Faculdade de Engenharia da Universidade do Porto

Reinforcement Learning para problemas de otimização

Sérgio António Moreira Fernandes

VERSÃO FINAL

Dissertação realizada no âmbito do
Mestrado Integrado em Engenharia Electrotécnica e de Computadores
Major Automação

Orientador: Prof. Luís Filipe Ribeiro dos Santos Guimarães

26 de Julho de 2019

Resumo

A gestão de stock é uma área extremamente importante para as cadeias de abastecimento, já que esta decisão controla todo o fluxo de material do sistema de modo a obter resultados desejados. Existem vários modelos de gestão de stock, sendo que o objetivo principal de todos eles é realizar as melhores encomendas possíveis de forma a minimizar os custos acarretados e maximizar o lucro obtido.

O objetivo desta dissertação é implementar um algoritmo de *Reinforcement Learning* que permita efetuar uma gestão de stock adequada, comparando-o com as técnicas tradicionais de gestão de stock. O *Reinforcement Learning* é uma área de *machine learning* que é treinada a partir de dados anteriores que levam a uma aprendizagem que permite otimizar resultados futuros. Existem também vários tipos de *Reinforcement Learning*, tendo sido destacado o *Q-Learning*, que se baseia numa matriz Q onde são definidos estados e ações.

Deste modo, é fundamental determinar estados e ações, bem como outros parâmetros relativos ao algoritmo de *Reinforcement Learning* implementado, que levem à otimização do resultado da função objetivo da gestão de stock.

O algoritmo desenvolvido apresentou, em média, resultados melhores que os outros métodos, embora exista ainda espaço para melhorias e implementação de especificações adicionais.

Abstract

Inventory Management is an extremely important area of supply chains, since it allows to control the flow of materials in the entire system to achieve the desired results. There are several models of inventory management and in every one of them the main objective is to place the best possible orders in order to minimize the resulting cost and increase the obtained profit.

The intention in this dissertation is to implement a Reinforcement Learning algorithm that allows for an adequate management of inventory, comparing it with the traditional inventory management techniques. Reinforcement Learning is an area of machine learning which is trained based on past data, learning the decisions that allow the optimization of future results. There are also several types of Reinforcement Learning, with the Q-Learning algorithm being selected for this study, which is based on a Q matrix where states and actions are defined and optimized.

Therefore, determining states and actions, as well as other parameters connected to the implemented Reinforcement Learning algorithm, that lead to optimal results of the inventory management objective function is essential.

The implemented algorithm had better results than the other methods, on average, although there is still room for improvement and implementation of additional specifications.

Agradecimentos

Em primeiro lugar, queria agradecer ao meu orientador Professor Doutor Luís Guimarães pelos conselhos e a ajuda e disponibilidade que demonstrou ao longo da realização desta dissertação.

Um forte abraço para os amigos que conheci ao longo deste curso, bem como para aqueles que conheci em contextos diferentes. Obrigado pelos momentos que partilhamos e pelo apoio que proporcionaram quer ao longo do meu percurso académico, quer fora dele.

Por último, mas com certeza não menos importante, queria agradecer aos meus pais e restante família. Obrigado pelo ensino, pelo carinho e por todos os valores que me transmitiram que me permitem ser a pessoa que sou hoje.

Sérgio Fernandes

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e símbolos

LT	*Lead Time*
MDP	*Markov Decision Process*
RT	*Review Time*
SL	*Service Level*

Capítulo 1

Introdução

O presente capítulo permite fazer uma introdução à dissertação "*Reinforcement Learning* para problemas de otimização". Inicialmente, na secção 1.1, é efetuado o enquadramento do tema e são apresentados os principais objetivos e motivações. Na secção 1.2 é exposta a estrutura da dissertação.

1.1. Enquadramento, objetivos e motivação

Com os avanços da tecnologia na área da eletrónica, automação e robótica, tem vindo a expandir-se o uso de inteligência artificial para a resolução de problemas complexos, permitindo a obtenção de resultados ótimos com pouca ou sem intervenção humana. Uma área de inteligência artificial que apenas se tornou mais reconhecida recentemente é o *Reinforcement Learning*.

Reinforcement Learning é uma área de *Machine Learning* baseada num agente que toma decisões interagindo com um ambiente. Em qualquer altura, o sistema encontra-se num estado e existe um momento em que o agente decide qual é a ação a tomar. Ao realizar essa ação, recebe uma recompensa e o sistema transita para um novo estado. A probabilidade de transitar para esse estado designa-se por probabilidade de transição. O objetivo principal é maximizar as recompensas recebidas, permitindo que o algoritmo aprenda a realizar esta otimização por si mesmo.

Os problemas de gestão de stock, que são abordados nesta dissertação na sua componente de problema de otimização, têm como objetivo encontrar o momento e quantidade ótima para um conjunto de encomendas num determinado horizonte temporal, de forma a minimizar os custos e maximizar o lucro obtido. A gestão de stock na cadeia de abastecimento é normalmente realizada separadamente em cada uma das atividades da cadeia de

abastecimento: Fornecimento, Produção e Distribuição. Em cada uma das três atividades é verificado o stock existente e é decidido se é necessário fazer uma encomenda à atividade anterior. Caso a atividade anterior não possua stock suficiente para satisfazer a encomenda, esta fica em *backorder*, isto é, em atraso, o que incorre em custos para o sistema.

Uma possível solução para este tipo de problemas é a utilização de *Reinforcement Learning*. Esta poderá ser uma boa abordagem já que otimiza o valor da função objetivo tendo em conta resultados obtidos no passado, permitindo ao algoritmo uma melhoria constante ao longo do tempo.

Neste contexto, o objetivo desta dissertação é efetuar a abordagem de problemas multi-período de gestão de stock, recorrendo a um algoritmo de *Reinforcement Learning*, combinado com conhecimento de domínio especializado, de forma a criar uma metodologia de otimização baseada em dados.

O tipo de *