar o problema (ou ainda obter valores precisos para seus parâmetros).

O mesmo tipo de análise sobre os tipos de custos também foi feito para os cenários dos conjuntos `C` e `D`, que consideram demandas regulares. A Figura 4.9 mostra os resultados finais, ou seja, os custos totais de operação para cada cenário, comparando os agentes PPO e PL. O PPO é melhor que o *baseline* em quase todos os cenários, e quanto maior a incerteza, maior a diferença. O agente PL é melhor somente no cenário `N0c1`, que não tem incerteza e, portanto, o agente alcança um valor ótimo. A comparação em relação aos tipos de custo é apresentada na Figura 4.10. O agente PPO atende melhor às demandas dos clientes em todos os cenários e tem menores níveis de penalidades de estoque (exceto no cenário `N0c1`). O agente PPO alcança melhores resultados finais trabalhando com uma maior quantidade de material. Em relação ao estoque, o agente PPO mantém menos material que o *baseline* nos cenários

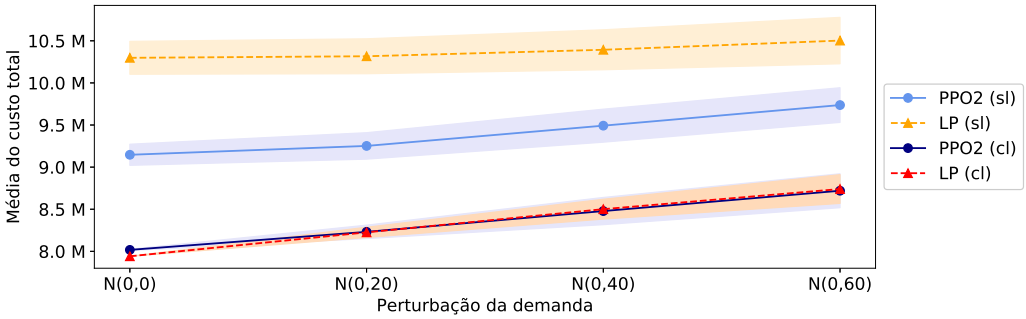

Figura 4.7. Resultados para os cenários dos conjuntos `A` e `B` (demandas sazonais). *sl* significa tempos de espera estocásticos e *cl* tempos de espera constantes. Os eixos horizontais se referem ao nível de perturbação da demanda, e os eixos verticais se referem aos custos totais de operação. Áreas sombreadas representam ± 1 desvio padrão. O agente PL é representado por linhas tracejadas e marcadores triangulares, enquanto o PPO por linhas sólidas e marcadores circulares. PPO tem resultados próximos do agente PL considerando tempos de espera constantes e é melhor com tempos de espera estocásticos.

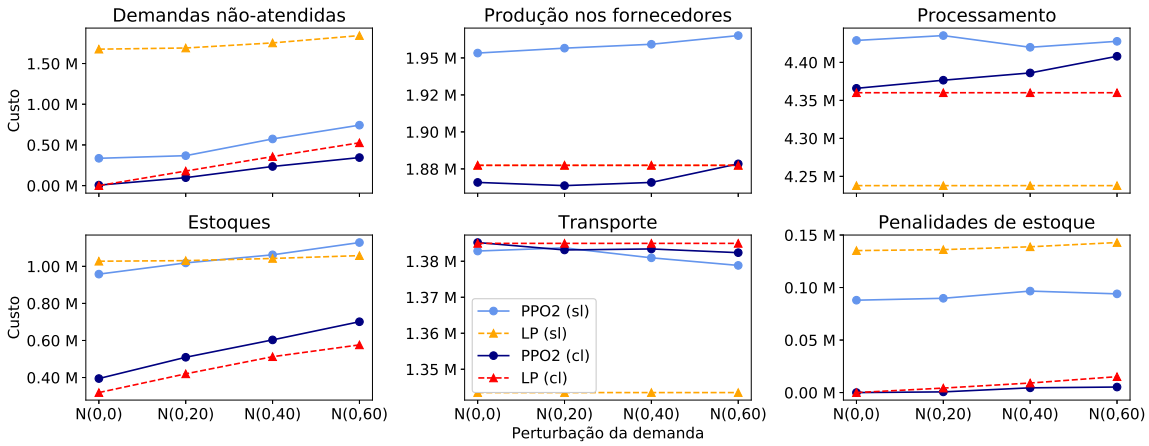

Figura 4.8. Custos por tipo para os cenários dos conjuntos `A` e `B` (demandas sazonais); *sl* significa tempos de espera estocásticos e *cl* tempos de espera constantes. Os eixos horizontais se referem ao nível de perturbação da demanda, e os eixos verticais se referem aos custos de operação por tipo. O agente PL é representado por linhas tracejadas e marcadores triangulares, enquanto o PPO por linhas sólidas e marcadores circulares.

com tempos de espera estocásticos e tem níveis mais próximos quando os tempos de espera são constantes.

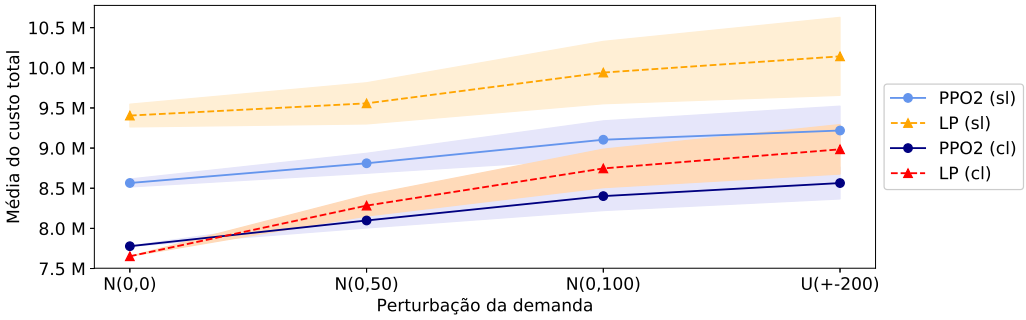

Figura 4.9. Resultados para os cenários dos conjuntos `C` e `D` (demandas regulares); *sl* significa tempos de espera estocásticos e *cl* tempos de espera constantes. Os eixos horizontais se referem ao nível de perturbação da demanda, e os eixos verticais se referem aos custos totais de operação. Áreas sombreadas representam ± 1 desvio padrão. O agente PL é representado por linhas tracejadas e marcadores triangulares, enquanto o PPO por linhas sólidas e marcadores circulares. PPO é melhor que o agente PL em todos os cenários (exceto no cenário `N0cl` no qual não há incerteza, e o agente PL alcança um valor ótimo), e a diferença é maior com tempos de espera estocásticos.

4.3.5 Curvas de Aprendizado

As curvas de aprendizado dos treinamentos do agente PPO, para o cenário `N20`, são apresentadas na Figura 4.11. Os eixos verticais se referem ao total de recompensas acumuladas por episódio, e os eixos horizontais ao número de períodos durante o treinamento. O gráfico da esquerda apresenta as curvas de aprendizado reais, de cada uma das cinco rodadas de treinamento. Como o agente PPO aprende uma política estocástica, a curva de aprendizado é um limite inferior do desempenho do algoritmo [Raffin et al., 2019]. Assim, para uma melhor avaliação de tal métrica, o gráfico da direita apresenta os valores médios relacionados às avaliações do modelo feitas durante o treinamento. Como mencionado na metodologia experimental proposta (Seção 4.2.5), o modelo é avaliado a cada 50 episódios, considerando 10 episódios a cada passo de avaliação. Portanto, os valores mostrados no gráfico da direita se referem à média desses 10 episódios para cada passo de avaliação. Podemos notar que, no começo do treinamento, as recompensas acumuladas estão entre -20 e -16 milhões, isto é, os custos totais de operação das soluções iniciais são da ordem de 16 a 20 milhões. Depois de um período inicial de exploração, os resultados começam a melhorar rapidamente, e as recompensas acumuladas chegam a -10 milhões depois de 1 milhão de períodos de treinamento. Depois deste ponto, as melhoras são mais lentas e os valores tendem a convergir depois de 4 milhões de períodos. Como apresentado, a solução final para este cenário tem valor por volta de -9,3 milhões. Estas curvas mostram que o agente PPO

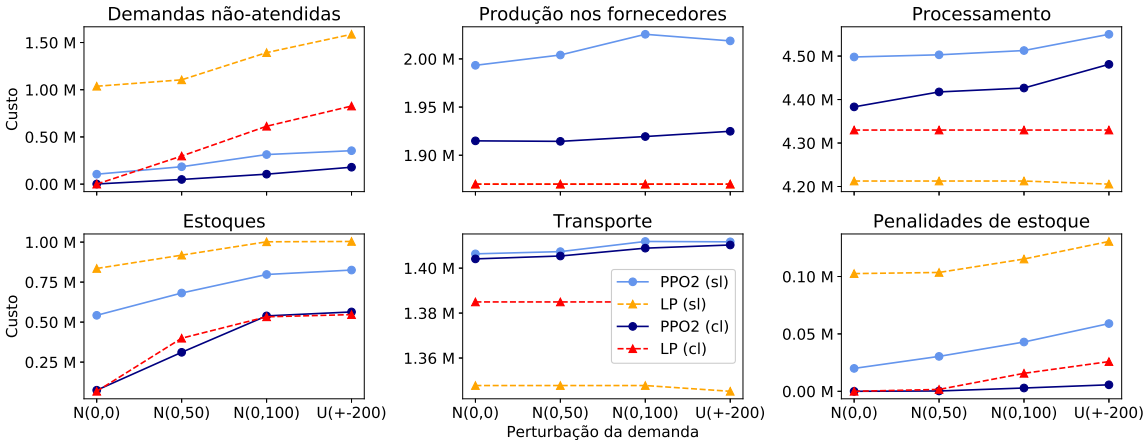

Figura 4.10. Custos por tipo para os cenários dos conjuntos `C` e `D` (demandas regulares); *sl* significa tempos de espera estocásticos e *cl* tempos de espera constantes. Os eixos horizontais se referem ao nível de perturbação da demanda, e os eixos verticais se referem aos custos de operação por tipo. O agente PL é representado por linhas tracejadas e marcadores triangulares, enquanto o PPO por linhas sólidas e marcadores circulares.

foi capaz de aprender como operar a cadeia de suprimentos a partir de uma solução praticamente aleatória. Elas mostram também que o processo de aprendizado se estabiliza antes do final do treinamento. As curvas de aprendizado para outros cenários seguem basicamente o mesmo padrão e não são apresentadas aqui por questão de espaço.

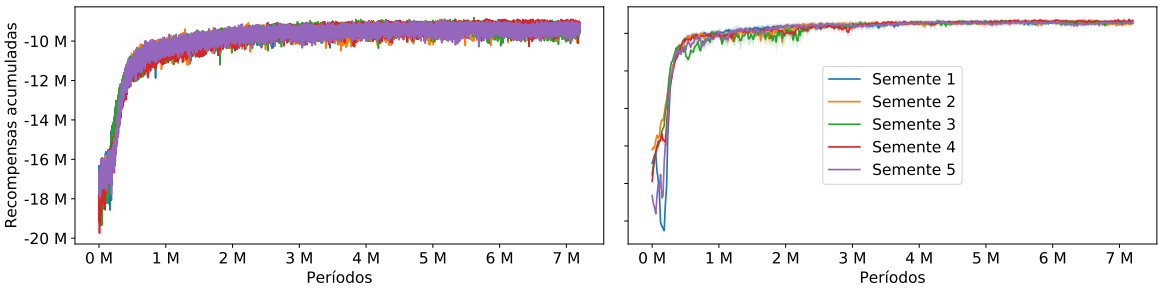

Figura 4.11. Curvas de aprendizado para o cenário `N20`. O gráfico da esquerda mostra as curvas de aprendizado das cinco rodadas de treinamento. O gráfico da direita mostra o valor médio das avaliações dos modelos realizadas durante as mesmas rodadas de treinamento.

Sobre o tempo de execução do algoritmo, em média, cada treinamento gastou menos de 220 minutos. É importante notar que, depois que o modelo de AR é treinado, a sua aplicação tem um tempo de execução mínimo. Basta fornecer o estado atual da cadeia para o modelo e a RNA fornecerá imediatamente uma saída com as decisões para o próximo período. Assim, embora o tempo de treinamento possa ter um tempo

de execução considerável, sua aplicação é rápida, o que seria uma vantagem importante em possíveis cenários de tomada de decisão em tempo-real.

4.3.6 Resumo dos Resultados

Os experimentos mostraram que o PPO pode ser uma boa ferramenta para se usar em cenários com tempos de espera estocásticos, independentemente das demandas serem sazonais ou não. Nestes cenários, o PPO foi melhor que o *baseline* (agente PL) principalmente por melhor atender às demandas incertas. Estes resultados foram alcançados operando uma maior quantidade de material e, ao mesmo tempo, respeitando melhor as capacidades de estoque. O algoritmo pode também ser útil em cenários com tempos de espera constantes e demandas não-sazonais, especialmente com maiores incertezas nas demandas. Em relação a cenários com tempos de espera constantes e demandas sazonais, o algoritmo PPO e o agente PL alcançaram resultados similares. Em tais situações, o PPO seria mais viável se fosse difícil modelar o problema ou obter valores precisos para seus parâmetros. Considerando todos os cenários, quanto maior a incerteza, maiores são os custos das soluções encontradas pelo algoritmo, como já seria esperado.

Nós também verificamos que o PPO pode construir estoques com sazonalidade. Os resultados mostraram que o agente começa a construir os estoques antes das demandas começarem a subir, e que níveis de estoque mais altos são usados no caso de tempos de espera estocásticos. Finalmente, as curvas de aprendizado mostraram que o PPO foi capaz de aprender como operar a cadeia de suprimentos, e que o processo de aprendizado se estabiliza antes do final do treinamento.

4.3.6.1 Implicações Gerenciais

Neste trabalho, endereçamos o problema de cadeia de suprimentos em uma abordagem de plano de produção, ou seja, as decisões de toda a cadeia são baseadas nas demandas dos clientes finais, como recomendado por Lee et al. [1997] para combater o efeito chicote. Há um único agente que controla todas as operações da cadeia como um tomador de decisões centralizado. Os resultados dos experimentos mostraram que, neste contexto, o algoritmo PPO pode ser uma boa escolha prática, especialmente se os tempos de espera são estocásticos, ou se as demandas têm alta incerteza. Em cenários com menor incerteza para as demandas (e tempos de espera constantes), estratégias baseadas em valores previstos ou médios podem lidar facilmente com o problema, mas quanto maior a incerteza mais difícil obter bons resultados com este tipo de abordagem. O algoritmo PPO é um algoritmo baseado em política, que é aproximada através de

uma RNA. Assim ele pode lidar com o problema sem a necessidade de agregar os valores de estado e de ações, e, portanto, pode explorar melhor o espaço de soluções. Os resultados mostraram que o algoritmo pode construir estoques com sazonalidade, minimizando o efeito chicote.

Uma outra característica do algoritmo PPO é que o modelo final (a solução) pode ser melhorada se o treinamento for continuado a partir dele. Assim pode ser interessante adaptar o modelo depois de uma mudança no cenário prático, como por exemplo, uma nova distribuição das demandas ou dos tempos de espera, ou uma modificação em alguma capacidade, etc. Finalmente, como é um método de AR Profundo *model-free*, o método de solução proposto precisa somente da simulação da cadeia de suprimentos. Isto pode ser uma vantagem em cenários nos quais seja difícil obter um modelo preciso da cadeia de suprimentos ou obter valores precisos para seus parâmetros.

4.4 Conclusões

Tomada de decisão sobre incerteza tem um forte apelo prático em gestão logística devido às inerentes complexidades envolvida no processo. Aplicações de Inteligência Artificial em problemas de planejamento de cadeias de suprimentos podem ser uma forma de melhorar a gestão logística e tem sido cada vez mais exploradas na literatura. Neste capítulo, apresentamos os resultados do uso de um algoritmo *Policy Gradient* (PPO) para resolver um problema de planejamento de produção e distribuição de produtos em uma cadeia de suprimentos multiestágio com incerteza nas demandas sazonais e nos tempos de espera. Exploramos 17 diferentes cenários em uma cadeia com quatro estágios e dois nós por estágio, considerando um horizonte de planejamento de 360 períodos. Em cada período, o agente de AR precisa decidir quanta matéria-prima produzir nos nós do primeiro estágio e a quantidade de material a ser enviada de cada nó para os nós do próximo estágio (níveis de estoque são definidos indiretamente). O objetivo é atender às demandas incertas dos clientes finais nos nós do último estágio, minimizando todos os custos incorridos (custos de operação, tais como produção nos fornecedores, estoque, transporte e processamento; e custos de penalização: se demandas não são atendidas ou capacidades de estoque são excedidas). Pelo nosso conhecimento, este trabalho é o primeiro a usar AR Profundo para lidar com o problema com um abordagem de planejamento de produção em uma cadeia de suprimentos com mais de dois estágios. Portanto, o problema resolvido tem espaços de estado e de ações com mais dimensões, sendo mais difícil de resolver que os apresentados em trabalhos relacionados. Outra contribuição é que nós fomos os primeiros a resolver o problema

com tempos de espera estocásticos usando um método de AR Profundo, mesmo se considerarmos os trabalhos relacionados que trataram do problema com uma abordagem baseada em pedidos.

Nós seguimos uma metodologia experimental robusta para verificar a qualidade e a adequação do algoritmo PPO para o problema proposto. Primeiramente, conduzimos um processo de ajuste dos hiperparâmetros para escolher os melhores valores dos hiperparâmetros do algoritmo. Depois, usamos tais valores para resolver o problema em diferentes cenários, considerando múltiplas rodadas de treinamento com diferentes sementes de números aleatórios. Finalmente, os resultados foram avaliados considerando 100 episódios para cada modelo treinado. Nós comparamos os resultados alcançados com um agente PL usado como *baseline*, construído a partir da solução de um modelo PL, considerando demandas previstas e tempos de espera médios. O agente PPO é melhor que o *baseline* em todos os cenários com tempos de espera estocásticos (de 7,3 a 11,2%), independentemente das demandas serem sazonais ou não. Em cenários com tempo de espera constante, o agente PPO é melhor quando as demandas incertas são não-sazonais (de 2,2 a 4,7%). Se as demandas incertas são sazonais e os tempos de espera constantes, PPO e PL têm basicamente o mesmo desempenho. Considerando os resultados experimentais, o PPO é uma ferramenta competitiva e adequada para o problema endereçado, e quanto maior a incerteza do cenário, maior a viabilidade deste tipo de abordagem. Uma análise detalhada sobre a construção de estoques com sazonalidade e tipos de custo também são apresentadas e discutidas (nas seções 4.3.3 e 4.3.4).

Em problemas de Pesquisa Operacional do mundo real, incertezas nos parâmetros de um modelo de planejamento são muito comuns. Como uma abordagem *model-free*, métodos de AR Profundo podem ser úteis em tais situações, o que poderia evitar excessos de capacidades e de estoques. Uma outra vantagem seria em problemas de tempo-real, nos quais o tempo de execução de um modelo de AR Profundo previamente treinado é muito rápido. No próximo capítulo, comparamos o PPO com outros algoritmos *Policy Gradient* para verificar qual deles é mais apropriado para o problema proposto. Um outro caminho possível, em trabalhos futuros, é o uso de abordagens de programação estocástica para resolver o modelo PL com parâmetros incertos e comparar com os algoritmos de AR.

Capítulo 5

Comparação de Algoritmos *Policy Gradient*

5.1 Introdução

Neste capítulo, apresentamos a aplicação e comparação de algoritmos de AR Profundo para resolver o problema tratado nesta tese. Algoritmos *Policy Gradient* estado-da-arte foram usados no problema de planejamento de produção e distribuição de produtos, considerando uma cadeia de suprimentos de quatro estágios com tempos de espera incertos, para minimizar custos totais de operação e ao mesmo tempo atender a demandas incertezas sazonais de clientes. Foram realizados experimentos em duas fases, considerando oito diferentes cenários. Na primeira fase, os algoritmos A2C, DDPG, PPO SAC e TD3 foram usados considerando treinamentos de tamanho fixo. Na fase seguintes, os dois melhores algoritmos da primeira fase tiveram seus hiperparâmetros ajustadas considerando um critério de parada diferente. Os resultados mostram que PPO e SAC têm melhor desempenho para o problema abordado. Eles têm comportamento de aprendizado, desempenho final, e complexidade amostral comparáveis, mas PPO é mais rápido em termos de tempo de execução.

O problema abordado é o mesmo descrito no capítulo anterior, e foram escolhidos oito cenários representativos dos 17 que foram lá tratados. A formulação PDM também foi mantida e usada da mesma forma para os algoritmos aqui experimentados. Preferimos não comparar aqui o *baseline* baseado no agente PL já que o foco é na comparação entre algoritmos de AR Profundo. A principal contribuição deste capítulo é a aplicação apropriada e a comparação de algoritmos *Policy Gradient* estado-da-arte para resolver um problema de cadeia de suprimentos multiestágio com dois nós por

estágio com incerteza nas demandas sazonais e nos tempos de espera. Outras contribuições são o uso de um critério de parada diferente no treinamento e de testes-t de Welch para comparação de desempenho e seleção dos algoritmos. Os resultados apresentados neste capítulo foram publicados na forma de artigo científico [Alves et al., 2021], apresentado na ICAISC 2021, *The 20th International Conference on Artificial Intelligence and Soft Computing*, ocorrida de forma *online* nos dias 21 a 23 de junho de 2021. O trabalho será incluído nos anais da conferência a serem publicados na série *Lecture Notes in Artificial Intelligence*.

O restante do capítulo é organizado como se segue. Na Seção 5.2 é apresentada a metodologia utilizada e na Seção 5.3 os experimentos e resultados alcançados. Por fim, as conclusões do presente capítulo são apresentadas na Seção 5.4.

5.2 Metodologia

Nosso principal objetivo é aplicar e comparar os algoritmos *Policy Gradient* A2C, DDPG, PPO, SAC e TD3 no problema proposto de cadeias de suprimentos. Na primeira fase, os algoritmos são usados em oito diferentes cenários experimentais usando treinamentos de tamanho fixo. Os dois melhores algoritmos são selecionados e usados na segunda fase, na qual eles têm seus hiperparâmetros ajustados considerando um novo critério de parada e são aplicados nos mesmos oito cenários (também com o novo critério de parada).

Os cenários experimentais se referem a uma cadeia de suprimentos de quatro estágios com dois nós por estágio. Eles foram selecionados daqueles apresentados no capítulo anterior (Tabela 4.2), de forma a se ter variedade em termos de tipos de demandas (sazonal e regular), tipos de tempos de espera (estocástico e constante) e nível de incerteza das demandas. Nós optamos por não utilizar todos os cenários devido ao tempo necessário para experimentação. Como será apresentado na seção de experimentos, alguns dos algoritmos têm tempo de execução considerável. As diferenças entre os cenários utilizados são apresentadas na Tabela 5.1.

5.2.1 Primeira Fase

Na primeira fase dos experimentos, os algoritmos são treinados[1] com três sementes aleatórias pré-definidas, considerando 10 mil episódios (ou 3,6 milhões de períodos). Em cada rodada de treinamento, o modelo é avaliado a cada 50 episódios, considerando o

[1]Para cada algoritmo são usados os valores padrões dos hiperparâmetros, já que eles foram ajustados considerando vários problemas contínuos de *benchmark* [Raffin et al., 2019].

Tabela 5.1. Cenários experimentais e suas diferenças

Conj.	Cenário	Demandas Sazonais	Função de Pert.	Pert. p	Tempos de espera Estocásticos
A	N20	✓	$\mathcal{N}$	20	✓
	N60	✓	$\mathcal{N}$	60	✓
B	N20cl	✓	$\mathcal{N}$	20	
	N60cl	✓	$\mathcal{N}$	60	
C	rN50		$\mathcal{N}$	50	✓
	rU200		$\mathcal{U}$	[-200,200]	✓
D	rN50cl		$\mathcal{N}$	50	
	rU200cl		$\mathcal{U}$	[-200,200]	

retorno (recompensas acumuladas) médio de 10 episódios. O modelo resultante de cada rodada é aquele com a melhor avaliação ao longo do treinamento. Uma forma comum de comparar algoritmos de AR Profundo é a análise do comportamento dos algoritmos durante o treinamento, usando curvas de aprendizado. Nós fazemos isso, mas como estamos interessados no desempenho dos algoritmos em relação a um problema específico, é também interessante comparar os resultados finais. Isto é feito calculando o retorno médio de 100 episódios (usando 10 sementes de número aleatórios pré-definidas para o ambiente) para os três melhores modelos (considerando ações determinísticas) de cada algoritmo.

Além de comparar os algoritmos, estamos também interessados em selecionar os dois com melhor desempenho para usá-los na próxima fase. Para tomar tal decisão nós também utilizamos testes-t de Welch [Colas et al., 2019], considerando os dados de avaliação das rodadas de treinamento da primeira fase. Os testes-t de Welch são mais interessantes para a comparação dos algoritmos pelo fato de não assumirem que as duas distribuições têm a mesma variância. Nós fazemos então comparações múltiplas considerando 95% de confiança e correção de Bonferroni. A principal razão para não usar todos os cinco algoritmos na segunda fase é o tempo necessário para executar todos os experimentos, especialmente para algoritmos *off-policy* (DDPG, SAC e TD3). Pode-se notar que, de certa forma, o PPO pode ser visto como uma evolução do A2C, e SAC uma evolução do TD3 (e este, por sua vez, é um DDPG melhorado). Portanto, seria esperado que os algoritmos mais novos tivessem melhor desempenho.

5.2.2 Segunda Fase

Na segunda fase dos experimentos, consideramos um critério de parada diferente, no qual um treinamento é interrompido se não há uma melhora no modelo depois de N avaliações consecutivas. A ideia é permitir que um algoritmo execute por mais tempo quando o modelo ainda está sendo melhorado e, por outro lado, poupar tempo de máquina parando o treinamento quando o algoritmo não consegue encontrar soluções melhores depois de muitos períodos. Os experimentos da primeira fase são usados para escolher o valor de N. Um outro possível critério de parada seria usar um mesmo limite de tempo de execução para os algoritmos. Mas, como nosso maior interesse é na qualidade da solução, adotamos o critério mencionado para permitir que os algoritmos continuassem executando enquanto houvesse melhora nos resultados.

Uma vez definido o valor de N, o ajuste de hiperparâmetros é feito para cada um dos algoritmos selecionados. O ajuste consiste em executar 100 treinamentos (tentativas) com diferentes combinações dos valores dos hiperparâmetros para o cenário `N20`. A primeira das tentativas usa os valores padrões para o algoritmo já que eles foram ajustados considerando vários problemas contínuos de *benchmark* [Raffin et al., 2019]. Da segunda à vigésima tentativa, os valores são escolhidos aleatoriamente de intervalos pré-definidos. O algoritmo TPE [Bergstra et al., 2011] e um mecanismo de poda por mediana são usados nas próximas 80 tentativas para, respectivamente, escolher os próximos valores dos parâmetros e interromper tentativas não promissoras. Em cada tentativa, o agente é avaliado a cada 50 episódios, pela média dos retornos de cinco episódios. Se uma tentativa não é podada, ela termina depois de N avaliações consecutivas sem melhora ou no máximo em 10 mil episódios de treinamento. Os melhores valores dos hiperparâmetros são escolhidos da tentativa com os melhores resultados.

Em seguida, a fase de treinamento consiste em usar os melhores valores encontrados para os hiperparâmetros em cinco rodadas de treinamento, com sementes de números aleatórios pré-definidas, para cada algoritmo nos oito cenários experimentais. Uma rodada de treinamento é interrompida depois de N avaliações consecutivas sem melhora ou após 20 mil episódios de treinamento. Como feito anteriormente, o modelo é avaliado a cada 50 episódios, considerando o retorno médio de 10 episódios, e o modelo final é aquele com a melhor avaliação ao longo do treinamento. A mesma avaliação dos melhores modelos e ferramentas de análise utilizadas na primeira fase dos experimentos são usadas nesta fase, mas agora considerando 5 rodadas de treinamento por cenário e algoritmo.

5.3 Experimentos e Resultados

Nos experimentos, usamos o mesmo ambiente computacional utilizado no capítulo anterior, um computador com GPU de 6 GB, processador 2.9 GHz x 6, e 32 GB de RAM. As implementações foram feitas usando Python 3.6.10. A implementação do ambiente também foi a mesma já usada anteriormente, e as versões dos algoritmos foram da biblioteca SB3 [Raffin et al., 2019]. Em relação à normalização de recompensas, apesar de ela não ser recomendada para todos os algoritmos experimentados, pudemos verificar nos experimentos exploratórios iniciais que ela melhora os resultados dos cinco algoritmos experimentados. Portanto, ela foi utilizada para todos os algoritmos. Em todos os experimentos consideramos RNAs da classe MLP e utilizamos o algoritmo *Adam* como método de gradiente. Os experimentos e resultados da primeira e segunda fase são apresentados nas seções 5.3.1 e 5.3.2, respectivamente. Já a seção 5.3.3 apresenta resultados de experimentos adicionais.

5.3.1 Experimentos da Primeira Fase

As curvas de aprendizado dos experimentos, considerando as avaliações durante o treinamento, são apresentadas na Figura 5.1. Eles mostram que o agente A2C não foi capaz de aprender, já que não converge em nenhum dos cenários. DDPG teve bons resultados em dois cenários, mas com uma grande variância entre as diferentes rodadas de treinamento nos outros. TD3 teve bons resultados em muitos cenários, mas apresentou alta variância nos cenários `rN50cl` e `rU200cl`. PPO e SAC tiveram os melhores resultados em termos de aprendizado pois suas curvas são bem comportadas em todos os cenários experimentados.

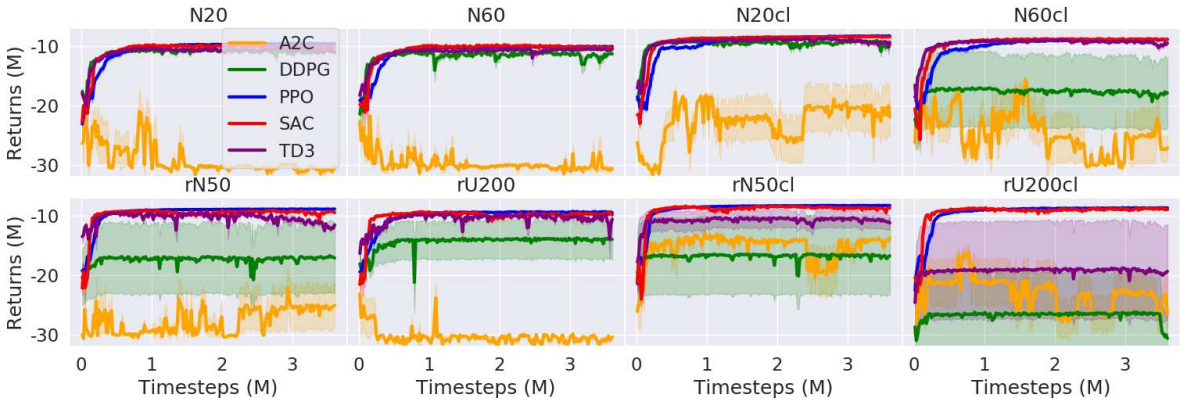

Figura 5.1. Experimentos da primeira fase: curvas de aprendizado médias por cenário e algoritmo, baseadas nas avaliações durante o treinamento. Os valores estão em milhões e as áreas sombreadas representam o erro padrão.

Nós também fizemos uma análise de comparação múltipla usando os dados coletados durante os treinamentos para selecionar os melhores algoritmos. O desempenho dos algoritmos foi sumarizado em quatro intervalos (cada um com 50 pontos de avaliação), usando testes-t de Welch, com 95% de confiança e correção de Bonferroni, para identificar o algoritmo com piores retornos médios em cada intervalo. O objetivo era encontrar os dois melhores algoritmos com confiança estatística, considerando estabilidade durante o treinamento. Os resultados são mostrados na Figura 5.2. Na primeira rodada, nós consideramos todos os cinco algoritmos, e como esperado, o agente A2C teve as piores médias em alguns cenários. Olhando as curvas de aprendizado, seria esperado que o A2C tivesse as menores médias também em outros cenários, mas isto não aconteceu devido à alta variância dos resultados do DDPG. Na segunda rodada, nós repetimos a análise mas excluindo o A2C, e nela o DDPG teve as piores médias em alguns cenários. O processo foi repetido, agora sem do DDPG, e o TD3 teve as piores médias. Portanto, essa análise reforça que PPO e SAC foram os algoritmos com melhor desempenho nos experimentos.

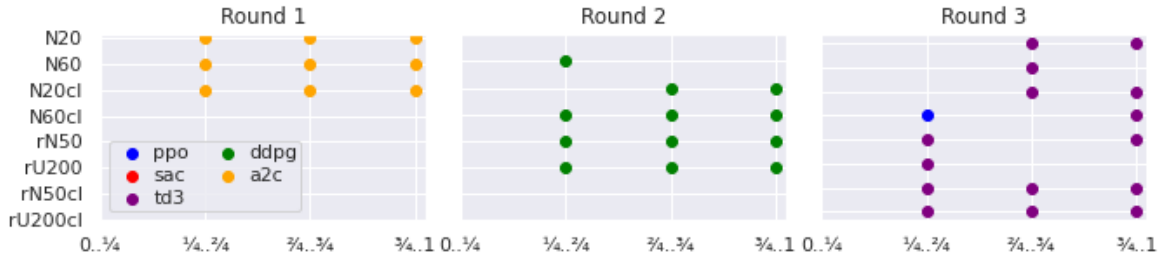

Figura 5.2. Análise de comparação múltipla por intervalos de treinamento e cenário. Marcadores indicam algoritmos com os piores retornos médios, considerando testes-t de Welch. Em cada rodada, o algoritmo com a menor média é excluído da avaliação.

A avaliação dos resultados finais (melhores modelos) é apresentada na Tabela 5.2. As linhas apresentam os cenários e cada coluna tem os resultados de um algoritmo. Para o PPO, a média dos custos totais de operação (negativo dos retornos) são apresentados. Nas outras colunas, os valores são as diferenças relativas para o resultado do PPO. Os resultados são compatíveis com o comportamento das curvas de aprendizado. SAC e PPO têm menores custos operacionais e alcançam resultados similares. TD3 tem resultados relativamente comparáveis ao PPO em vários cenários, mas custos muito altos em alguns outros. DDPG tem resultados ruins em vários cenários e A2C em todos eles.

Escolhemos os algoritmos PPO e SAC para a segunda fase dos experimentos, já que eles tiveram melhor desempenho considerando todas as métricas de avaliação:

Tabela 5.2. Experimentos da primeira fase: avaliação dos melhores modelos. Para o PPO, os valores são a média dos custos totais de operação (em milhares) e, para os outros, a diferença relativa para o PPO.

Cenário	PPO	A2C	DDPG	SAC	TD3
N20	9.534	+76,2%	+8,4%	+1,6%	+7,1%
N60	10.031	+76,5%	+6,8%	**-3,4%**	+3,1%
N20cl	8.259	+94,6%	+6,6%	+1,1%	+3,9%
N60cl	8.862	+55,3%	+91,9%	**-1,4%**	+0,7%
rN50	8.827	+94,3%	+90,5%	+0,3%	+5,9%
rU200	9.315	+109,6%	+47,1%	**-0,4%**	+3,1%
rN50cl	8.228	+46,6%	+98,9%	**-0,0%**	+23,7%
rU200cl	8.647	+56,8%	+204,2%	**-1,3%**	+117,4%

curvas de aprendizado, testes-t de Welch por intervalos de treinamento e resultados finais dos melhores modelos. Em relação aos tempos de execução dos algoritmos, a Figura 5.3 apresenta os tempos médios por rodada de treinamento. Podemos notar que o A2C e o PPO são os mais rápidos e o SAC o mais lento. Vale notar que, com as configurações que usamos para os algoritmos, o uso de um computador com GPU tem maior impacto no tempo de execução dos algoritmos *off-policy*, já que eles utilizam um *buffer* de repetição (DDPG, TD3 e SAC). Portanto, caso tivéssemos usado um computador sem GPU, a diferença no tempo de execução do A2C e do PPO para os demais seria ainda maior.

Figura 5.3. Experimentos da primeira fase: tempo médio de execução de cada algoritmo por rodada de treinamento.

5.3.2 Experimentos da Segunda Fase

5.3.2.1 Parametrização do Critério de Parada

O primeiro passo nesta fase foi a definição do valor do parâmetro N relacionado ao critério de parada proposto. A princípio, a ideia era escolher na primeira fase os três melhores algoritmos para a segunda fase dos experimentos[2]. Por isso, usamos os dados de treinamento dos três melhores algoritmos (PPO, SAC e TD3) para definir o valor do parâmetro N. A ideia seria escolher um valor que poupasse tempo de treinamento, ou seja, que terminasse o treinamento antes do tamanho máximo de 20 mil episódios, mas sem prejudicar a qualidade dos resultados. O compromisso a ser tratado é que, um valor muito grande de N pode fazer com que o treinamento não termine antes do tamanho máximo, já um valor muito pequeno pode levar a uma grande perda na qualidade dos resultados.

A estratégia que adotamos foi simular, para cada valor possível de N, o que teria acontecido com os resultados da primeira fase se tivéssemos adotado esse novo critério de parada. Para isso, usamos os dados das avaliações dos treinamentos da primeira fase (como as avaliações ocorrem a cada 50 episódios, os treinamentos de 10 mil episódios da primeira fase tiveram 200 avaliações). Com eles foi possível determinar qual teria sido o melhor modelo caso o treinamento tivesse terminado antes devido ao novo critério de parada. A partir dessas simulações, avaliamos, para cada valor de N, qual teria sido a pior perda de qualidade nos resultados por ter parado os treinamentos mais cedo, considerando os 90% melhores resultados. O objetivo era identificar o menor valor possível de N que levaria a uma perda de no máximo 1% nas avaliações dos melhores modelos, em pelo menos 90% das rodadas de treinamento. A Figura 5.4 apresenta os dados para os valores de N entre 0 e 100. O eixo horizontal apresenta os valores de N e o eixo vertical a perda de qualidade correspondente. O eixo vertical foi limitado aos valores de 0 a 2% para facilitar a visualização. Podemos perceber que, de acordo com o objetivo proposto, o valor de N a ser utilizado deveria ser 38, e resolvemos então adotar o valor arredondado de $N = 40$. Como as avaliações são feitas a cada 50 episódios, $N = 40$ significa que o treinamento é parado se não há melhorias no modelo depois de dois mil episódios (ou 720 mil períodos) de treinamento. Vale notar que, como acabamos fazendo os experimentos da segunda fase com os dois melhores algoritmos, o valor de $N = 40$ representa, na verdade, uma perda máxima de 0,7% (considerando 90% das rodadas de treinamento).

[2]Devido ao tempo necessário para execução dos experimentos e os prazos de submissão do artigo para a ICAISC 2021, acabamos depois utilizando apenas os dois melhores algoritmos

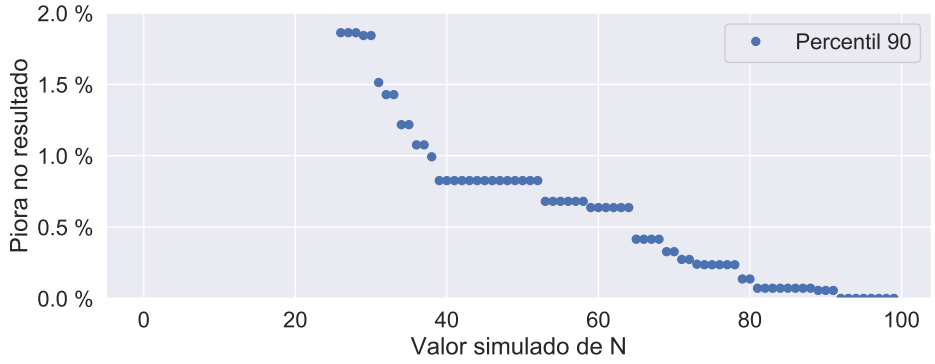

Figura 5.4. Resultados das simulações para definição do parâmetro N do critério de parada, com base nos dados da primeira fase dos experimentos. Para cada possível valor de N é a apresentada qual seria a maior perda de qualidade na avaliação dos melhores modelos, considerando 90% das rodadas de treinamento.

5.3.2.2 Ajuste dos Hiperparâmetros

O próximo passo foi o ajuste dos hiperparâmetros do PPO e SAC que seguiram a metodologia proposta, utilizando-se o cenário `N20`. Os intervalos pré-definidos para os hiperparâmetros de cada algoritmo foram escolhidos em experimentos preliminares. Começamos utilizando os valores padrões da biblioteca RL Baselines3 Zoo e então reduzimos as opções de alguns parâmetros para os valores que alcançaram os melhores resultados. Da mesma forma que fizemos no Capítulo 4, usamos a biblioteca RL Baselines3 Zoo com o código alterado para usar os valores pré-definidos e também para fixar os valores da primeira tentativa como os padrões da biblioteca Stable Baselines. As figuras 5.5 e 5.6 apresentam os resultados das tentativas de ajuste dos hiperparâmetros para os algoritmos PPO e SAC, respectivamente. O eixo horizontal apresenta as tentativas e o eixo vertical o valor do melhor modelo encontrado. Os valores dos modelos são representados pelos círculos azuis e a linha vermelha destaca o melhor valor encontrado até então. Podemos ver na figura que nas primeiras 20 tentativas, como os valores dos parâmetros são escolhidos de forma aleatória, os resultados têm uma grande variação. Já depois da vigésima tentativa, com o uso do algoritmo TPE para escolher os valores dos parâmetros, os resultados começam a se concentrar na faixa de maiores valores. Outro ponto que vale a pena destacar é que os valores dos parâmetros padrões dos algoritmos (primeira tentativa) são bons para o nosso problema já que eles têm resultado próximo dos melhores resultados encontrados.

Os melhores valores encontrados para os hiperparâmetros do PPO e SAC são apresentados na tabelas 5.3 e 5.4, respectivamente. Para cada hiperparâmetro é apresentado: o tipo de amostragem (categórica, uniforme, ou log-uniforme), os possíveis

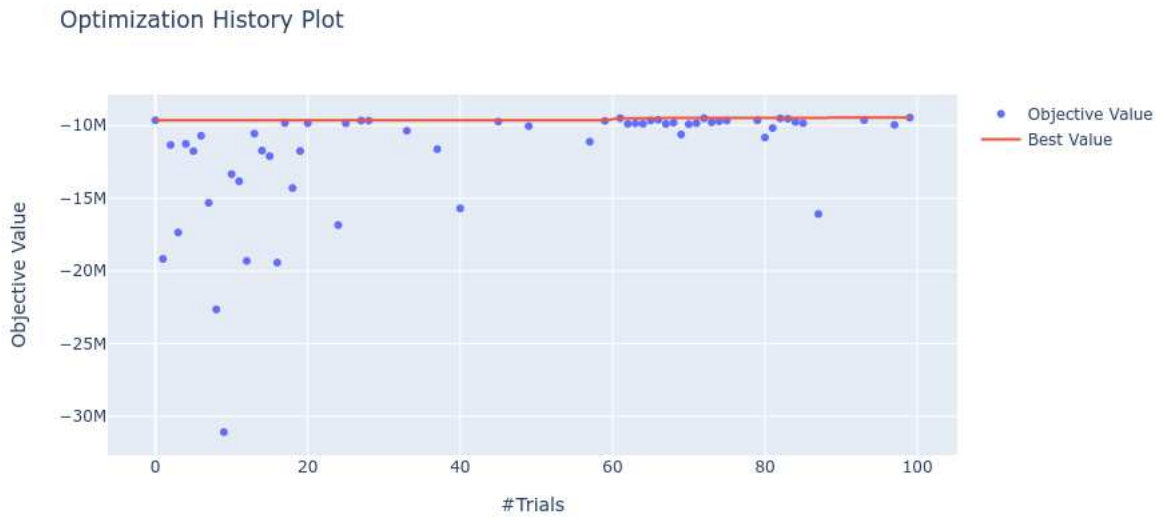

Figura 5.5. Valores dos melhores modelos encontrados para as tentativas de ajuste dos hiperparâmetros do algoritmo PPO. Tentativas podadas não são exibidas.

(intervalos de) valores pré-definidos, o melhor valor encontrado no final do processo de ajuste, e descrição do hiperparâmetro. Em relação ao tempo de execução, o ajuste para o SAC demorou 16 dias, mesmo usando o novo critério de parada e executando dois processos em paralelo. Já o ajuste para o PPO demorou menos de 48 horas.

5.3.2.3 Treinamentos

A fase de treinamento foi executada considerando cinco rodadas de cada algoritmo, com sementes pré-definidas de números aleatórios, e critério de parada de 40 avaliações consecutivas sem melhora (ou um máximo de 20 mil episódios), para cada um dos oito cenários experimentados. As curvas médias de aprendizado do PPO e SAC, considerando as avaliações feitas durante o treinamento, são apresentadas na Figura 5.7. É importante notar que, devido ao critério de parada, o tamanho de cada rodada de treinamento pode ser diferente. Portanto, a curva média considera este detalhes, e cada algoritmo pode terminar as curvas em pontos diferentes. Podemos observar que o PPO e o SAC tiveram desempenho comparável em termos de aprendizado, já que as curvas crescem rapidamente e se estabilizam em níveis semelhantes para todos os cenários. É interessante notar que, embora em muitos problemas o SAC tenha uma melhor eficiência amostral que o PPO, isso não foi observado para o problema proposto. De fato, em alguns cenários, a curva de aprendizado do PPO até cresce mais rápido que a do

Tabela 5.3. Melhores valores dos parâmetros do PPO encontrados no ajuste de hiperparâmetros. A coluna S indica as opções de amostragem, as quais são: C: categórica, U: amostrada a partir do intervalo com domínio linear, L: amostrada a partir do intervalo com domínio logarítmico, e -: fixo.

Parâmetro	**S.**	**Valores possíveis**	**Melhor valor**	**Descrição**
`n_steps`	C	$[2^5; 2^6; ...; 2^{11}]$	2^{10}	Tamanho da trajetória ao coletar dados (τ no Algoritmo 5)
`n_epochs`	C	[3;5;10;20]	20	Épocas de atualização do gradiente (K no Algoritmo 5)
`batch_size`	C	[64;128;256;512]	512	Tamanho do minilote para cada atualização do gradiente
`vf_coef`	U	[0;1]	$0,88883$	Coeficiente da função valor (c_1 na Equação 3.37)
`clip_range`	C	[0,1;0.2;0,3]	$0,1$	Coeficiente da função objetivo cortada (ϵ na Equação 3.36)
`gae_lambda`	C	[0,9;0,92;0,95;0,98;1]	$0,9$	Fator de desconto λ para calcular GAE
`gamma`	C	[0,95;0,98;0,99;0,995;0,999;0,9999]	$0,995$	Valor de γ usado para calcular GAE
`net_arch`	C	[(64,64);(128,128);(256,256)]	$(256, 256)$	Unidades das camadas ocultas das RNAs
`learning_rate`	L	[0,00001;0,01]	$0,00019$	Taxa de aprendizado do otimizador Adam
`activation_fn`	C	[ReLU,TanH]	TanH	Função de ativação das RNAs
`max_grad_norm`	C	[0,3;0,5;0,6;0,7;0,8;0,9;1;2;5]	$0,6$	Para cortar gradientes normalizados
`ortho_init`	C	[True,False]	False	Se é para usar inicialização ortogonal
`n_actors`	-	4	4	Número de atores (N no Algoritmo 5)
`ent_coef`	-	0	0	Coeficiente de entropia na função objetivo (c_2 na Equação 3.37)

Tabela 5.4. Melhores parâmetros do SAC encontrados no ajuste de hiperparâmetros. A coluna **S** indica as opções de amostrando, as quais são: C: categórica, U: amostrada a partir do intervalo com domínio linear, L: amostrada a partir do intervalo com domínio logarítmico, e -: fixo.

Parâmetro	**S.**	**Valores possíveis**	**Melhor valor**	**Descrição**
`buffer_size`	C	$[10^4; 10^5, 10^6]$	10^6	Tamanho do *buffer* de repetição $\mathcal{D}$ no Alg. 8
`batch_size`	C	$[2^6, 100, 2^8, 2^9, 2^10, 2^11]$	2^8	Tamanho do minilote B para cada atualização do gradiente
`learning_starts`	C	$[10^2, 10^3, 10^4, 2.10^4]$	10^2	Passos de coleta de transições antes de começar o aprendizado
`train_freq`	C	[1;2;10;180;360]	1	Atualiza o modelo a cada `train_freq` passos
`gamma`	C	[0,9;0,95;0,98;0,99;0,995;0,999;0,9999]	0,9999	Fator de desconto γ
`tau`	C	[0,001;0,005;0,01;0,02;0,05]	0,005	Coeficiente ρ da atualização de Polyak
`net_arch`	C	[(64,64);(256, 256);(400, 300)]	(256, 256)	Unidades das camadas ocultas das RNAs
`learning_rate`	L	[0,00001;0,01]	0,00047	Taxa de aprendizado do otimizador Adam
`ent_coef`	-	Aprendido automaticamente	-	Coeficiente de regularização de entropia (α na Equação 3.46)
`target_entropy`	-	Aprendido automaticamente	-	Entropia alvo quando aprendendo `ent_coef`
`gradient_steps`	-	Igual a `train_freq`	1	Passos de gradiente a realizar após cada coleta de dados
`activation_fn`	-	ReLU	ReLU	Função de ativação das RNAs

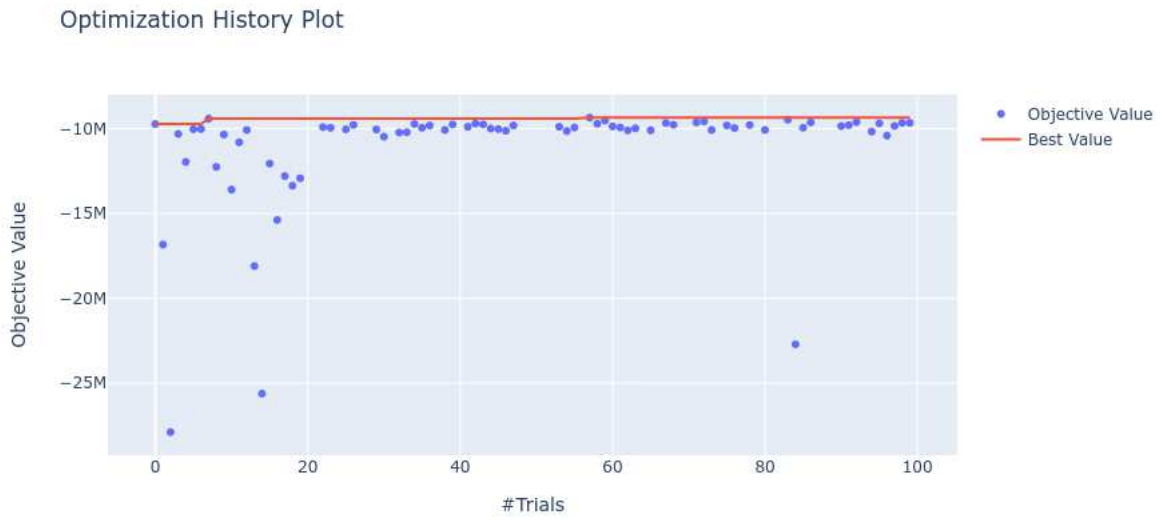

Figura 5.6. Valores dos melhores modelos encontrados para as tentativas de ajuste dos hiperparâmetros do algoritmo SAC. Tentativas podadas não são exibidas.

SAC.

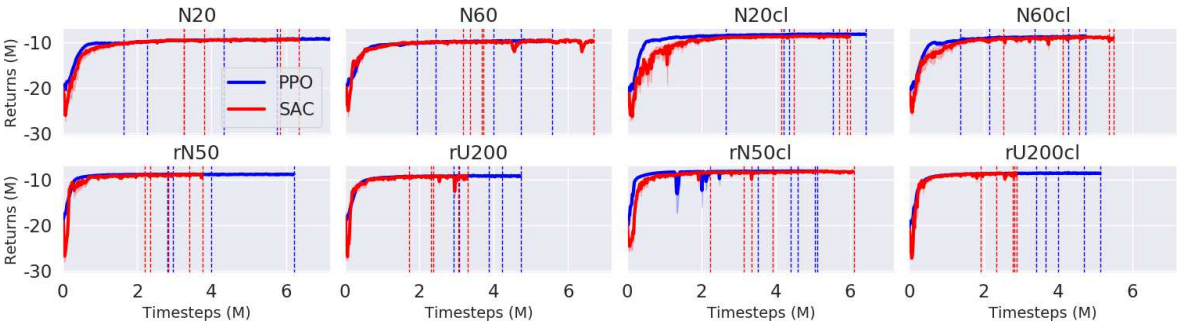

Figura 5.7. Curvas de aprendizado médias dos experimentos da segunda fase, baseadas nas avaliações feitas durante os treinamentos. Os valores estão em milhões e as áreas sombreadas representam o erro padrão. As curvas terminam antes devido ao critério de parada e as linhas tracejadas mostram quando cada rodada de treinamento terminou.

5.3.2.4 Resultados Gerais

Os resultados finais, considerando os melhores modelos, são mostrados na Tabela 5.5. Na tabela, a primeira coluna apresenta o nome do cenário, já a segunda e a terceira indicam se as demandas são sazonais e os tempos de espera são estocásticos, respectivamente. As colunas seguintes apresentam os valores médios e desvio padrão dos custos

totais de operação obtidos pelos melhores modelos dos algoritmos PPO e SAC. O menor valor entre os dois algoritmos é destacado em negrito desde que exista diferença significativa considerando os testes-t de Welch. Os mesmos resultados são também apresentados na Figura 5.8. O gráfico da esquerda mostra a média dos custos totais de operação para cenários com demandas regulares e o gráfico da direita para aqueles com demandas sazonais. Os cenários foram ordenados em cada gráfico pelo nível de incerteza, sendo os dois primeiros com tempo de espera constante e os dois seguintes com tempo de espera estocástico. Pode-se notar que, para ambos os algoritmos, quanto maior a incerteza do cenário, maiores são os custos. Considerando os testes-t de Welch, o PPO teve melhor desempenho nos cenários `rN50cl` e `N20cl`, ou seja, aqueles com menor incerteza em cada gráfico. Já o SAC foi melhor em três dos cenários com demandas sazonais. Nos outros três cenários com demandas regulares os dois algoritmos alcançaram resultados similares. Em termos de tempo de execução, cada rodada de treinamento do PPO e do SAC levaram, em média, 0,9 e 10,7 horas, respectivamente.

Tabela 5.5. Resultados da segunda fase dos experimentos. Para cada cenário é indicado se as demandas são sazonais ou não e se os tempos de espera são estocásticos ou não. A tabela apresenta a média e o desvio padrão (em milhares) dos custos totais de operação para os algoritmos PPO e SAC. Os números nos nomes dos cenários indicam o nível de perturbação da demanda.

Cenário	**Dem. Saz.**	**Tempos de Espera Estoc.**	**PPO**		**SAC**	
			Média	σ	*Média*	σ
`rN50cl`			**8.144**	103	8.275	132
`rU200cl`			8.584	203	8.575	220
`rN50`		✓	8.824	127	8.841	147
`rU200`		✓	9.170	282	9.121	296
`N20cl`	✓		**8.396**	189	8.581	104
`N60cl`	✓		9.051	355	**8.911**	225
`N20`	✓	✓	9.657	355	**9.394**	159
`N60`	✓	✓	10.009	379	**9.645**	281

5.3.2.5 Estoques com Sazonalidade

Nós também avaliamos as quantidades de material por tipo de operação ao longo do horizonte de planejamento encontrados pelas soluções do PPO e SAC, em cada cenário. Os resultados são apresentados nas figuras 5.9 e 5.10, para cenários com tempos de espera estocásticos e constantes, respectivamente. Ambos os algoritmos foram capazes de construir estoques com sazonalidade, mas o PPO trabalha com estoques um pouco

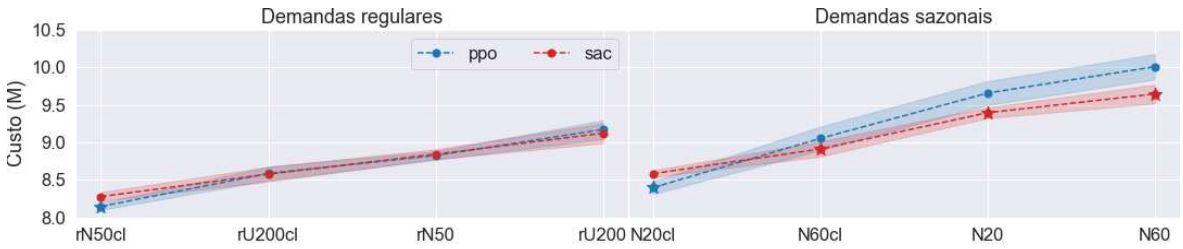

Figura 5.8. Médias dos custos totais de operação por cenário paras avaliações dos melhores modelos da segunda fase dos experimentos. Os valores estão em milhões e as áreas sombreadas representam o erro padrão. O marcador $\star$ indica se há uma diferença significativa entre os algoritmos, considerando testes-t de Welch.

menores que o SAC em todos os cenários. Como era de se esperar, ambos os algoritmos trabalham com níveis de estoques mais altos em cenários com tempo de espera estocástico. A produção de matéria-prima nos fornecedores também segue um padrão sazonal, sendo que o SAC tende a subir a produção antes do PPO. Além disso, o SAC foi melhor que o PPO em diminuir a produção ao final do episódio.

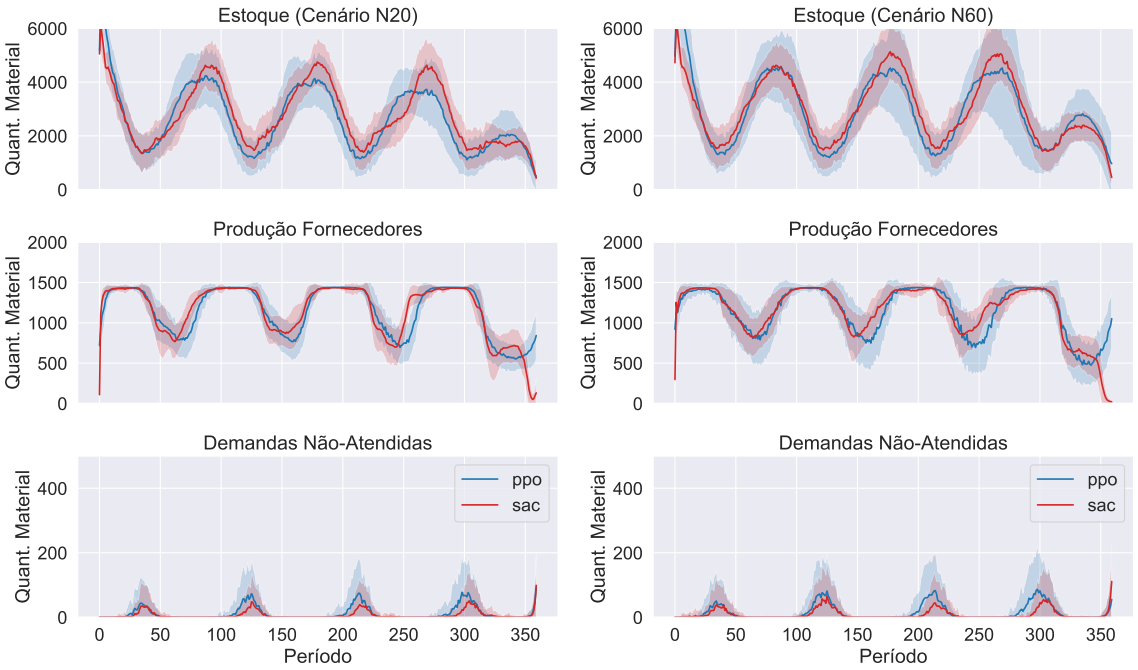

Figura 5.9. Quantidade média de material por tipo e por período considerando todos os episódios avaliados nos cenários com tempos de espera estocásticos. Áreas sombreadas representam ± 1 desvio padrão, e os valores se referem à soma de todos os nós da cadeia. Os estoques e a produção nos fornecedores seguem o padrão senoidal das demandas.

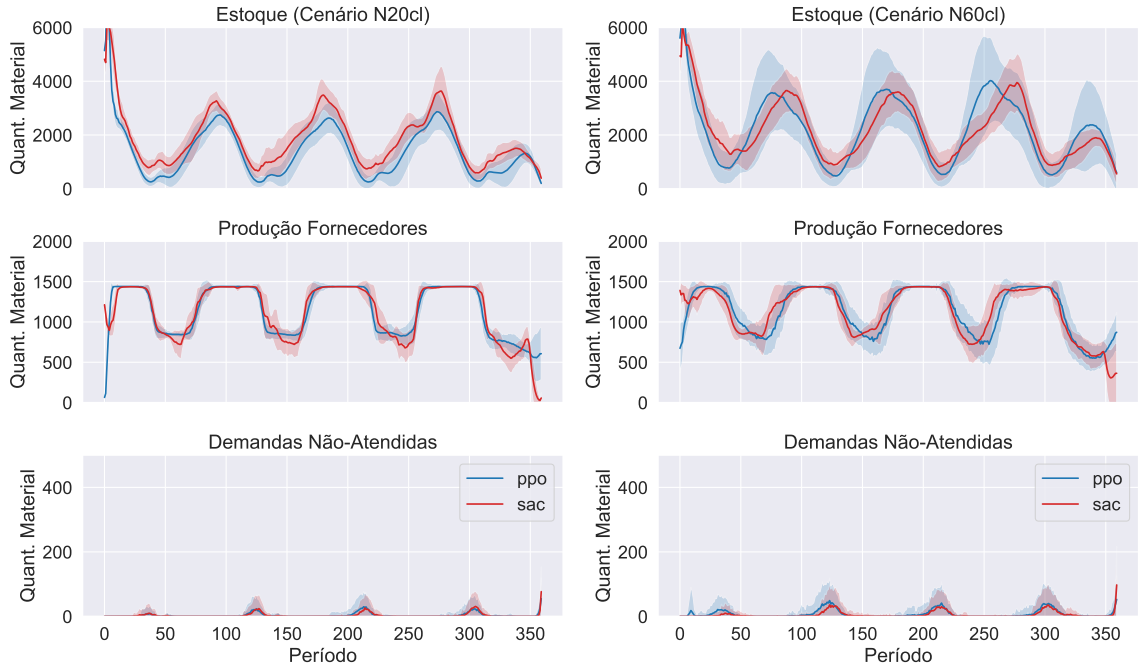

Figura 5.10. Quantidade média de material por tipo e por período considerando todos os episódios avaliados nos cenários com tempos de espera constantes. Áreas sombreadas representam ±1 desvio padrão, e os valores se referem à soma de todos os nós da cadeia. Os estoques e a produção nos fornecedores seguem o padrão senoidal das demandas.

5.3.2.6 Tipos de Custo

Analisamos também os resultados por tipos de custos para avaliar quais deles são responsáveis pela diferença entre os algoritmos. A Figura 5.11 mostra a composição dos custos finais para cada cenário de demanda sazonal, considerando cada tipo de custo. Os valores são a média de todos os episódios avaliados. É importante notar que o eixo vertical de cada gráfico tem faixas de valores diferentes. No cenário `N20`, PPO e SAC têm o mesmo nível de demandas não atendidas, mas o SAC opera com uma maior quantidade de material, como pode ser verificado pelos maiores custos de estoque e processamento. Isso faz com que o PPO seja melhor neste cenário. Já nos demais cenários, apesar do SAC manter um custo mais alto de processamento, ele atinge custos de estoque similares ao PPO e, principalmente, atende melhor às demandas. Assim, o SAC tem melhores resultados nestes cenários.

A Figura 5.12 mostra os custos por tipo para os cenários com demandas regulares. O algoritmo PPO tem melhor resultado no cenário `rN50cl` por atender melhor à demanda e ter menores custos de estoque, já que nos demais custos apresenta valores similares ao SAC. Já nos outros três cenários ambos os algoritmos alcançam resultados

totais similares, mas com o PPO operando mais material (maiores custos de fornecimento, processamento e transporte), mas por outro lado com menor custo de estoque e de penalização por excesso de estoque.

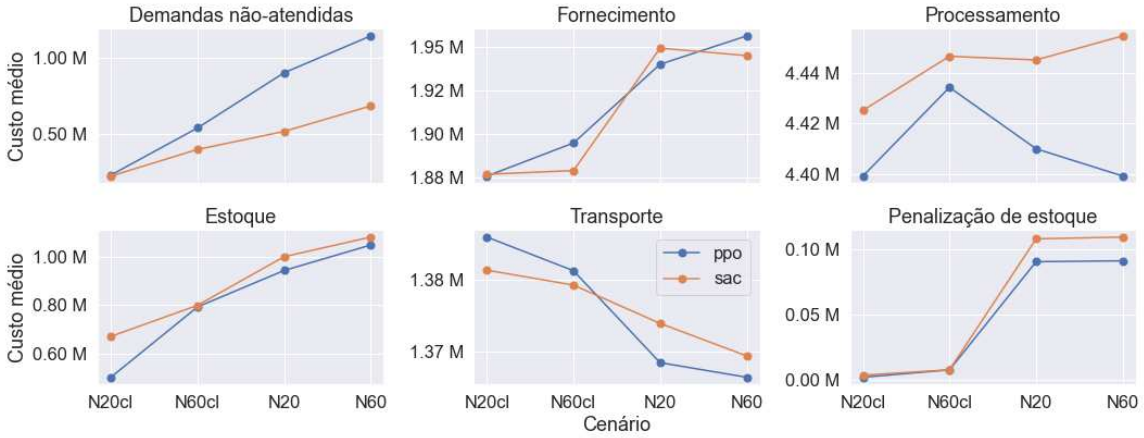

Figura 5.11. Custos por tipo para os cenários com demandas sazonais. Os cenários estão ordenados nos eixos horizontais pelo nível de incerteza, sendo os dois primeiros com tempos de espera constantes e os dois seguintes com tempos de espera estocásticos. Os eixos verticais se referem aos custos de operação por tipo.

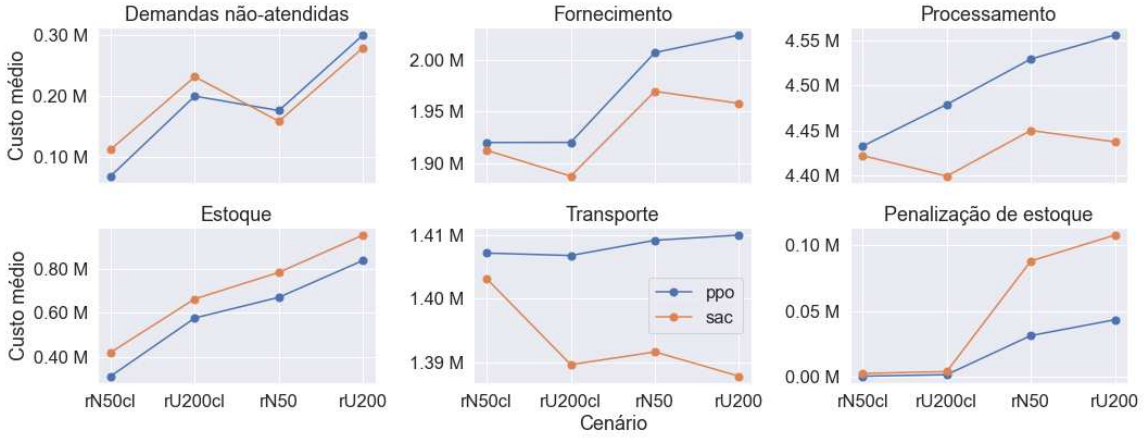

Figura 5.12. Custos por tipo para os cenários com demandas regulares. Os cenários estão ordenados nos eixos horizontais pelo nível de incerteza, sendo os dois primeiros com tempos de espera constantes e os dois seguintes com tempos de espera estocásticos. Os eixos verticais se referem aos custos de operação por tipo.

5.3.3 Experimentos Adicionais

5.3.3.1 PPO com 20 mil Episódios

Embora tenhamos feito uma comparação justa entre os algoritmos em relação ao critério de parada utilizado, em temos práticos o PPO precisou de apenas $^1/_{12}$ do tempo gasto pelo SAC (por rodada de treinamento, em média). Assim, decidimos executar novos experimentos com o PPO considerando treinamentos de tamanho fixo de 20 mil episódios, para comparar os resultados finais e o tempo de execução. A Figura 5.13 apresenta os resultados para os cenários com demandas regulares. Podemos notar que a melhora nesses cenários foi pequena. Mas já foi suficiente para que o PPO passasse a ser melhor que o SAC, considerando os testes-t de Welch, no cenário `rN50` (além de continuar melhor no cenário `rN50cl`). Os resultados para os cenários com demandas sazonais são mostrados na Figura 5.14. Nesses cenários a melhora dos resultados do PPO foi mais significativa, tendo uma redução nos custos totais nos quatro cenários apresentados. Além disso, considerando os testes-t de Welch, antes o SAC era melhor em três cenários, e agora permaneceu melhor apenas no cenário `N60` (mesmo assim com o PPO tendo um resultado bem mais próximo). O PPO manteve melhores resultados no cenário `N20cl` e passou a ter também no cenário `N60cl`. Apenas no cenário `N20` não foi possível apontar, com confiança estatística, qual algoritmo teve desempenho melhor. Em relação ao tempo de execução, cada treinamento do PPO levou em média 1,7h, o que ainda é apenas $^1/_6$ do tempo gasto pelo SAC.

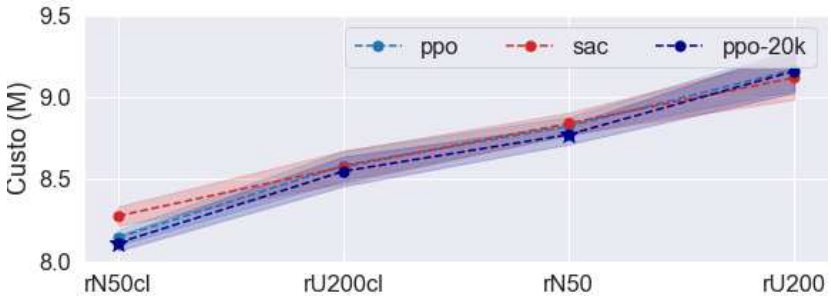

Figura 5.13. Médias dos custos totais de operação na avaliações dos melhores modelos dos experimentos adicionais, para cenários com demanda regular. Os valores estão em milhões e as áreas sombreadas representam o erro padrão. O marcador $\star$ indica se há uma diferença significativa entre os algoritmos, considerando testes-t de Welch.

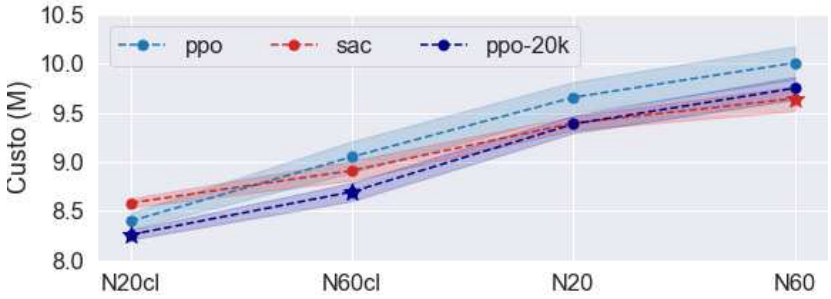

Figura 5.14. Médias dos custos totais de operação na avaliações dos melhores modelos dos experimentos adicionais, para cenários com demanda sazonal. Os valores estão em milhões e as áreas sombreadas representam o erro padrão. O marcador ⋆ indica se há uma diferença significativa entre os algoritmos, considerando testes-t de Welch.

5.3.3.2 Experimentos com A2C, DDPG e TD3

Os experimentos da primeira fase foram realizados considerando os valores padrões dos hiperparâmetros dos algoritmos, uma vez que eles foram ajustados para diversos problemas de espaços contínuos. Após a realização dos experimentos já apresentados, decidimos avaliar o impacto do ajuste de hiperparâmetros, fazendo experimentos adicionais com os algoritmos A2C, DDPG e TD3, seguindo a metodologia da segunda fase dos experimentos. Ou seja, fizemos os experimentos como se não os tivéssemos eliminado na fase anterior. Como apresentado na Tabela 5.6, os resultados dos três algoritmos melhoraram de forma substancial, especialmente em relação à estabilidade considerando os diferentes cenários. O algoritmo A2C apresentou resultados médios bem melhores do que na primeira fase, mas com grande variância nos cenários `N20cl`, `N60cl` e `N60`. O algoritmo DDPG apresentou resultados médios ruins nestes mesmos cenários e resultados consistentes nos demais cenários. Já o TD3 teve resultados consistentes em todos os cenários avaliados. Entretanto, os três algoritmos continuaram apresentado resultados piores que o PPO e o SAC, o que indica que estes seriam os melhores algoritmos escolhidos mesmo se tivéssemos considerado os cinco algoritmos na segunda fase dos experimentos.

5.4 Conclusões

Neste capítulo, apresentamos os experimentos com cinco algoritmos *Policy Gradient* em uma cadeia de suprimentos de quatro estágios com dois nós por estágio e incertezas nas demandas sazonais e nos tempos de espera. Na primeira fase, os algoritmos foram treinados considerando treinamentos de tamanho fixo em oito cenários representativos.

Tabela 5.6. Resultados da segunda fase dos experimentos incluindo os algoritmos eliminados na primeira fase. A tabela apresenta a média e o desvio padrão (em milhares) dos custos totais de operação para os cinco algoritmos utilizados. Os números nos nomes dos cenários indicam o nível de perturbação da demanda.

Cenário	**A2C**		**DDPG**		**PPO**		**SAC**		**TD3**	
	Média	σ	*Média*	σ	*Média*	σ	*Média*	σ	*Média*	σ
`rN50cl`	8.213	191	8.418	103	**8.144**	103	8.275	132	8.500	162
`rU200cl`	8.617	216	8.607	203	8.584	203	8.575	220	8.876	245
`rN50`	9.009	150	8.986	127	8.824	127	8.841	147	9.278	232
`rU200`	9.674	326	9.257	282	9.170	282	9.121	296	9.820	360
`N20cl`	9.447	4.872	14.700	189	**8.396**	189	8.581	104	8.914	113
`N60cl`	9.730	3.603	16.159	355	9.051	355	**8.911**	225	9.373	241
`N20`	10.997	243	10.296	355	9.657	355	**9.394**	159	10.302	244
`N60`	11.427	4.176	13.929	379	10.009	379	**9.645**	281	10.503	290

Nós usamos curvas de aprendizado, ferramental estatístico para comparação múltipla por intervalo de treinamento, e resultados finais com os melhores modelos, para selecionar os dois algoritmos com melhor desempenho (PPO e SAC). Nós não chegamos a aprofundar a análise de porque estes algoritmos tiveram o melhor desempenho, mas algumas hipóteses podem ser levantadas. Primeiro que, sabemos que o PPO possui a ideia de múltiplos atores do A2C, mas limita o tamanho do passo nas atualizações da política. Dessa forma, os resultados sugerem que essa limitação do passo é importante para o problema que abordamos, já que o PPO teve resultado bem superior ao A2C, mesmo quando consideramos o ajuste de hiperparâmetros. Já em relação ao DDPG e o TD3, sabemos que TD3 é uma evolução direta do DDPG, e os resultados dos nossos experimentos parecem indicar que as melhorias do algoritmo se traduzem em melhoria de resultados para o nosso problema. Ao analisarmos os resultados de cada rodada de treinamento, vemos que em algumas rodadas ambos os algoritmos alcançam resultados similares ao SAC e PPO. Mas o problema deles é que uma ou outra rodada de treinamento (mais frequentemente o DDPG) tem resultado bem abaixo, o que indica que eles têm um comportamento menos robusto para a solução do nosso problema. Por fim, os resultados também sugerem que a maximização de entropia introduzida pelo SAC melhora os resultados de métodos *off-policy* para o problema que abordamos.

Na segunda fase dos experimentos, os hiperparâmetros dos dois algoritmos, PPO e SAC, foram ajustados com um critério de parada que considera um número máximo de avaliações consecutivas sem melhora no modelo. Os algoritmos foram treinados nos mesmos cenários da primeira fase e tiveram resultados comparáveis em termos de curvas de aprendizado, resultados finais e complexidade amostral. Contudo, o PPO é mais

rápido em termos de tempo de execução gastando 1/8 do tempo do SAC para o ajuste de hiperparâmetros, e 1/12 por rodada de treinamento. Experimentos adicionais com o PPO, com treinamentos de tamanho fixo, mostraram que ele alcança um desempenho um pouco melhor que o SAC nos resultados finais, mesmo ainda gastando somente 1/6 do tempo de execução. Portanto, os resultados experimentais sugerem que o PPO pode ser uma boa escolha para ser usada na prática para o problema abordado.

Capítulo 6

Extensão: Cadeias Multiproduto

Neste capítulo, apresentamos uma extensão do problema abordado na presente tese, considerando cadeias de suprimentos multiproduto. Na Seção 6.1 são apresentadas as modificações na definição do problema. A formulação PDM e o modelo PL com parâmetros incertos adaptados para essa extensão são apresentados nas seções 6.2 e 6.3, respectivamente. Já as seções 6.4 e 6.5 apresentam a metodologia experimental e os resultados dos experimentos realizados. Por fim, a Seção 6.6 apresenta as conclusões do presente capítulo.

6.1 Definição do Problema Multiproduto

Após a realização dos experimentos de comparação dos algoritmos *Policy Gradient* apresentada no capítulo anterior, direcionamos a pesquisa para o tratamento de uma versão mais geral do problema original. Nesta versão, consideramos que existem N tipos matérias-primas e N tipos de produtos, e que cada tipo de matéria-prima pode ser processada para produzir um tipo de produto. Nesse contexto, os varejistas precisam atender às demandas de clientes separadas para cada tipo de produto. Em relação às capacidades, consideramos que as capacidades dos estoques locais e de produção nos fornecedores (primeiro nível da cadeia) são dadas para cada tipo de produto. Já as capacidades de processamento nas fábricas são compartilhadas para todos os tipos de materiais. Ou seja, se uma fábrica tem uma determinada capacidade c, tal capacidade limita o processamento da fábrica considerando a soma de todos os tipos de matéria-prima. Nesta extensão, o problema passou a ter também capacidades de transporte para cada par origem-destino da cadeia. A capacidade de transporte também é compartilhada, considerando que todos os tipos de materiais são transportados em conjunto. Vale notar que as capacidades compartilhadas de transporte e de pro-

cessamento é que tornam o problema mais complexo, já que, se elas também fossem dadas por tipo de material, poderíamos resolver um problema separado para cada tipo de material. A razão de processamento nas fábricas e todos os custos operacionais (exceto os de transporte) passam a ser tratados para cada tipo de material. Devido à inclusão das capacidades compartilhadas de processamento e de transporte, passou a ser necessário tratar, com penalizações, possíveis excessos em ambos os casos. Tais penalizações e também as que já existiam têm os mesmos valores independente do tipo de material. Por fim, os tempos de espera de produção de matéria-prima nos fornecedores são independentes para cada tipo de matéria-prima. Já os tempos de espera de transporte têm o mesmo valor para todos os tipos de material, já que considera-se que os materiais são enviados juntos em um mesmo meio de transporte.

A partir da definição apresentada para a extensão do problema, podemos notar que ela tem maior complexidade que o problema original. Principalmente pelo fato de que será necessário incluir informações por produto na definição dos estados e decisões por produto na definição das ações. Dessa forma, o número de dimensões nos espaços de estados e de ações cresce com a quantidade de tipos de materiais tratados. Além disso, a possibilidade de excessos das capacidades compartilhadas de transporte e processamento também dificultam o problema, pois são mais situações com as quais os agentes de AR precisarão aprender a lidar.

Por fim, podemos observar que o problema poderia ter sido definido de forma ainda mais geral, considerando capacidades compartilhadas e por tipo de material ao mesmo tempo em todas as capacidades (produção nos fornecedores, estoque, processamento e transporte). De toda forma, adotamos a definição apresentada por entender que ela tem aplicabilidade prática e é suficiente para fornecer respostas de interesse sobre a aplicação de métodos *Policy Gradient* na resolução de problemas de cadeias de suprimentos multiproduto.

6.2 Formulação PDM

A formulação PDM do problema estendido segue a mesma ideia geral da formulação do problema original, mas agora considerando multiprodutos. As alterações nos estados, ações, dinâmica do ambiente e recompensas são apresentados nas subseções a seguir.

6.2.1 Espaço de Estados

Considerando uma cadeia de suprimentos com quatro estágios, dois nós por estágio e N tipos de produto, um estado, para um dado período t, é um vetor contínuo de

$26 \times N + 1$ dimensões contendo os seguintes valores.

- O nível atual de estoque *de cada tipo* de matéria-prima ou produto em cada nó.
- Para cada fornecedor:
  - a quantidade *de cada tipo* de matéria-prima que está sendo produzida e que estará disponível no próximo período $(t + 1)$;
  - a soma das quantidades *de cada tipo* de matéria-prima que estão sendo produzidas e que estarão disponíveis nos períodos após o próximo.
- Para cada outro nó:
  - a quantidade *de cada tipo* de material em transporte e que chegará no nó em questão no próximo período (é a soma do material *de um determinado tipo* enviado pelos dois nós do estágio anterior);
  - a soma das quantidades *de cada tipo* de material em transporte que chegarão nos períodos após o próximo.
- As demandas dos clientes finais *de cada tipo* de produto em cada varejista para o próximo período $(t + 1)$.
- O número de períodos restantes até o fim do episódio

Podemos notar, portanto, que o número de dimensões do espaço de estados cresce de maneira linear com a quantidade de tipos produtos. O número de dimensões seria, por exemplo, 53 no caso de dois tipos de produto e 79 no caso de três tipos de produto.

A Seção 6.4 apresenta como a normalização dos estados foi tratada nos experimentos realizados.

6.2.2 Espaço de Ações

Considerando uma cadeia de suprimentos com quatro estágios, dois nós por estágio e N tipos de produto, uma ação é um vetor contínuo de $14 \times N$ dimensões com os seguintes valores.

- A quantidade *de cada tipo* de material a ser produzida em cada fornecedor.
- Para cada nó (exceto os varejistas):
  - a quantidade *de cada tipo* de material a ser enviada para cada um dos dois nós do próximo estágio.

Assim como no caso do espaço de estados, o número de dimensões do espaço de ações cresce linearmente com o número de tipos de produtos. Em um exemplo com dois tipos de produto, o espaço de ações teria 28 dimensões, e com três tipos, 42 dimensões.

A Seção 6.4 apresenta como a normalização das ações foi tratada nos experimentos realizados.

6.2.3 Dinâmica do Ambiente

A simulação da cadeia de suprimentos foi alterada para se adequar às definições do problema multiproduto. A principal alteração, claro, se refere a tratar o fluxo de múltiplos tipos de matéria-prima e de produto ao longo da cadeia de suprimentos, considerando estoques e produção nos fornecedores por tipo de material, e transporte e processamento conjunto de vários tipos de material. As demandas e os tempos de espera estocásticos passaram a ser gerados para cada tipo de produto (exceto no caso do tempo de espera de transporte, que é único para todos os tipos de material, conforme já apresentado).

Em relação ao transporte de materiais, é importante notar que, ao considerar as decisões do agente referentes ao envio de material a partir de um determinado nó de origem, a soma das quantidades de cada tipo de material pode ser maior do que a capacidade compartilhada de transporte. Caso isso ocorra, a simulação considera que os materiais são colocados no meio de transporte na ordem do conjunto de tipos de produtos até que sua capacidade seja atingida. Dessa forma, as decisões do primeiro material são seguidas, depois do segundo, e assim sucessivamente, até que se complete a capacidade de transporte. As quantidades de materiais que excederem o limite de transporte são mantidas no estoque do nó de origem. O mesmo tipo de tratamento é feito para tratar a capacidade compartilhada de processamento das fábricas. Caso a decisão do agente leve a um processamento maior do que a capacidade de processamento, o material excedente é mantido no estoque da fábrica.

6.2.4 Recompensas

Em relação às recompensas, a ideia geral permanece a mesma de considerar o negativo da soma dos custos de operação e de penalização de cada período. Os custos de produção nos fornecedores, estoque e processamento são contabilizados por tipo de material. Já os custos de transporte e as penalizações possuem valores independentes do tipo de material. Como mencionado na subseção sobre a dinâmica do ambiente, pode ser que o agente tome decisões que levariam a quantidades de material maiores

do que as capacidades de transporte e processamento. Em ambos os casos, o material em excesso é mantido em estoque, mas uma penalização é contabilizada de forma a que o agente possa aprender a evitar tais decisões.

6.3 Modelo PL com Parâmetros Incertos

O modelo de PL com parâmetros incertos apresentado na Seção 2.4 foi generalizado para resolver a problema estendido considerando multiprodutos. As principais alterações se referem às mudanças na definição dos parâmetros relacionados aos tipos de material e no tratamento dos mesmos nas restrições e na função objetivo. Os parâmetros de capacidade de estoque, produção nos fornecedores, custos de operação (exceto de transporte) e materiais iniciais (de estoque, fornecimento e transporte) são agora tratados por tipo de material. Já a capacidade de processamento é tratada de forma compartilhada para todos os produtos. Por fim, o modelo passou a incluir uma restrição de capacidade de transporte, compartilhada para todos os tipos de material, para cada par origem-destino.

Os conjuntos do modelo são apresentados na Tabela 6.1, onde q é o número de nós da cadeia, h o horizonte de planejamento (ou tamanho do episódio), l^{max} é o máximo valor possível de tempo de espera e ρ é o número de tipos de material. Como a relação entre matéria-prima e produto é de um para um, considera-se que a matéria-prima de índice r é processada para gerar um produto também de índice r. O período 0 se refere ao estado inicial da cadeia. As variáveis do modelo são apresentadas na Tabela 6.2 e seus parâmetros na Tabela 6.3. De forma similar ao modelo anterior, os parâmetros p_{ijnr}, f_n e t_{ijnm} são binários, e todos os demais são inteiros. f_n é igual a 1 para todas as fábricas, e vale 0 para os demais nós. p_{ijnr} é usado para definir quais nós são fornecedores, de qual tipo de matéria-prima, e também para mapear os tempos de espera. Há um valor estocástico de tempo de espera para cada fornecedor, para cada tipo de matéria-prima, em cada período, de forma que $p_{ijnr} = 1$ somente se n é um fornecedor, e a realização do tempo de espera para produzir a matéria-prima r em cada fornecedor no período $i - j$ é j; caso contrário p_{ijnr} é zero. De forma similar, t_{ijnm} define quais pares de nós podem podem ter transporte e também mapeia os tempos de espera.

O objetivo do modelo PL é minimizar os custos totais de operação e a função

Tabela 6.1. Conjuntos do modelo PL

Conjunto	**Descrição**
$N = \{1, ..., q\}$	conjunto de nós da cadeia de suprimentos
$I = \{0, 1, ..., h + l^{max}\}$	conjunto de períodos (ou *time steps*)
$J = \{1, ..., l^{max}\}$	conjunto de tempos de espera possíveis
$R = \{1, ..., \rho\}$	conjunto de tipos de matéria-prima/produto

Tabela 6.2. Variáveis do modelo PL

Var.	**Descrição**
S_{inr}	o nível de estoque do nó n no período i do material r
T_{ijnmr}	a quantidade do material r enviada do nó n no período $i - j$ para chegar no nó m no período i
P_{ijnr}	quantidade da matéria-prima r produzida no fornecedor n no período $i - j$ que estará disponível no período i
F_{inr}	quantidade da matéria-prima r processada pela fábrica n no período i
D^e_{inr}	excesso do material r descartado por exceder a capacidade de estoque do nó n no período i
D^d_{inr}	unidades do produto r faltantes para atender à demanda de cliente pelo varejista n no período i

objetivo[1] é dada pela Equação 6.1.

$$\begin{aligned} \min \quad & \sum_{i\in I}\sum_{n\in N}\sum_{r\in R}\left(c^s_{nr}S_{inr} + c^f_{nr}F_{inr} + c^e D^e_{inr} + c^d D^d_{inr}\right) + \\ & \sum_{i\in I}\sum_{j\in J}\sum_{n\in N}\sum_{r\in R}\left(c^p_{nr}P_{ijnr} + \sum_{m\in N} c^t T_{ijnmr}\right) \end{aligned} \tag{6.1}$$

As restrições são definidas como se segue. As restrições 6.2 controlam o estoque, transporte de material, e as demandas. As capacidades são tratadas pelas restrições 6.3, 6.4, 6.5 e 6.6. As restrições 6.7 são usadas para calcular a quantidade de matéria-prima processada nas fábricas. E, finalmente, as restrições 6.8, 6.9 e 6.10 são usadas

[1]Em relação aos resultados apresentados nas seções de experimentos, os custos relacionados aos materiais iniciais (estoque, fornecimento e transporte) não são considerados na função objetivo.

Tabela 6.3. Parâmetros do modelo PL

Par.	Descrição
q	número de nós da cadeia
h	horizonte de planejamento (tamanho do episódio)
c^s_{nr}	custo de estocar uma unidade do material r no nó n
c^p_{nr}	custo de produzir uma unidade da matéria-prima r no fornecedor n
c^f_{nr}	custo de processar uma unidade da matéria-prima r na fábrica n
c^t	custo de enviar uma unidade de material de um nó para outro
c^e	custo por unidade de material que excede capacidade de estoque, transporte ou processamento
c^d	custo por não atendimento à demanda (por unidade de material)
b^s_{nr}	capacidade de estoque do material r no nó n
b^p_{nr}	capacidade de produção da matéria-prima r no fornecedor n
b^f_n	capacidade de processamento no nó n
b^t_{nm}	quantidade máxima de material que pode ser enviada a partir do nó n para o nó m
r_n	razão de processamento no nó n
l^{max}	tempo de espera máximo (para produção nos fornecedores e para transporte)
l^{avg}	tempo de espera médio (para produção nos fornecedores e para transporte)
f_n	indica se é possível processar matéria-prima no nó n
p_{ijnr}	indica se é possível produzir a matéria-prima r no nó n no período $i-j$ para estar disponível no período i
t_{ijnm}	indica se é possível enviar material do nó n no período $i-j$ para chegar no nó m no período i
s_{nr}	nível inicial de estoque do material r no nó n
p_{inr}	quantidade inicial da matéria-prima r produzida pelo fornecedor n que estará disponível no período i (definido para $i \leq l^{avg}$)
t_{inmr}	quantidade inicial da matéria-prima r enviada pelo nó n para o nó m que estará disponível no período i (definido para $i \leq l^{avg}$)
d_{inr}	demanda estocástica de cliente do produto r a ser atendida pelo nó n no período i

para se levar em conta os materiais iniciais de fornecimento, transporte e estoque.

$$\begin{aligned} S_{inr} = S_{(i-1)nr} + \sum_{j \in J} P_{ijnr} + \sum_{j \in J} \sum_{m \in N} T_{ijmnr} - D^{e}_{inr} \\ -r_n \Big(\sum_{j \in J} \sum_{m \in N} T_{(i+j)jnmr} \Big) - d_{inr} + D^{d}_{inr} \\ \forall \quad i \in \{1, ..., h\}, n \in N, r \in R \end{aligned} \tag{6.2}$$

$$0 \leq P_{ijnr} \leq p_{ijnr} b^{p}_{nr} \quad \forall \quad i \in I, j \in J, n \in N, r \in R \tag{6.3}$$

$$0 \leq \sum_{r \in R} F_{inr} \leq b^{f}_{n} \quad \forall \quad i \in I, n \in N \tag{6.4}$$

$$0 \leq \sum_{r \in R} T_{ijnmr} \leq t_{ijnm} b^{t}_{nm} \quad \forall \quad i \in I, j \in J, n \in N, m \in N \tag{6.5}$$

$$\begin{aligned} 0 \leq S_{(i-1)nr} + \sum_{j \in J} P_{ijnr} + \sum_{j \in J} \sum_{m \in N} \sum_{r \in R} T_{ijmnr} - D^{e}_{inr} \leq b^{s}_{nr} \\ \forall \quad i \in \{1, ..., h\}, n \in N, r \in R \end{aligned} \tag{6.6}$$

$$F_{inr} = f_n r_n \Big(\sum_{j \in J} \sum_{m \in N} T_{(i+j)jnmr} \Big) \quad \forall \quad i \in \{1, ..., h\}, n \in N, r \in R \tag{6.7}$$

$$P_{iinr} = p_{inr} \quad \forall \quad i \in \{1, ..., l^{avg}\}, n \in N, r \in R \tag{6.8}$$

$$T_{iinmr} = t_{inmr} \quad \forall \quad i \in \{1, ..., l^{avg}\}, n \in N, m \in N, r \in R \tag{6.9}$$

$$S_{0nr} = s_{nr} \quad \forall \quad n \in N, r \in R \tag{6.10}$$

6.4 Metodologia Experimental

Nesta seção são apresentados os cenários experimentais (Seção 6.4.1), a escolha do algoritmo utilizado e sua forma de aplicação (Seção 6.4.2), o agente PL usado como baseline (Seção 6.4.3) e a metodologia experimental utilizada (Seção 6.4.4).

6.4.1 Cenários Experimentais

Para a definição dos cenários experimentais, nossa ideia inicial consistiu em partir de alguns dos cenários experimentados anteriormente, mas estendendo-os para considerar dois e três tipos de produto. Optamos por usar como referência cenários com demanda sazonal, por serem mais complexos, e com maior perturbação de demanda, por serem cenários com maior incerteza. Criaríamos então cenários considerando tanto tempos de espera constantes quanto estocásticos. Desta forma, escolhemos como referência os cenários `N60` e `N60c1` (apresentados na Tabela 4.2). Nesta seção vamos apresentar o que os novos cenários multiproduto têm de diferente em relação a estes cenários de referência e, portanto, os parâmetros não apresentados aqui para os cenários têm valores iguais aos dos cenários de referência.

Para uma melhor avaliação da aplicação de um algoritmo *Policy Gradient* em um problema multiproduto, optamos por criar quatro conjuntos de cenários, com variações na configuração das demandas e nos custos de cada tipo de produto. A Tabela 6.4 apresenta as características de cada conjunto de cenários (os cenários em si de cada conjunto são apresentados mais adiante). A primeira coluna da tabela apresenta o nome do conjunto. Para cada conjunto são apresentadas as configurações do primeiro, segundo e terceiro tipos de produtos (r_1, r_2 e r_3, respectivamente), conforme apontado na segunda coluna da tabela. Vale notar que nos cenários com dois tipos de produtos, são usadas apenas as configurações de r_1 e r_2. A terceira coluna da tabela indica o número de picos da demanda sazonal, ou *reg.* se a demanda é regular (não-sazonal). A quarta coluna apresenta os valores mínimo e máximo possíveis de demanda, e a quinta coluna os valores mínimos e máximos da demanda sazonal antes de considerar a perturbação (ou o valor médio no caso de demanda regular). Por fim, a sexta coluna indica o valor dos custos operacionais (produção nos fornecedores, estoque, processamento e transporte) podendo ser os mesmos dos cenários de referência (*originais*), ou em uma escala crescente em relação a tais cenários. Na tabela podemos observar que o conjunto A se refere a cenários mais próximos dos cenários de referência, alterando-se apenas a quantidade de tipos de produto. Já no conjunto B, os custos operacionais para cada tipo de produto são crescentes. Os cenários do conjunto C têm os mesmos custos operacionais dos cenários de referência, mas a configuração das demandas é diferente para cada tipo de produto. O primeiro tipo de produto tem demanda como nos cenários anteriores, já o segundo tem demanda regular e com faixa de valores ligeiramente mais baixa, e o terceiro tipo de produto tem demanda sazonal como o primeiro, mas com dois picos sazonais ao invés de quatro. Por fim, os cenários do conjunto D têm tanto configurações de demanda quanto custos crescentes de acordo com cada tipo

de produto.

Tabela 6.4. Configuração de demandas e de custos para os produtos em cada conjunto de cenários. A notação segue a Seção 4.2.1 e os custos originais podem ser vistos na Tabela 4.1.

Conj.	**Tipo de Produto**	Picos (z)	**Demandas** (min,max)	(min^{sin}, max^{sin})	**Custos**
A	r_1 r_2 r_3	4	(0,400)	(100,300)	originais
B	r_1 r_2 r_3	4	(0,400)	(100,300)	originais 2*originais 3*originais
C	r_1 r_2 r_3	4 *reg.* 2	(0,400) (0,300) (0,400)	(100,300) 150 (100,300)	originais
D	r_1 r_2 r_3	4 *reg.* 2	(0,400) (0,300) (0,400)	(100,300) 150 (100,300)	originais 2*originais 3*originais

Na Tabela 6.5 são apresentados os cenários multiproduto experimentados. A primeira coluna apresenta o conjunto ao qual o cenário pertence, a segunda o nome do cenário e a terceira a quantidade de tipos de produto. Na quarta e quinta colunas podemos notar que todos os cenários têm o mesmo tipo de perturbação, dado por uma distribuição Normal com média zero e desvio padrão 60. Por fim, a última coluna da tabela indica se o cenário possui tempos de espera estocásticos ou não.

6.4.2 Escolha do Algoritmo e Aplicação

Devido ao tempo necessário para a realização dos experimentos, decidimos escolher apenas um dos algoritmos para os experimentos com os cenários multiproduto. No Capítulo 5, dentre os cinco algoritmos *Policy Gradient* utilizados no problema original, os melhores resultados foram obtidos com os algoritmos PPO e SAC. Entre os dois, como o PPO teve tempo de execução bem menor, optamos por utilizá-lo nos experimentos reportados neste capítulo.

Na Seção 4.2.2 apresentamos como o PPO foi utilizado no problema original, e, nesta seção, apresentaremos as adaptações feitas na forma de aplicação para o problema multiproduto. A ideia geral permanece a mesma de usar valores de estado e ação

Tabela 6.5. Configuração dos novos cenários multiproduto.

Conj.	Cenário	Tipos de Produto	Função de Pert.	Pert. p	Tempos de espera Estocásticos
A	m2p_N60cl	2	$\mathcal{N}$	60	✓
	m2p_N60	2			
	m3p_N60cl	3			✓
	m3p_N60	3			
B	m2p_N60cl_i	2	$\mathcal{N}$	60	✓
	m2p_N60_i	2			
	m3p_N60cl_i	3			✓
	m3p_N60_i	3			
C	m2p_N60cl_dbp	2	$\mathcal{N}$	60	✓
	m2p_N60_dbp	2			
	m3p_N60cl_dbp	3			✓
	m3p_N60_dbp	3			
D	m2p_N60cl_dbp_i	2	$\mathcal{N}$	60	✓
	m2p_N60_dbp_i	2			
	m3p_N60cl_dbp_i	3			✓
	m3p_N60_dbp_i	3			

normalizados ou reescalonados em relação ao intervalo $[-1, 1]$. O que apresentamos aqui são as mudanças em relação aos limites máximos usados na normalização.

Como feito anteriormente, os valores de estado são divididos por um limite máximo e então reescalonados do intervalo $[0, 1]$ para o intervalo $[-1, 1]$. Seguindo a notação usada para o modelo PL (Seção 6.3), seja b^s_{nr} a capacidade de estoque do material r no nó n, b^p_{nr} a capacidade de produção da matéria-prima r no fornecedor n, b^t_{mn} a capacidade de transporte do nó n para o nó m, d^{max}_r o maior valor possível de demanda para o produto r, l^{max} o maior valor possível de tempo de espera e h o horizonte de planejamento. Os limites máximos usados na normalização são:

- b^s_{nr} para o nível de estoque atual de cada material em cada nó.
- Para cada fornecedor:
  - b^p_{nr} para a matéria-prima r sendo produzida e que estará disponível no próximo período.
  - $b^p_{nr} * (l^{max} - 1)$ para a soma da matéria-prima r que está sendo produzida e que estará disponível depois do próximo período.
- Para cada outro nó:

  - $b^t_{ln} + b^t_{mn}$ para o material em transporte que chegará no nó n no próximo período e que foi enviado pelos nós antecessores l e m.
  - $(b^t_{ln} + b^t_{mn}) * (l^{max} - 1)$ para o material em transporte que chegar[a no nó n depois do próximo período e que foi enviado pelos nós antecessores l e m.

- d^{max}_r para demandas de cliente referentes ao produto r em cada varejista.

- h para o número de períodos restantes até o final do episódio.

Em relação aos valores das ações, e como feito nos experimentos anteriores, a saída do algoritmo PPO é um vetor cujos valores estão no intervalo $[-1, 1]$. Estes valores são reescalonados para o intervalo $[0, 1]$ e então transformados nos valores realmente usados na simulação da cadeia de suprimentos. A transformação nos valores é basicamente a mesma apresentada na Seção 4.2.2, mas agora considerando a capacidade de produção do fornecedor por tipo de matéria-prima (b^p_{nr}) nas decisões sobre quanto produzir em cada fornecedor, e o estoque atual por tipo de material (b^s_{nr}) no caso das decisões sobre quanto transportar.

É importante ressaltar que, conforme já apresentado na Seção 6.2, para os cenários multiproduto, a simulação da cadeia foi alterada para tratar as capacidades compartilhadas de transporte e processamento. A ideia geral é seguir a decisão do agente, na ordem dos tipos de produto, enquanto a capacidade de transporte (ou processamento) não for atingida. Uma vez atingida a capacidade, o material que não pode ser transportado (ou processado) é mantido em estoque.

6.4.3 O *Baseline*: Agente PL

O agente PL utilizado como *baseline* nos cenários multiproduto é construído da mesma forma como é apresentado na Seção 4.2.3, mas agora adaptado ao modelo PL multiproduto (Seção 6.3). Portanto, o agente PL é construído a partir das variáveis de decisão P_{ijnr} e T_{ijnmr} do modelo (produção nos fornecedores e transporte de material, respectivamente). As outras variáveis de decisão (estoque, material processado, excesso de material e produto faltante) não precisam ser codificadas no agente, já que elas são tratadas pela simulação da cadeia. Os valores das variáveis de decisão utilizadas são normalizados e reescalonados para o intervalo $[-1, 1]$, utilizando os mesmos limites máximos adotados na aplicação do PPO (apresentados na seção anterior).

6.4.4 Metodologia Experimental

A metodologia experimental utilizada para os cenários multiproduto segue a mesma apresentada na Seção 4.2.5, com pequenas mudanças. O ajuste de hiperparâmetros é feito da mesma forma, agora executado com o cenário `m2p_N60` e os valores encontrados são utilizados nos demais cenários. A terceira fase, de avaliação final dos resultados, também é feita da mesma forma.

Em relação à segunda fase da metodologia, referente aos treinamentos, nos experimentos do Capítulo 4 cada treinamento consiste em executar o algoritmo por 7,2 milhões de períodos (ou 20 mil episódios). Já na segunda fase dos experimentos apresentados no Capítulo 5, uma rodada de treinamento é interrompida depois de: (a) $N = 40$ avaliações consecutivas sem melhora (ou seja, dois mil episódios ou 720 mil períodos), ou (b) após 20 mil episódios de treinamento. Por fim, nos experimentos adicionais com o PPO relatados também no Capítulo 5, adotamos novamente treinamentos fixos de 20 mil episódios. A partir desta experiência, decidimos adotar a seguinte estratégia para os experimentos realizados com os cenários multiproduto. Cada rodada de treinamento tem um número mínimo de 10 mil episódios, e pode ser interrompida após: (a) $N = 100$ avaliações consecutivas sem melhora (ou seja, cinco mil episódios ou 1,8 milhões de períodos), ou (b) após o máximo de 30 mil episódios (ou 10,8 milhões de períodos). Essa configuração foi adotada porque nos experimentos da segunda fase do Capítulo 5 algumas rodadas de treinamento do PPO foram interrompidas precocemente e os resultados do algoritmo melhoraram depois com treinamentos de tamanho fixo. Por outro lado, aumentamos o tamanho máximo do treinamento para 30 mil episódios, pelo fato do problema multiproduto ser mais complexo. Uma outra possível abordagem seria realizar experimentos exploratórios para se encontrar o melhor tamanho de treinamento, mas, para evitar esse passos, consideramos que cinco mil episódios sem melhora é uma quantidade razoável para considerar que o algoritmo poderá não melhorar até o limite de 30 mil. Na seção de resultados apresentamos uma avaliação sobre essa estratégia adotada.

6.5 Experimentos e Resultados

Os experimentos foram realizados no mesmo ambiente dos experimentos anteriores, ou seja, usando Python 3.6.10 em um computador com processador 2.9 GHz x 6, 32 GB de RAM, GPU de 6 GB, e Ubuntu Linux 20.04. A simulação da cadeia de suprimentos foi implementada em Python seguindo o padrão OpenAI Gym [Brockman et al., 2016] e o modelo PL foi resolvido usando o *solver* CPLEX 12.10 via interface

Python. Foi utilizada a versão do PPO da biblioteca Stable Baselines 3 (SB3) [Raffin et al., 2019], e a biblioteca RL Baselines3 Zoo [Raffin, 2020] foi usada no ajuste de hiperparâmetros. Assim como nos experimentos anteriores, consideramos RNAs da classe MLP e utilizamos o algoritmo *Adam* como método de gradiente.

O restante desta seção é organizado como se seque. Na Seção 6.5.1 apresentamos a fase de ajuste dos hiperparâmetros. A Seção 6.5.2 apresenta os resultados gerais para todos os cenários experimentados. Uma análise sobre os resultados para cada tipo de produto é apresentada na Seção 6.5.3. Finalmente, na Seção 6.5.4, as curvas de aprendizado do PPO e os tamanhos dos treinamentos são discutidos.

6.5.1 Ajuste dos Hiperparâmetros

O primeiro passo dos experimentos foi o ajuste dos hiperparâmetros do PPO que seguiram a metodologia proposta, utilizando-se o cenário `m2p_N60`. Os intervalos pré-definidos para os hiperparâmetros do algoritmos foram os mesmos dos experimentos anteriores. Os melhores valores encontrados para os hiperparâmetros do PPO são apresentados na Tabela 6.6. Para cada hiperparâmetro é apresentado: o tipo de amostragem (categórica, uniforme, ou log-uniforme), os possíveis (intervalos de) valores pré-definidos, o melhor valor encontrado no final do processo de ajuste, e descrição do hiperparâmetro. Em relação ao tempo de execução, o processo de ajuste demorou cerca de 50 horas (considerando a execução de dois processos paralelos).

6.5.2 Resultados Gerais

A Tabela 6.7 sumariza os resultados dos experimentos comparando os agentes PPO e PL para todos os cenários. A primeira coluna mostra o conjunto de cenários, a segunda indica a quantidade de tipos de produtos, a terceira coluna mostra se os tempos de espera são estocásticos, e a quarta coluna apresenta os nomes dos cenários. As quatro colunas seguintes apresentam a média e o desvio padrão dos custos totais de operação para os agentes PL e PPO, respectivamente. As últimas duas colunas mostram o ganho do PPO sobre o agente PL (diferença e porcentagem, respectivamente).

Analisando os dados da tabela podemos notar claramente uma grande diferença entre os cenários com tempos de espera estocásticos e constantes. Independentemente do conjunto, ou seja, independentemente da configuração adotada para as demandas e para os custos operacionais, o agente PPO tem resultados próximos ao PL nos cenários com tempos de espera constantes. Em tais cenários o PPO têm resultados piores da ordem de 0,0% a -2,7%. Já nos cenários com tempos de espera estocásticos, os

Tabela 6.6. Melhores parâmetros do PPO encontrados no ajuste de hiperparâmetros. A coluna **S** indica as opções de amostrando, as quais são: C: categórica, U: amostrada a partir do intervalo com domínio linear, L: amostrada a partir do intervalo com domínio logarítmico, e -: fixo.

Parâmetro	**S.**	**Valores possíveis**	**Melhor valor**	**Descrição**
`n_steps`	C	$[2^5; 2^6; ...; 2^{11}]$	2^9	Tamanho da trajetória ao coletar dados (τ no Algoritmo 5)
`n_epochs`	C	[3;5;10;20]	3	K Épocas de atualização do gradiente (K no Algoritmo 5)
`batch_size`	C	[64;128;256;512]	128	Tamanho do minilote para cada atualização do gradiente
`vf_coef`	U	[0;1]	$0,46574$	Coeficiente da função valor (c_1 na Equação 3.37)
`clip_range`	C	[0,1;0,2;0,3]	$0,2$	Coeficiente da função objetivo cortada (ϵ na Equação 3.36)
`gae_lambda`	C	[0,9;0,92;0,95;0,98;1]	$0,95$	Fator de desconto λ para calcular GAE
`gamma`	C	[0,95;0,98;0,99;0,995;0,999;0,9999]	$0,99$	Valor de γ usado para calcular GAE
`net_arch`	C	[(64,64);(128,128);(256,256)]	$(64, 64)$	Unidades das camadas ocultas das RNAs
`learning_rate`	L	[0,00001;0,01]	$0,00022$	Taxa de aprendizado do otimizador Adam
`activation_fn`	C	[ReLU,TanH]	ReLU	Função de ativação das RNAs
`max_grad_norm`	C	[0,3;0,5;0,6;0,7;0,8;0,9;1;2;5]	1	Para cortar gradientes normalizados
`ortho_init`	C	[True,False]	True	Se é para usar inicialização ortogonal
`n_actors`	-	4	4	Número de atores (N no Algoritmo 5)
`ent_coef`	-	0	0	Coeficiente de entropia na função objetivo (c_2 na Equação 3.37)

Tabela 6.7. Resultados para todos os cenários considerados: cada conjunto de cenários indica a configuração dos custos e das demandas por tipo de produto. A tabela apresenta, para cada cenário, a média e o desvio padrão (em milhares) dos custos totais de operação para os agentes PL (o *baseline*) e PPO. As últimas duas colunas apresentam o ganho do PPO sobre o PL (diferença e porcentagem, respectivamente).

Conj.	**# Prod.**	**T.Esp. Estoc.**	**Cenário**	**Agente PL**		**Agente PPO**		**Ganho**	
				Média	σ	*Média*	σ	valor	%
	2		`m2p_N60cl`	17.395	218	17.674	422	-279	-1,6 %
A	2	✓	`m2p_N60`	26.209	626	20.632	524	5.577	27,0 %
	3		`m3p_N60cl`	26.437	296	26.767	504	-330	-1,2 %
	3	✓	`m3p_N60`	40.911	991	31.693	784	9.218	29,1 %
	2		`m2p_N60cl_i`	25.477	207	25.842	506	-365	-1,4 %
B	2	✓	`m2p_N60_i`	35.369	708	29.410	572	5.959	20,3 %
	3		`m3p_N60cl_i`	49.629	308	50.457	966	-827	-1,6 %
	3	✓	`m3p_N60_i`	63.378	868	56.041	721	7.337	13,1 %
	2		`m2p_N60cl_dbp`	15.064	218	15.068	317	-5	-0,0 %
C	2	✓	`m2p_N60_dbp`	23.212	650	16.957	331	6.254	36,9 %
	3		`m3p_N60cl_dbp`	23.993	282	24.653	468	-660	-2,7 %
	3	✓	`m3p_N60_dbp`	36.721	826	28.440	435	8.281	29,1 %
	2		`m2p_N60cl_dbp_i`	20.863	222	21.000	403	-137	-0,7 %
D	2	✓	`m2p_N60_dbp_i`	27.123	593	23.365	436	3.758	16,1 %
	3		`m3p_N60cl_dbp_i`	45.173	380	45.389	708	-216	-0,5 %
	3	✓	`m3p_N60_dbp_i`	54.685	756	50.065	559	4.620	9,2 %

resultados do agente PPO são significativamente melhores que os do baseline, tendo diferenças entre 9,2% e 36,9%. Tais resultados são compatíveis com os observados anteriormente (Seção 4.3.2), que mostram que o algoritmo PPO é mais indicado para cenários com tempos de espera estocásticos.

A Figura 6.1 apresenta os resultados para os cenários com tempos de espera constantes dos conjuntos A e B, e a Figura 6.2 apresenta os mesmos tipos de dados para os cenários dos conjuntos C e D. Podemos notar que, como esperado, os custos totais crescem de acordo com o número de produtos e os valores totais são maiores nos cenários com custos crescentes. O mesmo comportamento é observado nos cenários com demandas diferentes por tipo de produto (conjuntos C e D). Já as figuras 6.3 e 6.4 apresentam os resultados dos cenários com tempos de espera estocásticos dos conjuntos A e B e C e D, respectivamente. O mesmo comportamento é verificado nesses cenários, com custos totais maiores para cenários com mais produtos e com custos crescentes. Pode-se perceber que a diferença entre o PPO e o agente PL é maior nos cenários com custos originais.

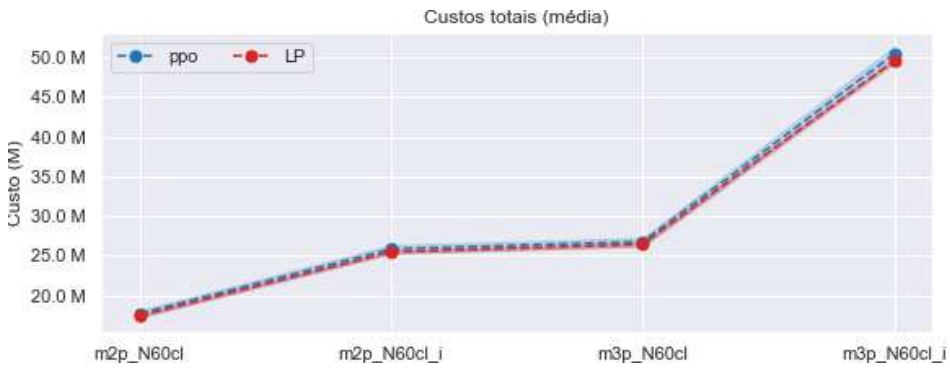

Figura 6.1. Custos totais (médios) para os cenários com tempos de espera constantes dos conjuntos A e B. O PPO e agente PL alcançam resultados semelhantes. Os custos crescem com o número de produtos e, claro, quando os custos de cada tipo de produto são crescentes.

6.5.3 Resultados por Tipo de Produto

Nesta seção avaliamos os resultados considerando os valores encontrados para cada tipo de produto. Vamos analisar primeiro os cenários com tempos de espera constantes. Nos cenários do conjunto A, com configuração de custos original e mesma configuração de demanda, as quantidades para cada tipo de operação (fornecimento, processamento, transporte e estoque) são muito próximas para os três tipos de produto. E os valores também são parecidos para ambos os agentes (PPO e PL). Nos cenários do conjunto B,

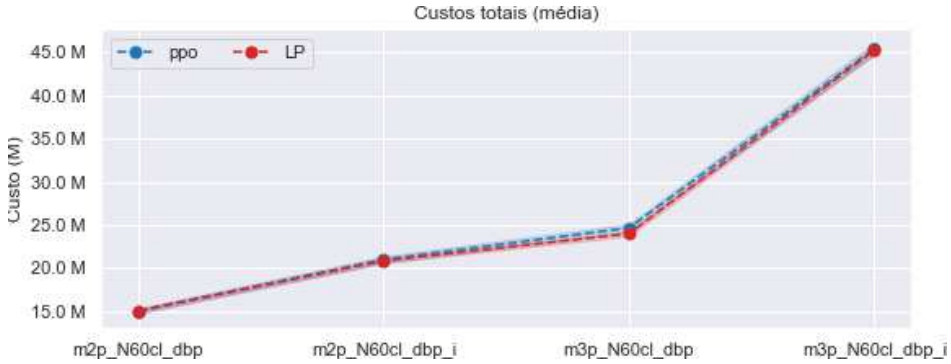

Figura 6.2. Custos totais (médios) para os cenários com tempos de espera constantes dos conjuntos C e D. O PPO e agente PL alcançam resultados semelhantes. Os custos crescem com o número de produtos e, claro, quando os custos de cada tipo de produto são crescentes.

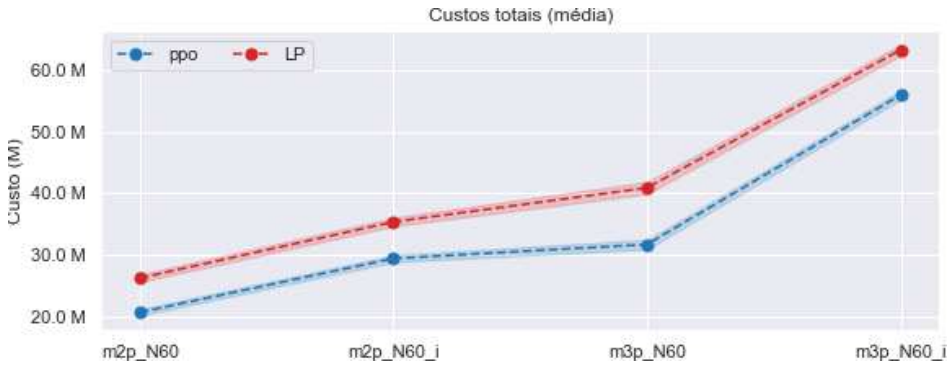

Figura 6.3. Custos totais (médios) para os cenários com tempos de espera estocásticos dos conjuntos A e B. O PPO alcança melhores resultados que o agente PL. Mais uma vez, como esperado, os custos crescem com o número de produtos.

com custos crescentes e mesma configuração de demanda, os resultados do agente PL e PPO são bem próximos para aos dois primeiros produtos. Já para o terceiro produto, PPO produz menos material e perde mais demandas dos clientes. Nos cenários do conjunto C, com configuração de custos original e demandas por tipo de produto, ambos os agentes produzem menos do segundo tipo de produto, o que é esperado já que a demanda dele é menor. E, no geral, o agente PL faz um pouco mais de estoque que o PPO. Por fim, nos cenários do conjunto D, com custos crescentes e demandas por tipo de produto, os resultados são similares aos do grupo B, mas a diferença para o terceiro produto agora é maior. Vale notar que, em todos os cenários com três tipos produtos e custos crescentes, ou seja, também incluindo os cenários com tempos de espera estocásticos, o agente PPO, em comparação ao baseline, aprendeu a produzir uma menor quantidade do terceiro tipo de produto que tem maior custo, levando a

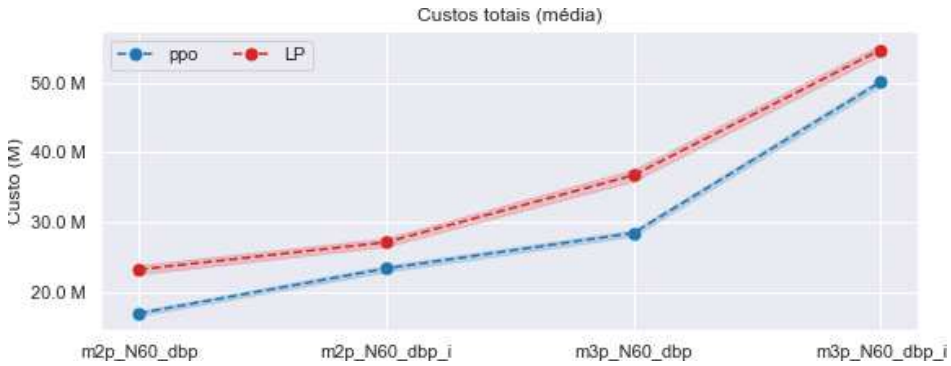

Figura 6.4. Custos totais (médios) para os cenários com tempos de espera estocásticos dos conjuntos C e D. O PPO alcança melhores resultados que o agente PL. Mais uma vez, como esperado, os custos crescem com o número de produtos.

uma maior penalização por não atendimento à demanda.

Em relação aos cenários com tempos de espera estocásticos, os dados mostram que o agente PL teve penalizações muito altas por exceder a capacidade de processamento em todos os cenários (Figura 6.5). Ele teve também penalizações maiores que o PPO por exceder a capacidade de transporte e por não atendimento à demanda (com exceção do terceiro tipo de produto). Vale ressaltar que em uma operação real, os custos de penalização por não atendimento à demanda e por excesso da capacidade de estoques são tipos de custos que podem, de certa forma, ser efetivamente incorridos. O primeiro, se considerarmos que a penalização por demanda não atendida representa o custo de compra de um produto de uma empresa concorrente para atender a demanda sem perder o cliente. Já o segundo, porque ocorreria um problema real de ter que lidar com um material que não pode ser armazenado em estoque, o que ocasionaria um custo operacional. Mas as penalizações por excesso da capacidade de transporte e de processamento não são custos que ocorreriam de fato. Como o material em excesso é simplesmente mantido em estoque, tais penalizações fazem sentido para guiar o aprendizado do agente, mas talvez não seja interessante considerá-las na comparação entre os agentes.

Nesse sentido, a Tabela 6.8 apresenta a comparação entre os agentes PPO e PL desconsiderando os custos com as penalizações por exceder as capacidades de transporte e processamento. A primeira coluna mostra o conjunto de cenários, a segunda indica a quantidade de tipos de produtos, a terceira coluna mostra se os tempos de espera são estocásticos, e a quarta coluna apresenta os nomes dos cenários. As duas colunas seguintes apresentam a média dos custos totais de operação para os agentes PL e PPO, respectivamente. As últimas duas colunas mostram o ganho do PPO sobre o agente PL

(diferença e porcentagem, respectivamente). Analisando os dados da tabela podemos notar que continua existindo uma grande diferença entre os cenários com tempos de espera estocásticos ou constantes. Mas agora a diferença do PPO para o agente PL nos cenários com tempos de espera estocásticos é menor do que a vista na Tabela 6.7. Em tais cenários, os resultados do agente PPO são melhores que o agente PL com diferenças entre 1,4% e 10,9% (enquanto antes, com todas as penalizações, a faixa era de 9,2% a 36,9%). Na tabela, podemos observar que as diferenças do PPO para o agente PL são mais significativas nos cenários com custos iguais para todos os tipos de produtos. As maiores quedas nas diferenças são observadas nos cenários com custos crescentes. Isso parece ter acontecido pelo fato do agente PPO ter aprendido a produzir uma menor quantidade do produto mais caro. Tal situação pode ter ocorrido pelo fato da penalização por demanda não atendida ter o mesmo valor, independente do tipo de produto. Já em relação aos cenários com tempos de espera constantes, a mudança foi pequena, o PPO teve resultados piores que o *baseline* da ordem de 0,1% a -2,5% (e antes era de 0,0% a -2,7%).

6.5.4 Curvas de Aprendizado e Tamanhos dos Treinamentos

As Figuras 6.6 e 6.7 apresentam as curvas de aprendizado dos treinamentos do PPO para os cenários `m3p_N60_dbp_i` e `m3p_N60cl_dbp_i`, respectivamente. Os eixos verticais se referem ao total de recompensas acumuladas por episódio, e os eixos horizontais ao número de períodos durante o treinamento. Os gráficos da esquerda apresentam as curvas de aprendizado reais das cinco rodadas de treinamento. Já os gráficos da direita apresentam os valores médios relacionados às avaliações do modelo feitas durante o treinamento. As curvas dos demais cenários não são apresentadas por questão de espaço, mas possuem comportamento razoavelmente similar. As curvas das avaliações mostram certa estabilidade já antes do final dos treinamentos. Já as curvas do próprio treinamento continuam com um leve crescimento até o final dos treinamentos, especialmente nos cenários com tempos de espera estocásticos.

Os tamanhos médios dos treinamentos são mostrados na Figura 6.8. Considerando todos os cenários, os treinamentos têm tamanho médio de 27 mil episódios, sendo a menor média de 23 mil episódios e a maior de 30 mil (ou seja, neste caso todas as rodadas de treinamento chegaram a 30 mil episódios). Já em relação ao tempo de execução do algoritmo, em média, cada treinamento gastou cerca de 3 horas.

Tabela 6.8. Resultados para todos os cenários considerados, desconsiderando as penalizações por exceder as capacidades de transporte e processamento. Cada conjunto de cenários indica a configuração dos custos e das demandas por tipo de produto. A tabela apresenta, para cada cenário, a média (em milhares) dos custos totais de operação para os agentes PL (o *baseline*) e PPO. As últimas duas colunas apresentam o ganho do PPO sobre o PL (diferença e porcentagem, respectivamente).

Conj.	**T.Esp. Estoc.**	**Cenário**	**Agente PL** *Média*	**PPO** *Média*	**Ganho** valor	%
A		`m2p_N60cl`	17.395	17.656	-261	-1,5 %
	✓	`m2p_N60`	21.397	19.947	1.451	7,3 %
		`m3p_N60cl`	26.437	26.740	-303	-1,1 %
	✓	`m3p_N60`	34.006	30.668	3.338	10,9 %
B		`m2p_N60cl_i`	25.477	25.814	-337	-1,3 %
	✓	`m2p_N60_i`	29.790	28.661	1.129	3,9 %
		`m3p_N60cl_i`	49.629	50.424	-795	-1,6 %
	✓	`m3p_N60_i`	56.789	55.316	1.473	2,7 %
C		`m2p_N60cl_dbp`	15.064	15.056	8	0,1 %
	✓	`m2p_N60_dbp`	17.654	16.643	1.011	6,1 %
		`m3p_N60cl_dbp`	23.993	24.607	-614	-2,5 %
	✓	`m3p_N60_dbp`	30.139	27.860	2.279	8,2 %
D		`m2p_N60cl_dbp_i`	20.863	20.980	-117	-0,6 %
	✓	`m2p_N60_dbp_i`	23.811	22.979	832	3,6 %
		`m3p_N60cl_dbp_i`	45.173	45.332	-159	-0,4 %
	✓	`m3p_N60_dbp_i`	50.259	49.580	679	1,4 %

6.6 Conclusões

Neste capítulo apresentamos uma extensão do problema original de forma a tratar agora diferentes tipos de produto. A definição do problema é apresentada e as principais diferenças são: a definição de custos e de demandas por tipo de produto, a definição de capacidades dos estoques locais e de produção nos fornecedores (primeiro nível da cadeia) também por tipo de produto e, por fim, a definição das capacidades de processamento nas fábricas e de transporte compartilhadas para todos os tipos de materiais. Esta versão do problema é mais complexa tanto por ter espaços de estados e de ações com mais dimensões (por envolverem informações e decisões para cada tipo de produto), como por ser necessário lidar com penalizações por excesso das capacidades compartilhadas de processamento e transporte. A formulação PDM e o modelo PL utilizados anteriormente foram estendidos para formalizar esta nova versão do problema tratado.

Para avaliar o desempenho de algoritmos *Policy Gradient* no problema multiproduto, definimos cenários com dois e com três produtos, com configurações de custos e de demandas iguais entre eles e também com custos crescentes e com configurações de demandas diferentes entre os tipos de produto. Realizamos experimentos com o algoritmo PPO, por ter sido o algoritmo com melhores resultados nos experimentos do Capítulo 5, já que teve desempenho semelhante ao SAC, mas com melhores tempos de execução. Os resultados do algoritmo foram comparados com um *baseline* baseado no modelo de PL, considerando demandas esperadas e tempos de espera médios. O *baseline* segue a mesma ideia adotada no Capítulo 4, mas agora adaptado para multiprodutos. Os resultados encontrados mostram que o algoritmo PPO pode ser interessante para cenários com tempos de espera estocásticos, como já acontecia no problema original. Os ganhos sobre o baseline são da ordem de 9,2% a 36,9% considerando todos os custos e penalizações, ou da ordem de 1,4% a 10,9% se desconsiderarmos as penalizações por excesso de capacidades de processamento e transporte. Já para cenários com tempo de espera constante os resultados do PPO são ligeiramente piores que o *baseline*, da ordem de 0,0% a -2,7% considerando todos os custos e penalizações, ou da ordem de 0,1% a -2,5% se desconsiderarmos as penalizações por excesso de capacidades de processamento e transporte.

A análise dos resultados mostra que, nos cenários com tempo de espera estocástico, o *baseline* utilizado tem soluções ruins, não lidando bem com as capacidades compartilhadas de processamento e transporte. Nesse sentido, em trabalhos futuros pode ser interessante a avaliação de algum outro tipo de *baseline*, como, por exemplo, o uso de estratégias de *rolling horizon* ou de algoritmos de Programação Estocástica. Já nos cenários com custos crescentes, o algoritmo PPO aprendeu a produzir uma menor quantidade do tipo de produto mais caro, levando a uma maior penalização por demanda não atendida. Isso seria esperado pelo fato da penalização de demanda não atendida ser a mesma para todos os produtos, enquanto o produto mais caro tem custos operacionais três vezes maiores que o produto mais barato. Portanto, uma outra sugestão para trabalhos futuros seria verificar se o PPO alcançaria melhores resultados com valores de penalização por demanda não atendida que fossem escalados com os custos dos produtos.

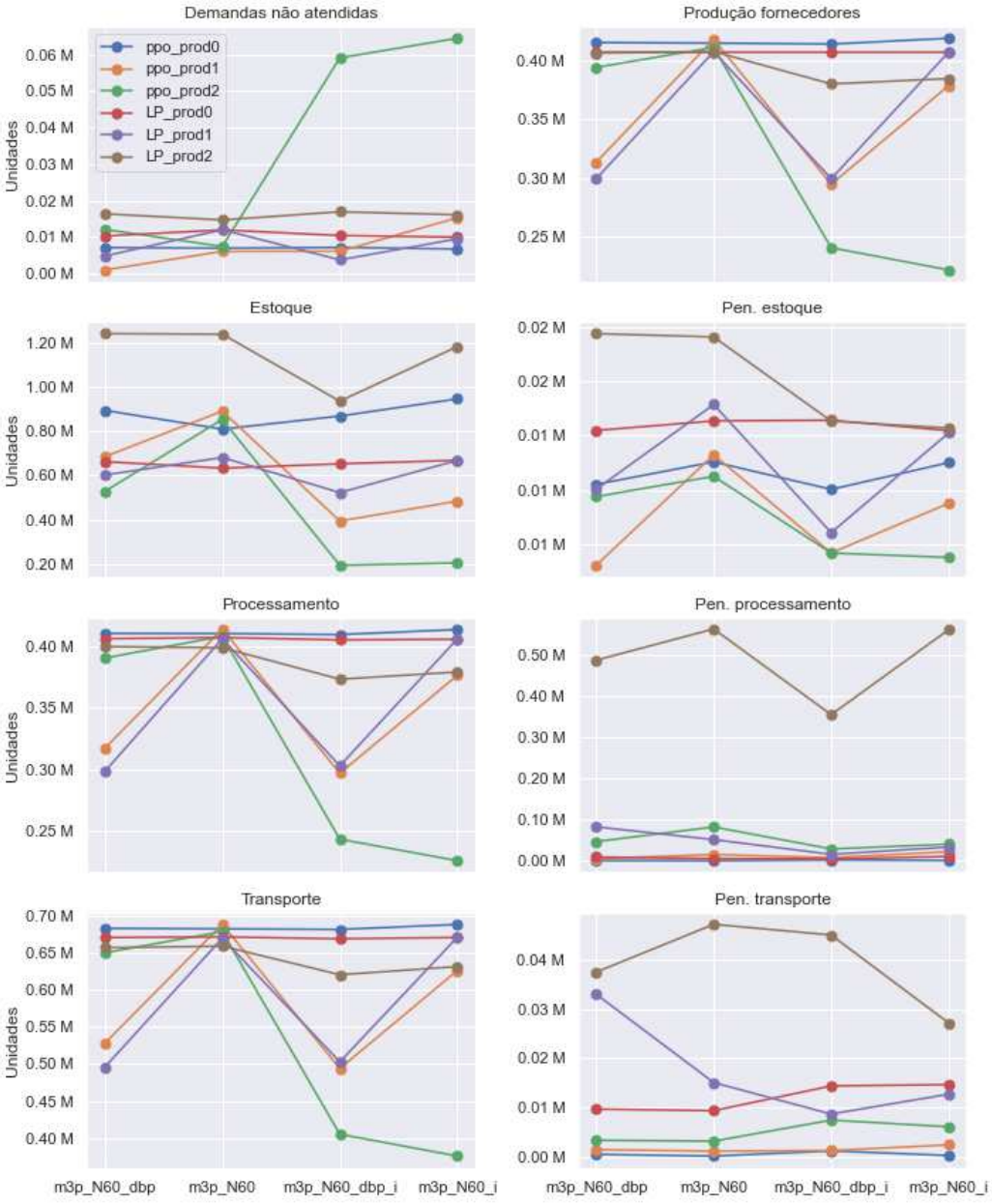

Figura 6.5. Quantidade de material por tipo de operação para os cenários com três produtos e tempos de espera estocásticos. Cada gráfico apresenta as quantidades de material referentes a um tipo de operação (ou de penalização). Os eixos verticais apresentam as quantidades de material (os valores são diferentes para cada gráfico), e os eixos horizontais representam os cenários. Os cenários foram organizados de acordo com o crescimento dos custos totais (cenários com `_i` no nome têm custos crescentes, e com `_dbp` têm configuração de demanda diferente para cada tipo de produto).

Figura 6.6. Curvas de aprendizado para o cenário `m3p_N60_dbp_i`. O gráfico da esquerda mostra as curvas de aprendizado das cinco rodadas de treinamento. O gráfico da direita mostra o valor médio das avaliações do modelo feitas durante as mesmas rodadas de treinamento.

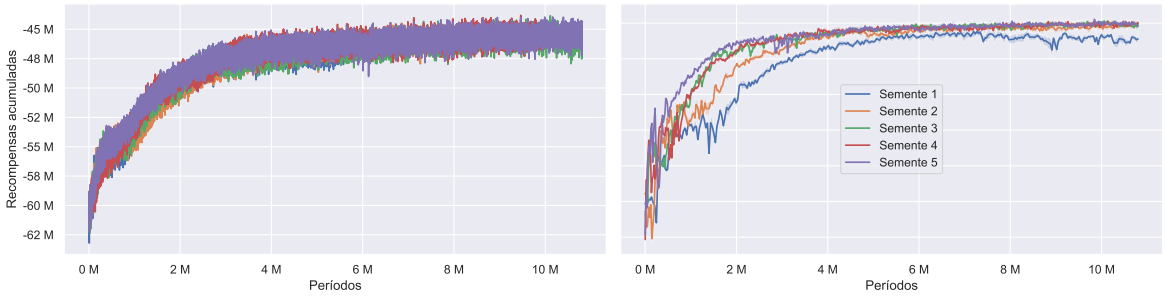

Figura 6.7. Curvas de aprendizado para o cenário `m3p_N60cl_dbp_i`. O gráfico da esquerda mostra as curvas de aprendizado das cinco rodadas de treinamento. O gráfico da direita mostra o valor médio das avaliações do modelo feitas durante as mesmas rodadas de treinamento.

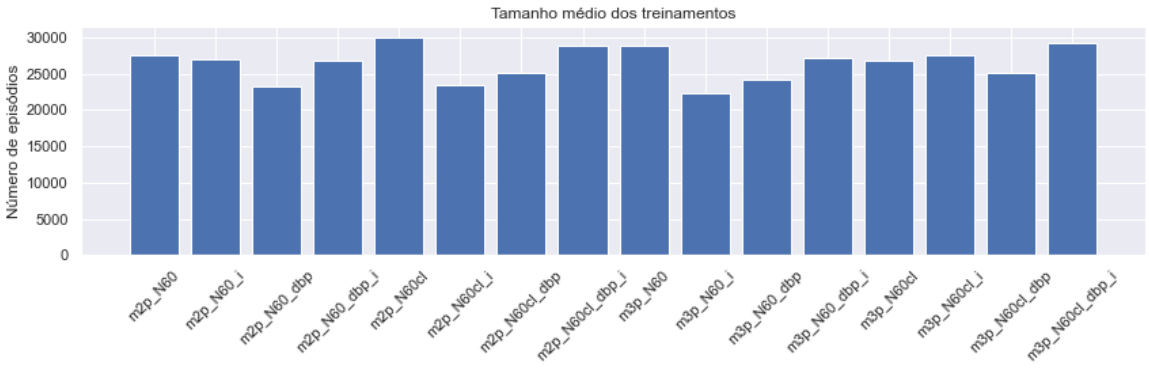

Figura 6.8. Tamanhos médios dos treinamentos dos cenários multiproduto experimentados.

Capítulo 7

Conclusões

Nesta tese trabalhamos com o problema de planejamento de produção e distribuição de produtos em uma cadeia de suprimentos com quatro estágios e dois nós por estágio. Nós consideramos incertezas nas demandas sazonais e nos tempos de espera, em uma abordagem na qual um tomador de decisões centralizado deve atender às demandas incertas dos clientes com o menor custo possível de operação. Em uma primeira versão do problema há um único tipo de matéria-prima e produto, e, posteriormente, consideramos múltiplos produtos. Nós apresentamos uma formulação PDM para o problema original e para a extensão e, da mesma forma, apresentamos modelos PL com parâmetros incertos para as duas versões do problema. A formulação PDM foi adaptada de forma a se obter bons resultados com os algoritmos *Policy Gradient* experimentados. Após um estudo de caso inicial, comparamos o algoritmo PPO com um *baseline* baseado na solução de um modelo PL (considerando demandas esperadas e tempos de espera médios), em 17 cenários com demandas regulares e sazonais, tempos de espera constantes e estocásticos e diferentes níveis de incerteza para as demandas. Os resultados indicam que o PPO traz bons resultados, em comparação com o baseline, em cenários com maior incerteza, especialmente na presença de tempos de espera estocásticos.

Na fase seguinte, comparamos cinco algoritmos *Policy Gradient* (A2C, DDPG, PPO, SAC e TD3) em oito dos 17 cenários usados anteriormente. Dividimos os experimentos em duas etapas. Na primeira, comparamos os algoritmos utilizando um tamanho fixo de treinamento, e selecionamos os dois melhores (PPO e SAC) com base nas curvas de aprendizado, comparações múltiplas usando testes-t de Welch por intervalo de treinamento e avaliação dos melhores modelos encontrados. Na segunda etapa, ajustamos os hiperparâmetros dos algoritmos considerando um critério de parada referente ao número máximo de avaliações do modelo sem melhora. Os dois algoritmos

obtiveram resultados semelhantes, mas o PPO tem um tempo de execução bem menor. Novos experimentos com o PPO, utilizando treinamentos de maior tamanho, mas ainda assim com menor tempo de execução que o SAC, mostraram que o PPO pode ser uma melhor escolha prática para o problema abordado. Vale ressaltar que os resultados encontrados para os algoritmos estão ligados à forma como as recompensas foram definidas, considerando o negativo dos custos operacionais a cada período. Fizemos experimentos exploratórios com o inverso do custo usando o PPO e os resultados foram bem piores, mas poderiam ainda ser pensadas outras formas de recompensa que poderiam trazer resultados diferentes para cada algoritmo.

Por fim, trabalhamos na versão estendida do problema, considerando dois e três tipos de produto. A modelagem do problema passou a considerar penalizações por excessos de capacidades compartilhadas de transporte e processamento, e foram criados 16 novos cenários com diferentes configurações de custos e de demandas por tipo de produto. O algoritmo PPO foi utilizado e comparado ao *baseline* baseado em modelo PL, obtendo melhores resultados nos cenários com tempos de espera estocásticos.

A partir das perguntas de pesquisa apresentadas na introdução desta tese (Capítulo 1), fazemos as seguintes considerações:

1. Os resultados indicam que é possível resolver o problema com métodos *Policy Gradient*, considerando a cadeia experimentada em uma abordagem de um único agente, com demandas estocásticas regulares e tempos de espera constantes.

2. Algoritmos *Policy Gradient*, especialmente PPO e SAC, conseguem resolver o problema em cenários com demandas incertas sazonais e tempos de espera estocásticos, com bons resultados em comparação ao *baseline*.

3. A adaptação da formulação PDM é um passo importante na obtenção de bons resultados com algoritmos *Policy Gradient*, especialmente a normalização dos estados, ações e recompensas.

4. As soluções encontradas pelos algoritmo PPO e SAC são melhores que o *baseline* adotado nos cenários com tempos de espera estocásticos, e são comparáveis nos cenários com tempos de espera constantes.

5. Os algoritmos PPO e SAC têm melhor desempenho que os algoritmos A2C, DDPG e TD3 nos cenários experimentados, sendo que o PPO tem melhor tempo de execução que o SAC.

6. A metodologia experimental adotada, consistindo em ajuste de hiperparâmetros, treinamentos com múltiplas sementes e avaliação dos melhores modelos (com

todas as suas nuances apresentadas) são importantes para a obtenção de bons resultados com os algoritmos utilizados.

7. O algoritmo PPO consegue resolver a versão estendida do problema, com multiprodutos, com resultados melhores que o *baseline* nos cenários com tempos de espera estocásticos.

Atingidos os objetivos propostos, aproveitamos a experiência adquirida na realização do trabalho para sugerir diferentes opções de continuidade para trabalhos futuros[1]. Elas são apresentadas a seguir.

1. Utilização de melhores *baselines* para avaliar o desempenho dos algoritmos de AR.
    - Como o tempo para solução do modelo PL é muito pequeno, uma possibilidade seria a implementação de uma heurística que, a cada passo da simulação da cadeia, resolvesse diferentes modelos PL considerando um conjunto de possíveis cenários de realização para as demandas e os tempos de espera, e utilizasse a média das soluções encontradas como o próximo passo a ser dado na simulação.
    - Outro caminho seria a utilização de métodos de Programação Estocástica, que são ferramentas de Otimização mais apropriadas para a solução de problema com incerteza.
2. Contribuição na área de AR através da adaptação de algum algoritmo *Policy Gradient*, especificamente para resolver o problema abordado.
3. Alterações na modelagem do problema:
    - Inclusão da demanda esperada (já que ela está disponível, sendo utilizada pelo *baseline*) e/ou das últimas realizações de demanda na definição dos estados, a fim de avaliar o impacto disso na qualidade das soluções encontradas.
    - Inclusão das decisões referentes a quanto estocar no espaço de ações do problema. Se por um lado isso aumenta o tamanho do problema, por outro dá maior flexibilidade para as ações tomadas pelo agente.

[1] A implementação do ambiente (simulação da cadeia de suprimentos) será disponibilizada na plataforma *GitHub*, no endereço: `https://github.com/caburu/gym-supplychain`

- Na modelagem atual o material que chegará depois do próximo período é somado em um único valor. Seria interessante experimentar diferentes níveis de agregação destes valores, pois, se por um lado o número de dimensões do espaço de estados aumenta, por outro, a informação disponível para o agente se torna mais precisa.
- Na modelagem atual, a capacidade de produção dos fornecedores é considerada somente no período no qual a produção das matérias-primas é disparada, independente do tempo necessário para o material se tornar disponível. Seria interessante considerar que a capacidade do fornecedor fosse afetada também nos períodos seguintes até que o material se torne disponível.
- Em trabalhos similares é comum que a demanda seja observada somente no período em que ocorre, ao invés de ser revelada para o próximo período como nós adotamos neste trabalho. Essa opção poderia então ser avaliada.
- Nos cenários multiproduto, dividir a decisão de transporte em duas etapas, considerando primeiro quanto cada produto ocuparia do transporte, para depois decidir quanto viria de cada origem. O objetivo seria verificar se essa abordagem diminui as penalizações por excesso de transporte.

4. Solução do problema considerando a abordagem baseada em pedidos, conforme apresentada no início do Capítulo 2.

5. Experimentos com versões mais gerais do problema (por exemplo: cadeias maiores, mais tipos de produtos) para verificar se os algoritmos experimentados continuam adequados para a solução do problema. Para cenários muito complexos, a utilização de GNNs (*Graph Neural Networks*) pode ser um bom caminho a ser explorado.

6. Solução da versão discreta do problema, na qual as quantidades de material são dadas por valores inteiros.

7. Experimentar a utilização dos algoritmos em configurações de cadeia diferentes das apresentadas.

   - Uma ideia seria considerar situações com realizado diferente do previsto. Ou seja, treinar o algoritmo considerando um determinado cenário, mas avaliá-lo em condições inesperadas, como alguma mudança na distribuição das demandas a partir de um certo período do horizonte de planejamento.
   - Outra possibilidade seria considerar custos variáveis ao longo do horizonte de planejamento, ou ainda, incluir incertezas nos custos envolvidos.

8. Aprendizado de solução a partir de dados históricos, sejam eles dados reais de operação, ou gerados a partir da solução ótima de modelos PL para cenários amostrados. Métodos de AR *Offline*, *Imitation Learning* e/ou métodos heurísticos são potenciais candidatos para resolver o problema com essa abordagem.

9. Em relação aos experimentos:

    - No ajuste de hiperparâmetros, avaliar se a realização de um número maior de tentativas com tamanhos menores de treinamento, poderia levar a melhorias nos resultados encontrados. A ideia é verificar se, com um tempo de ajuste similar, é possível alcançar valores melhores para os parâmetros.
    - Experimentar o algoritmo PPO com um maior número de atores em paralelo, já que nos experimentamos realizados usamos sempre quatro atores.

Por fim, esperamos que este trabalho possa contribuir com a comunidade acadêmica, especialmente com aqueles interessados na aplicação de métodos de AR Profundo em problemas de tomada de decisão sequencial sob incerteza.

Referências Bibliográficas

Achiam, J. (2018). Spinning Up in Deep Reinforcement Learning. `https://spinningup.openai.com`.

Akiba, T.; Sano, S.; Yanase, T.; Ohta, T. & Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2623--2631.

Alves, J. C. & Mateus, G. R. (2020). Deep reinforcement learning and optimization approach for multi-echelon supply chain with uncertain demands. In Lalla-Ruiz, E.; Mes, M. & Voß, S., editores, *Proceedings of the 11th International Conference on Computational Logistics*, volume 12433 of *Lecture Notes in Computer Science*, pp. 584--599, Cham. Springer International Publishing.

Alves, J. C. & Mateus, G. R. (2021). Multi-echelon supply chains with uncertain seasonal demands and lead times using deep reinforcement learning. *Submitted to Annals of Operations Research*.

Alves, J. C.; Silva, D. M. d. & Mateus, G. R. (2021). Applying and comparing policy gradient methods to multi-echelon supply chains with uncertain demands and lead times. In Rutkowski, L.; Scherer, R.; Korytkowski, M.; Pedrycz, W.; Tadeusiewicz, R. & Zurada, J. M., editores, *Proceedings of the 20th International Conference on Artificial Intelligence and Soft Computing*, volume 12855 of *Lecture Notes in Computer Science*, pp. 229--239, Cham. Springer International Publishing.

Bergstra, J.; Bardenet, R.; Bengio, Y. & Kégl, B. (2011). Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24:2546--2554.

Bertsimas, D. & Tsitsiklis, J. N. (1997). *Introduction to linear optimization*, volume 6. Athena Scientific Belmont, MA. ISBN 978-1886529199.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J. & Zaremba, W. (2016). Openai gym. *arXiv preprint:1606.01540*.

Chaharsooghi, S. K.; Heydari, J. & Zegordi, S. H. (2008). A reinforcement learning model for supply chain ordering management: An application to the beer game. *Decision Support Systems*, 45(4):949 – 959. ISSN 0167-9236.

Colas, C.; Sigaud, O. & Oudeyer, P.-Y. (2019). A Hitchhiker's Guide to Statistical Comparisons of Reinforcement Learning Algorithms. In *ICLR Worskhop on Reproducibility*, Nouvelle-Orléans, United States.

Efron, B. & Tibshirani, R. J. (1994). *An introduction to the bootstrap*. CRC press. ISBN 978-0412042317.

François-Lavet, V.; Henderson, P.; Islam, R.; Bellemare, M. G. & Pineau, J. (2018). An introduction to deep reinforcement learning. *Found. Trends Mach. Learn.*, 11(3–4):219–354. ISSN 1935-8237.

Fujimoto, S.; van Hoof, H. & Meger, D. (2018). Addressing function approximation error in actor-critic methods. In Dy, J. & Krause, A., editores, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1587--1596. PMLR.

Geevers, K. (2020). Deep reinforcement learning in inventory management. Master in business engineering, `http://essay.utwente.nl/85432/`, University of Twente.

Giannoccaro, I. & Pontrandolfo, P. (2002). Inventory management in supply chains: a reinforcement learning approach. *International Journal of Production Economics*, 78(2):153 – 161. ISSN 0925-5273.

Gijsbrechts, J.; Boute, R. N.; Van Mieghem, J. A. & Zhang, D. (2019). Can deep reinforcement learning improve inventory management? performance on dual sourcing, lost sales and multi-echelon problems. *Performance on Dual Sourcing, Lost Sales and Multi-Echelon Problems (July 29, 2019)*.

Goodfellow, I.; Bengio, Y. & Courville, A. (2016). *Deep Learning*. MIT Press. ISBN 978-0262035613. `http://www.deeplearningbook.org`.

Haarnoja, T.; Zhou, A.; Abbeel, P. & Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Dy, J. & Krause, A., editores, *Proceedings of the 35th International Conference on Machine*

Learning, volume 80 of *Proceedings of Machine Learning Research*, pp. 1861--1870. PMLR.

Hachaïchi, Y.; Chemingui, Y. & Affes, M. (2020). A policy gradient based reinforcement learning method for supply chain management. In *2020 4th International Conference on Advanced Systems and Emergent Technologies (IC_ ASET)*, pp. 135–140.

Henderson, P.; Islam, R.; Bachman, P.; Pineau, J.; Precup, D. & Meger, D. (2018). Deep reinforcement learning that matters. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1).

Hill, A.; Raffin, A.; Ernestus, M.; Gleave, A.; Kanervisto, A.; Traore, R.; Dhariwal, P.; Hesse, C.; Klimov, O.; Nichol, A.; Plappert, M.; Radford, A.; Schulman, J.; Sidor, S. & Wu, Y. (2018). Stable baselines. `https://github.com/hill-a/stable-baselines`.

Hutse, V. (2019). Reinforcement Learning for Inventory Optimisation in multi-echelon supply chains. Master in business engineering, `http://lib.ugent.be/catalog/rug01:002790831`, Ghent University.

Kemmer, L.; von Kleist, H.; de Rochebouët, D.; Tziortziotis, N. & Read, J. (2018). Reinforcement learning for supply chain optimization. In *European Workshop on Reinforcement Learning*, volume 14.

Kingma, D. P. & Ba, J. (2017). Adam: A method for stochastic optimization. *arXiv preprint:1412.6980*.

Laumanns, M. & Woerner, S. (2017). Multi-echelon supply chain optimization: Methods and application examples. In Póvoa, A. P. B.; Corominas, A. & de Miranda, J. L., editores, *Optimization and Decision Support Systems for Supply Chains*, pp. 131--138, Cham. Springer International Publishing.

Lee, H. L.; Padmanabhan, V. & Whang, S. (1997). The bullwhip effect in supply chains. *Sloan management review*, 38:93–102.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D. & Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint:1509.02971*.

Mazyavkina, N.; Sviridov, S.; Ivanov, S. & Burnaev, E. (2021). Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134:105400. ISSN 0305-0548.

Mitchell, T. (1997). *Machine Learning*. McGraw-Hill Education. ISBN 9780070428072.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D. & Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In Balcan, M. F. & Weinberger, K. Q., editores, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1928--1937, New York, New York, USA. PMLR.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G. et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529--533.

Morales, M. (2020). *Grokking Deep Reinforcement Learning*. Manning Publications. ISBN 9781617295454.

Mortazavi, A.; Arshadi Khamseh, A. & Azimi, P. (2015). Designing of an intelligent self-adaptive model for supply chain ordering management system. *Engineering Applications of Artificial Intelligence*, 37:207 – 220. ISSN 0952-1976.

Oroojlooyjadid, A. (2019). *Applications of Machine Learning in Supply Chains*. Tese de doutorado, `https://preserve.lehigh.edu/etd/4364`, Lehigh University.

Peng, Z.; Zhang, Y.; Feng, Y.; Zhang, T.; Wu, Z. & Su, H. (2019). Deep reinforcement learning approach for capacitated supply chain optimization under demand uncertainty. In *2019 Chinese Automation Congress (CAC)*, pp. 3512–3517.

Perez, H. D.; Hubbs, C. D.; Li, C. & Grossmann, I. E. (2021). Algorithmic approaches to inventory management optimization. *Processes*, 9(1). ISSN 2227-9717.

Pinedo, M. L. (2009). *Planning and Scheduling in Manufacturing and Services*. Springer-Verlag New York. ISBN 978-1-4419-0909-1.

Pochet, Y. & Wolsey, L. A. (2006). *Production planning by mixed integer programming*. Springer Science & Business Media. ISBN 978-0-387-29959-4.

Powell, W. B. (2019). A unified framework for stochastic optimization. *European Journal of Operational Research*, 275(3):795 – 821. ISSN 0377-2217.

Raffin, A. (2018). Rl baselines zoo. `https://github.com/araffin/rl-baselines-zoo`.

Raffin, A. (2020). Rl baselines3 zoo. `https://github.com/DLR-RM/rl-baselines3-zoo`.

Raffin, A.; Hill, A.; Ernestus, M.; Gleave, A.; Kanervisto, A. & Dormann, N. (2019). Stable baselines3. `https://github.com/DLR-RM/stable-baselines3`.

Rumelhart, D. E.; Hinton, G. E. & Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533--536.

Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M. & Moritz, P. (2015). Trust region policy optimization. In Bach, F. & Blei, D., editores, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1889--1897, Lille, France. PMLR.

Schulman, J.; Moritz, P.; Levine, S.; Jordan, M. & Abbeel, P. (2018). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint:1506.02438*.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A. & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint:1707.06347*.

Seelenmeyer, S. & Kißler, A. (2020). Multimodal logistics - application of optimization methods and future usage of artificial intelligence. `https://co-at-work.zib.de/slides/Freitag_25.9/SAP_Optimization_Multimodal_Logistics_Slides.pdf`. Presented at CO@Work 2020 Summer School.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M. et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484--489.

Stadtler, H.; Stadtler, H.; Kilger, C.; Kilger, C.; Meyr, H. & Meyr, H. (2015). *Supply chain management and advanced planning: concepts, models, software, and case studies*. Springer Texts in Business and Economics. Springer, Berlin, Heidelberg.

Sterman, J. D. (1989). Modeling managerial behavior: Misperceptions of feedback in a dynamic decision making experiment. *Management science*, 35(3):321--339.

Sutton, R. S. & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press. ISBN 9780262039246.

Apêndice A

Estudo de Caso Inicial

Este capítulo apresenta a primeira aplicação que fizemos do algoritmo PPO para resolver uma versão mais simples do problema abordado nesta tese. Nesta versão, as demandas são sempre regulares (não-sazonais) e os tempos de espera constantes, e, além disso, o algoritmo foi aplicado em um único cenário utilizado como estudo de caso. Para evitar repetições, ao invés de uma apresentação completa do problema e da metodologia, o texto do presente capítulo descreve as diferenças em relação ao problema geral descrito no Capítulo 2 e também em relação à metodologia apresentada no Capítulo 4, Os resultados aqui descritos foram publicados em um artigo científico [Alves & Mateus, 2020], apresentado na ICCL 2020, *International Conference on Computational Logistics*, e incluído nos anais da conferência que foram publicados na série *Lecture Notes in Computer Science*.

O capítulo está organizado como se segue. Na Seção A.1 são apresentadas as diferenças na definição do problema e na sua modelagem em relação ao problema mais geral tratado na tese. Já as seções A.2 e A.3 apresentam, respectivamente, a metodologia experimental e os experimentos realizados. Por fim, as conclusões do presente capítulo são apresentadas na Seção A.4.

A.1 Problema e Modelagem

O problema tratado neste capítulo é uma versão mais simples do problema apresentado no Capítulo 2. As diferenças basicamente são:

1. As demandas são sempre regulares (não-sazonais).

2. Os tempos de espera são sempre contantes, e têm valor igual a dois.

3. As fábricas possuem capacidade de processamento ilimitada.

4. Os estoques das fábricas são de produto (pós-processamento).

A formulação PDM também possui diferenças, tanto pelas simplificações do problema, como também por ter sido uma primeira formulação que depois foi aprimorada. Em relação ao espaço de estados, a formulação apresentada no artigo [Alves & Mateus, 2020] é mais genérica, podendo servir para redes de quaisquer tamanhos, enquanto a formulação apresentada na Seção 2.3 é focada em uma cadeia de quatro estágios e dois nós por estágio. De qualquer forma, em termos práticos, como o estudo de caso é feito em uma cadeia com essa configuração, a definição do espaço de estados é compatível. Além disso, no artigo a normalização dos estados é parte da formulação PDM, enquanto no Capítulo 4 ela foi apresentada separadamente na metodologia (Seção 4.2.2). Há um diferença na normalização, já que aqui o material em transporte é normalizado pela capacidade de estoque do destino. Mas isso pode gerar valores maiores que um, caso a quantidade de material que esteja chegando dos dois nós do estágio anterior seja maior que a capacidade de estoque. Isso foi tratado de forma mais adequada na definição apresentada na Seção 4.2.2, considerando-se a soma das capacidades de estoque das origens.

Em relação ao espaço de ações, assim como para os estados, a definição no artigo [Alves & Mateus, 2020] é mais genérica, podendo servir para redes de quaisquer tamanhos, enquanto a formulação apresentada na Seção 2.3 é focada na configuração de rede efetivamente experimentada. Mas, em termos práticos, não há diferença entre as formulações considerando o estudo de caso realizado.

A dinâmica do ambiente, dada pela simulação da cadeia, tem como diferenças o fato das fábricas terem capacidade ilimitada de processamento, e dos estoques das fábricas serem de produto e, portanto, armazenarem material após o processamento. Na versão completa do problema, apresentada no Capítulo 2, os estoques de das fábricas são de matéria-prima e, portanto, anteriores ao processamento. Essa mudança foi feita porque em um cenário real, com tempos de espera estocásticos, não faria sentido a fábrica não ter um *buffer* para amortecer a grande variação na chegada de materiais. Sem isso, muito material seria perdido por exceder a capacidade de fábrica e isso estaria fora do controle do planejamento de produção. Consideramos que essa seria uma decisão de configuração da cadeia, tomada em um nível estratégico, e seria anterior às decisões de planejamento de produção.

Finalizando a formulação PDM, a definição das recompensas utilizada aqui é a mesma já definida na Seção 2.3.

Já sobre o modelo PL com parâmetros incertos, no artigo [Alves & Mateus, 2020] foi apresentado um modelo mais simples, já que as demandas são sempre regulares e os tempos de espera constantes e iguais a dois. Mas como o modelo PL apresentado na Seção 2.4 é mais geral, ou seja, também resolve o estudo de caso tratado aqui, preferimos não apresentar o modelo neste capítulo.

Por fim, em relação ao agente PL usado como *baseline* nos experimentos, a única diferença aqui é o tratamento da normalização do material em transporte na construção do agente, pela diferença na representação do estado citada anteriormente.

A.2 Metodologia Experimental

Um dos objetivos do presente estudo de caso é verificar se uma abordagem de AR Profundo pode resolver, com bons resultados, o problema de operação da cadeia de suprimentos com demandas incertas (não-sazonais). Escolhemos o algoritmo PPO pelo seu bom desempenho em muitos problemas. Para usar o algoritmo é necessário implementar a simulação do ambiente, ou seja, da operação da cadeia de suprimentos, seguindo a formalização do problema apresentada na seção A.1.

Depois de escolher o algoritmo e implementar o ambiente, o próximo passo é fazer o ajuste dos hiperparâmetros do algoritmo. A metodologia proposta consiste em usar 100 diferentes combinações de valores (gerados aleatoriamente) para os hiperparâmetros. Para cada combinação o agente é treinado por mil episódios (com avaliações a cada 50 episódios). Os valores dos parâmetros usados na tentativa com os melhores resultados são usados nos experimentos.

O experimento consiste em usar o PPO para resolver um cenário de uma cadeia de suprimentos com quatro estágios e dois nós por estágio, com os parâmetros dados pela Tabela A.1[1]. Todos os custos são por unidade de matéria-prima ou produto.

Para verificar a robustez dos resultados alcançados por algoritmos como o PPO, é importante usar diferentes sementes de números aleatórios no processo de treinamento. A metodologia proposta é treinar o algoritmo dez vezes. Cada treinamento consiste em executar o algoritmo por dez mil episódios, avaliando o modelo a cada 50 episódios (considerando dez episódios em cada passo de avaliação). O melhor modelo encontrado em cada treinamento é usado para avaliar os resultados. A avaliação dos resultados consiste em simular o ambiente por 100 episódios para cada modelo encontrado no

[1] O custo de demanda não atendida c^d foi considerado como $3c^q$ onde c^q é o custo total de operação para entregar uma unidade de produto. O valor de c^q foi calculado como 72 para o cenário apresentado (considerando os custos mais altos de fornecimento e processamento, os custos totais de transporte e custos de estoque por oito períodos).

Tabela A.1. Parâmetros do cenário usado nos experimentos.

Grupo	**Parâm.**	**Valor**	**Detalhes**
Cadeia	N	1,...,8	2 fornecedores, 2 fábricas, 2 distribuidores e 2 varejistas (nesta ordem)
Horizonte	H	360	tamanho do episódio
Custos	c_n^s	1	para todos os nós
	c_n^p	6,4	para cada fornecedor, respectivamente
	c_n^f	12,10	para cada fábrica, respectivamente
	c^t	2	em toda a cadeia
Custos Penal.	c^e	10	material descartado por excesso de estoque
	c^d	216	por demanda não atendida
Capacidades	b_n^s	200, 300	para cada par de nós do mesmo estágio
	b_n^p	120, 150	para cada fornecedor, respectivamente
	b_n^t	200, 300	para cada par de nós do mesmo estágio
Razão de proces.	r_n	3	para fábricas (1 para demais nós)
Tempo de espera	l	2	para toda a cadeia
Config. cadeia	t_{nm}	*	modelando rede como da Figura 1.1
	f_n	1	para fábricas (0 para os demais)
Valores iniciais	s_n	0	para todos os nós
	p_{in}	60	para cada fornecedor
(definido para	t_{inm}	60	se m é fornecedor ou fábrica
$i \leq l$)	t_{inm}	20	se m distribuidor ou varejista
Demandas	d_{in}	[10,20]	amostrada aleatoriamente para cada varejista

processo de treinamento. Com esta abordagem, planeja-se executar um total de mil episódios de avaliação e a métrica de resultado é a média e o desvio padrão de todos esses episódios.

Um agente baseado no modelo PL é usado como um *baseline* a ser comparado com o PPO. Para criar este agente, o modelo PL é resolvido para uma versão determinística do problema (considerando o cenário proposto, e valores de demandas constantes e iguais à demanda média esperada, ou seja, 15 no cenário apresentado). A solução encontrada no modelo PL é codificada em ações do agente e, dessa forma, o agente PL interage com o mesmo cenário usado pelo PPO, com demandas estocásticas. Com esta abordagem as ações do agente PL causam flutuações nos estoques dos varejistas, mas, como as demandas são amostradas aleatoriamente no intervalo $[10, 20]$, espera-se que o agente possa obter bons resultados.

A.3 Experimentos e Resultados

Os experimentos foram feitos usando a linguagem Python 3.7.8 em um notebook Dell Vostro com processador 2.4 GHz x 4, com 3.7 GB de RAM e Ubuntu Linux 20.04. Foi usada a implementação do PPO da biblioteca Stable Baselines 2.10 [Hill et al., 2018], e a biblioteca RL Baselines Zoo [Raffin, 2018] foi usada para o ajuste dos hiperparâmetros. A simulação da cadeia de suprimentos (o ambiente) foi implementada em Python seguindo o padrão OpenAI Gym (Brockman et al. [2016]). O modelo PL foi resolvido usando CPLEX 12.10 via interface Python.

Os passos propostos na metodologia para o ajuste dos hiperparâmetros, apresentados na seção A.2, foram seguidos. Os valores dos parâmetros foram amostrados aleatoriamente a partir de intervalos padrões dados pela biblioteca RL Baselines Zoo. Além dos parâmetros tratados pela biblioteca, foi incluída a parametrização da arquitetura das redes neurais, considerando três opções, com camadas internas contendo 64, 128 ou 256 nós. Durante o processo, tentativas não promissoras foram abortadas usando regra de poda por mediana, ou seja, uma tentativa era podada se tivesse um resultado intermediário pior do que a mediana dos resultados intermediários das tentativas anteriores. Os melhores valores encontrados para os parâmetros são apresentados na Tabela A.2.

Tabela A.2. Melhores valores encontrados para os parâmetros do PPO no ajuste de hiperparâmetros.

Parâmetro	**Valor**	**Descrição**
N	4	Número de atores (fixo)
`n_steps`	128	Número de passos para coletar trajetórias
`n_epochs`	50	Número de épocas
`learning_rate`	1.48e-5	Taxa de aprendizado
`gamma`	0.95	Fator de desconto
`gae_lambda`	0.99	Compromisso entre viés e variância das vantagens
`nminibatches`	1	Número de minilotes de treinamento por atualização
`cliprange`	0.2	Parâmetro de *clipping*
`ent_coef`	0.03452	Coeficiente de entropia
`net_arch`	[256, 256]	Camadas internas das redes neurais

Os experimentos foram feitos de acordo com a metodologia apresentada na Seção A.1 e usando os valores de hiperparâmetros ajustados do PPO. A Tabela A.3 mostra a comparação entre os agentes PPO e PL. A primeira coluna mostra a semente de números aleatórios usada para o ambiente (simulação da cadeia) e a segunda coluna apresenta um limite inferior considerando as demandas geradas com a semente aleatória

correspondente (o limite foi obtido resolvendo-se o modelo PL usando as demandas, depois que todas elas foram geradas). A terceira e a quarta colunas apresentam a média e o desvio padrão do custo total de operação obtidos pelo agente PL, ao passo que as duas últimas colunas mostram os mesmos dados para o agente PPO. No caso do agente PL os valores se referem à média de 10 episódios, e no caso do PPO os mesmos 10 episódios foram avaliados, mas para cada um dos 10 treinamentos (perfazendo um total de 100 episódios de avaliação). Os resultados mostram que o PPO é um pouco melhor que o agente PL em todas as sementes usadas. A diferença média para o limite inferior (ou seja, a solução ótima se todas as demandas fossem conhecidas de antemão) é de 6.2% para o PPO e 7.6% para o agente PL. É interessante notar que o PPO tem baixa variância em todas as execuções, o que demonstra a robustez dos resultados. Uma outra comparação entre os agentes é apresentada na Tabela A.4. A primeira coluna mostra os tipos de custo, a segunda e a terceira colunas apresentam os valores para os agentes PL e PPO, respectivamente, e a quarta coluna mostra a diferença (valor do PPO menos valor do agente PL). Os resultados mostram que o ganho do PPO vem pelo fato de ter um melhor atendimento à demanda (a despeito de ter um custo de operação maior, já que ao final o custo adicional vale a pena).

Tabela A.3. Comparação dos agentes PL e PPO no cenário proposto. PPO tem custos totais de operação menores em todas as avaliações.

Semente	**Limite inferior**		**Agente PL**		**Agente PPO**	
	Avg	σ	Avg	σ	Média	σ
1	552.270	4.829	596.937	9.480	585.628	4.535
2	550.200	4.702	595.402	6.654	584.807	5.339
3	552.321	4.535	593.015	6.716	585.986	4.149
4	552.240	4.791	591.691	6.266	586.327	4.852
5	551.402	3.659	597.336	4.678	586.258	3.895
6	554.136	4.985	596.335	10.817	588.066	4.757
7	551.918	2.948	588.668	6.677	586.932	3.063
8	552.115	5.391	593.984	6.804	586.174	5.500
9	549.470	4.164	592.375	9.310	583.728	4.561
10	552.077	2.257	591.973	9.335	586.212	3.173
Média	**551.815**	**4.228**	**593.772**	**8.308**	**586.012**	**4.585**

Para investigar em mais detalhes o comportamento do agente PPO no cenário proposto, nós sumarizamos, por período, as demandas não atendidas e a quantidade de material (matéria-prima e produto) em transporte e em estoque de todas as rodadas de avaliação; e o resultado é mostrado na Figura A.1. Todos os valores se referem à média da soma total para toda a cadeia. O gráfico superior esquerdo mostra as demandas não

atendidas por período e o gráfico superior direito apresenta as demandas não atendidas acumuladas; ambos os agentes têm valores muito baixos por período, mas o agente PPO é melhor e os valores acumulados enfatizam a diferença entre os agentes. Analisando o gráfico inferior esquerdo, pode-se notar que o agente PL mantém uma quantidade de material constante em transporte, como esperado (depois de liberar os produtos iniciais e antes do fim do horizonte), e o agente PPO, em média, aprende a manter o mesmo nível de transporte. Finalmente, o gráfico inferior direito mostra que o agente PPO tende a manter, na média, um nível constante de estoque, enquanto o agente PL tende a aumentar a quantidade de produto em estoque (considerando a média de todas as avaliações). Pode-se perceber que o nível de estoque do agente PL aumenta de acordo com as demandas não atendidas acumuladas.

Tabela A.4. Comparação dos agentes PL e PPO por tipos de custo (em média). O ganho do PPO vem do melhor atendimento à demanda.

Tipo de custo	**Agente PL**	**Agente PPO**	**dif.**
Fornecimento	125.760	129.264	3.504
Estoque	22.056	21.475	-581
Processamento	319.440	324.657	5.217
Transporte	106.080	108.034	1.954
Demanda não atendidas	20.436	2.581	-17.855
Penalidades de estoque	0	0	0
Total	**593.772**	**586.012**	**-7.760**

A curva de aprendizado do agente PPO (média para os 10 treinamentos) é mostrada na Figura A.2. O gráfico mostra a médias das recompensas acumuladas por episódio. Pode-se notar que o treinamento é estável e converge após cerca de quatro mil episódios. O tempo médio de execução das rodadas de treinamento foi de cerca de 3,5 horas. É importante notar que não foi usada paralelização, uma vez que o foco do trabalho foi no comportamento do algoritmo. Se o tempo total de execução for um fator importante, seria simples obter melhores tempos usando paralelização em um ambiente com GPUs.

A.4 Conclusões

O estudo de caso apresentado neste capítulo, usa uma abordagem de Aprendizado por Reforço Profundo, nomeadamente o algoritmo PPO, para operar uma cadeia de suprimentos com demandas estocásticas. O cenário experimental consiste em uma cadeia com quatro estágios e dois nós por estágio. A cada período um agente de AR

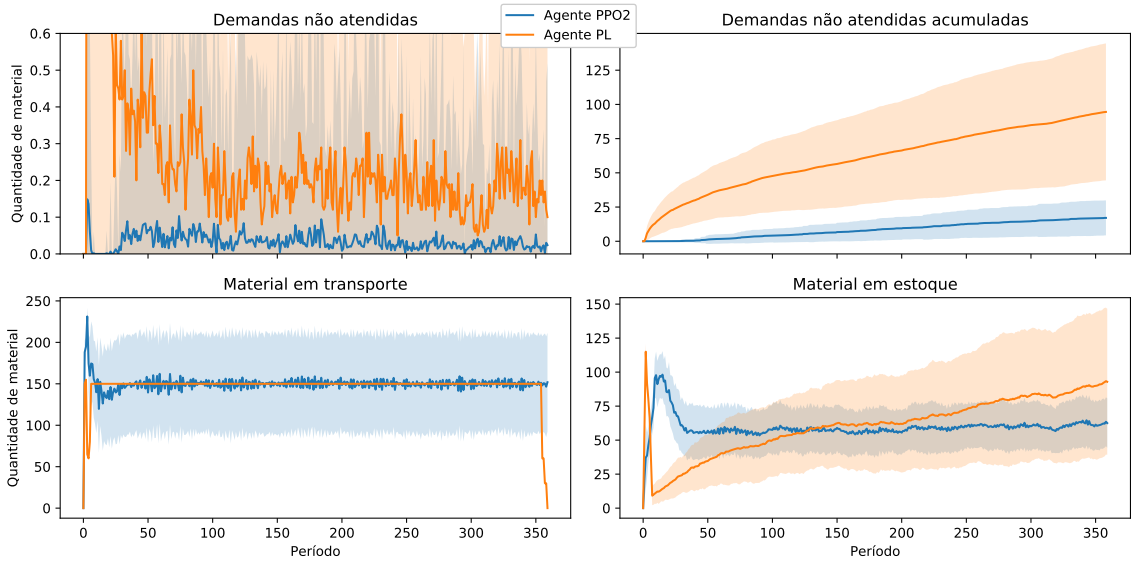

Figura A.1. Comparação do desempenho dos agentes PL e PPO por período (valores se referem à soma para toda a cadeia e são a média de todas as rodadas de avaliação).

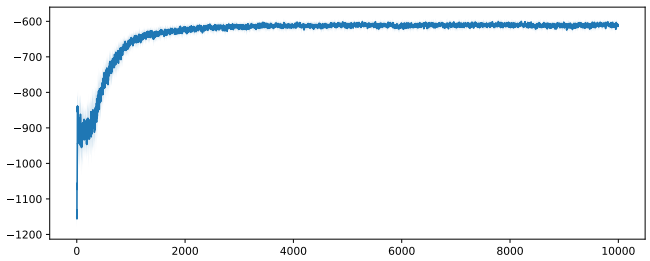

Figura A.2. Curva de aprendizado média do PPO usando o cenário proposto.

precisa decidir a quantidade de matéria-prima a ser produzida pelos fornecedores e a quantidade de material que cada nó deve enviar para os nós do próximo estágio (usando esta abordagem, os níveis de estoque são definidos de forma indireta). O objetivo é atender demandas incertas de clientes nos nós do último estágio e minimizar todos os custos envolvidos (fornecimento, estoque, transporte, processamento e penalizações por demanda não atendida e excesso de estoque). As decisões tomadas pelo agente de AR são mapeadas em um espaço contínuo ao invés de se usar níveis agregados, como é comum para esse tipo de problema na literatura. Pelo nosso conhecimento, este é o primeiro trabalho que usa um algoritmo de AR Profundo para uma cadeia de suprimentos multiestágio com mais de um nó por estágio, e, portanto, usa espaços de estados e ações tão grandes. Uma outra contribuição é resolver o problema deste tamanho sem limitar as possíveis decisões do agente, por usar espaço contínuo de ações

ao invés da abordagem comum de níveis de agregação discretizados.

Para verificar a qualidade e a robustez do algoritmo PPO usado no problema proposto, a metodologia experimental proposta foi seguida, fazendo-se o ajuste dos hiperparâmetros e realizando-se múltiplos treinamentos com diferentes sementes de números aleatórios. Um modelo de Programação Linear, considerando a demanda esperada média, é usado para construir um agente usado como *baseline*. As ações do agente PL são construídas a partir da solução do modelo de Programação Linear. O agente PPO obtém resultados ligeiramente melhores em termos de custos totais de operação, principalmente devido ao melhor atendimento à demanda, com baixa variância em todas as diferentes sementes utilizadas, demonstrando que é uma abordagem competitiva para resolver este tipo de problema.

Apêndice B

Definição das Recompensas

B.1 Recompensas como o Inverso do Custo

Como citado no Capítulo 2, na fase inicial dos experimentos nós chegamos a considerar o uso do inverso dos custos operacionais a cada período como a função de recompensa. Este apêndice fornece mais detalhes sobre essa avaliação.

Como os experimentos realizados eram exploratórios, os cenários utilizados tinham características um pouco diferentes dos apresentados anteriormente. De toda forma, o principal foco é destacar a diferença nos resultados ao considerar as recompensas como o negativo ou como o inverso do custo.

Foram realizados experimentos com um cenário com demanda regular e quatro com demanda sazonal. O primeiro cenário é o mesmo do estudo de caso usado nos experimentos apresentados no Apêndice A. Já os cenários com demandas sazonais possuem quatro picos como os utilizados no Capítulo 4, mas com senoidal base com valor mínimo 250 e máximo 350 (ao invés dos valores 100 e 300), e com menor nível de perturbação das demandas. Além disso, estes cenários ainda consideravam que os estoques das fábricas eram de produto, como na versão do problema utilizada no Apêndice A. Os experimentos foram realizados com o algoritmo PPO, com a versão da biblioteca Stable Baselines 2 [Hill et al., 2018], e utilizando os valores padrões para os hiperparâmetros. Foram consideradas três rodadas de treinamento de 10 mil episódios.

A Tabela B.1 apresenta um resumo dos resultados encontrados. Em cada linha é apresentado um cenário. A primeira coluna indica o tipo de demanda, a segunda a perturbação da demanda e a terceira se os tempos de espera são constantes ou estocásticos. A quarta coluna traz os resultados ao considerar a recompensa como o negativo do custo e a quinta como o inverso do custo. Por fim, a última coluna apresenta, para referência, os resultados considerando um agente com política aleatória.

Analisando os dados podemos notar que o uso das recompensas como o inverso do custo levou a resultados muito ruins, chegando a ser pior do que um agente com política aleatória.

Tabela B.1. Experimentos preliminares para avaliação da recompensa como o inverso do custo. Como pode ser visto, os resultados foram muito ruins, piores do que um agente com política aleatória.

Tipo Demanda	Nível Pert.	Tempo Esp. Estoc.	Recompensa Negativo	Recompensa Inverso	Agente Aleatório
regulares		constante	0,6 M	1,4 M	1,2 M
sazonais	10	constante	8,2 M	21,1 M	17,1 M
sazonais	20	constante	8,5 M	24,5 M	17,1 M
sazonais	10	estocástico	12,6 M	26,1 M	18,1 M
sazonais	20	estocástico	12,6 M	27,2 M	18,2 M

A Figura B.1 apresenta as curvas de aprendizado para os experimentos com o cenário com demandas sazonais, nível de perturbação $\mathcal{N}(0, 10)$ e tempos de espera constantes. O gráfico de cima mostra os resultados considerando as recompensas como o negativo do custo e o gráfico debaixo como o inverso de custo. Podemos notar que no primeiro gráfico as curvas são bem comportadas, subindo rapidamente no início dos treinamentos e se estabilizando em um patamar mais alto. Já quando a recompensa é o inverso do custo as curvas apresentam comportamento ruim. Mesmo levando-se em conta que os valores têm escala diferente, já que são o inverso dos custos, vemos em primeiro lugar que são necessários cerca de cinco mil episódios para que os valores saiam do patamar mais baixo. Mas o principal ponto é que a curva não se estabiliza ao final do treinamento, tendo picos de valores bem mais altos em alguns episódios.

A partir desses resultados, levantamos a hipótese de que os resultados são ruins quando as recompensas são o inverso do custo porque os valores de recompensa acabam sendo muito baixos e muito próximos de zero. Isso poderia prejudicar o desempenho de algoritmos que usam RNAs já que os valores são muito baixos em relação à distribuição Normal com média zero e desvio um que geralmente são utilizados por algoritmos como o PPO. Além disso, os valores de recompensa seriam muito próximos entre si, o que poderia gerar sinais fracos para o agente conseguir diferenciar entre ações boas e ruins. A partir disso, decidimos fazer novos experimentos considerando a multiplicação de uma constante pelo inverso do custo. Para a definição do valor da constante, analisamos os resultados dos melhores modelos gerados pelos experimentos com o agente treinado usando recompensa como negativo do custo, e verificamos que em 95% dos períodos o custo é maior do que 750. Experimentamos então o valor 750 como a constante, ou seja,

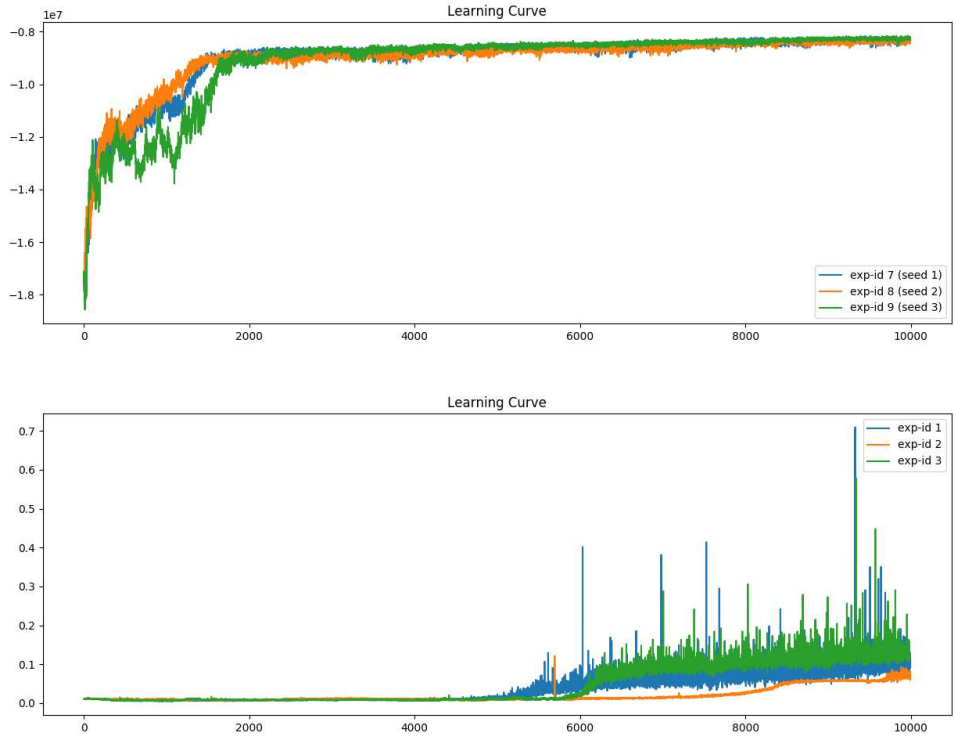

Figura B.1. Curvas de aprendizado para o cenário com demandas sazonais, perturbação $\mathcal{N}(0, 10)$ e tempos de espera constantes. O gráfico de cima apresenta os resultados para experimentos com recompensas como o negativo dos custos e o gráfico de baixo considerando o inverso dos custos.

considerando que c é a soma dos custos em um período, as que recompensas são dadas por $\frac{750}{c}$. Fizemos uma rodada de treinamento, usando o cenário de demandas regulares e os resultados ficaram ainda piores (passaram de 1,4 M para 1,6 M). Aumentamos então o tempo de treinamento para 40 mil episódios e consideramos a normalização automática de recompensas considerando média móveis. Os resultados ficaram em torno de 1,3 M, e a curva de aprendizado (Figura B.2) continuou mal comportada.

Dados os resultados apresentados nesta seção, e considerando os resultados que já tínhamos obtido considerando a recompensa como o negativo do custo, decidimos descartar o uso de recompensas como o inverso do custo.

B.2 Recompensas Somente ao Final do Episódio

Durante a fase de experimentos preliminares, chegamos a considerar também a definição de recompensas como o negativo dos custos totais de operação. Com essa abordagem, as recompensas são diferentes de zero somente ao final do episódio. A argumentação

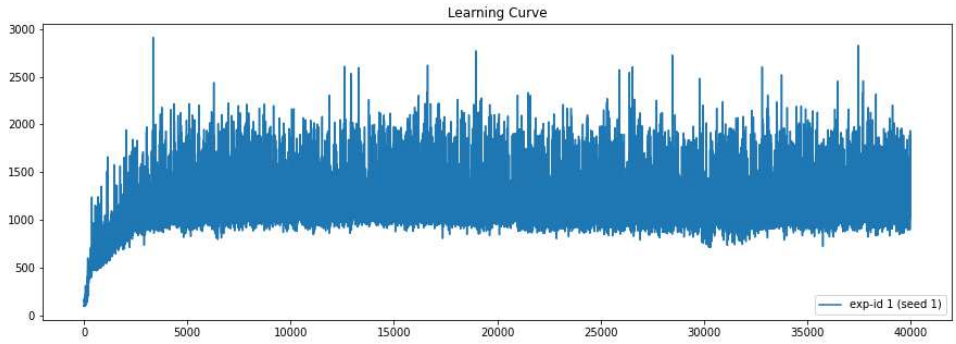

Figura B.2. Curva de aprendizado de treinamento com 40 mil episódios, com recompensas dadas por 750 dividido pelo custo e usando normalização de recompensas.

para usar essa abordagem é que, no problema abordado, estamos interessados no custo total de operação, não importando como ele é distribuído ao longo do horizonte de planejamento. Por outro lado, recompensas definidas dessa forma são muito esparsas o que prejudica o desempenho de métodos de AR, já que o agente recebe um sinal apenas ao final do episódio. De qualquer forma, quisemos fazer os experimentos porque talvez essa abordagem chegasse a valores finais melhores, mesmo que talvez precisasse de mais tempo de treinamento.

Experimentamos uma rodada de treinamento de 10 mil episódios, com o PPO da biblioteca Stable Baselines 2, considerando os parâmetros padrões do algoritmo. As curvas de aprendizado para o cenário com demandas regulares e um dos cenários com demandas sazonais são apresentados nas figuras B.3 e B.4, respectivamente. Podemos notar que as curvas se estabilizam em patamares muito ruins. Talvez os resultados pudessem ser melhores com treinamentos bem maiores, mas optamos por não continuar nessa linha de investigação por dois motivos. Primeiro que as curvas se estabilizam em certos patamares; se elas não tivessem se estabilizado, poderia ser que com mais treinamento elas começassem a subir. E, em segundo lugar, nós já tínhamos resultados bem melhores considerando as recompensas como o negativo do custo. Dessa forma, decidimos descartar o uso de recompensas somente ao final do episódio.

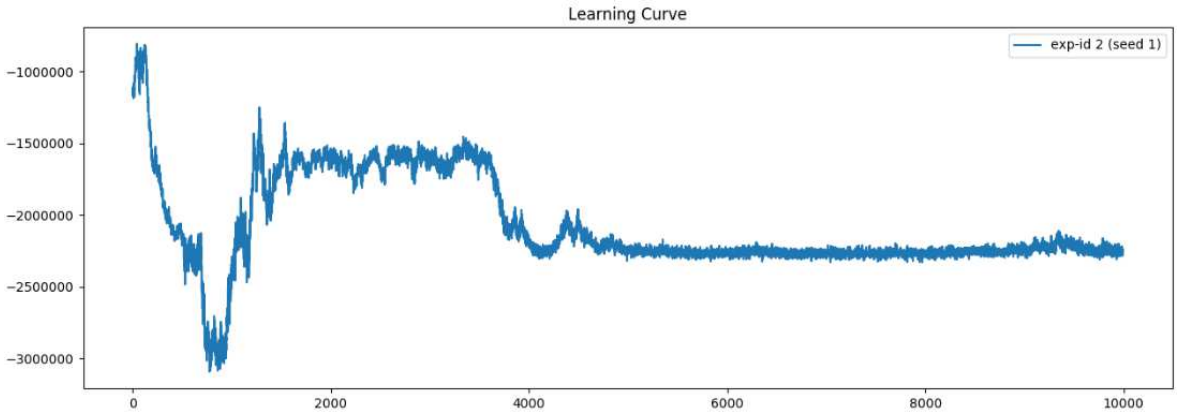

Figura B.3. Curva de aprendizado para treinamento em cenário com demanda regular, considerando recompensas dadas apenas ao final do episódio (como o negativo dos custos totais).

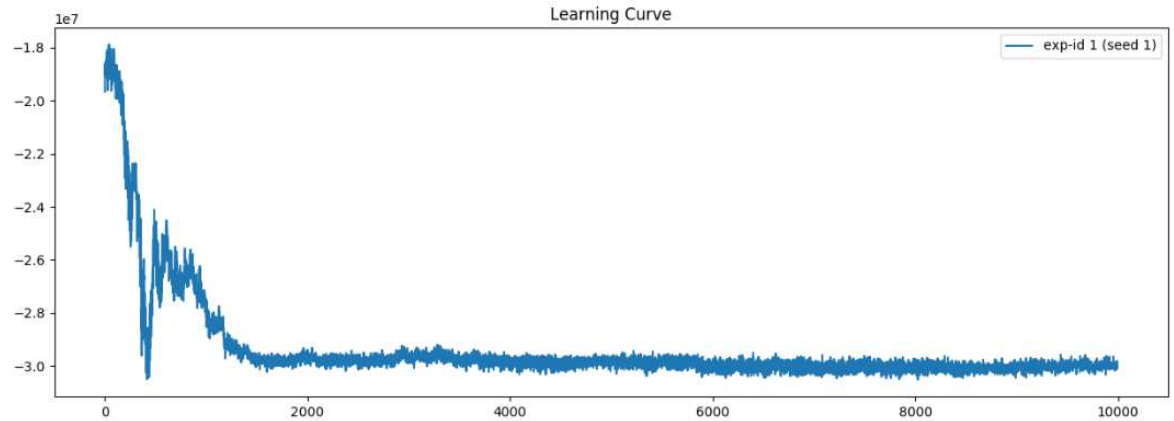

Figura B.4. Curva de aprendizado para treinamento em cenário com demanda sazonal, considerando recompensas dadas apenas ao final do episódio (como o negativo dos custos totais).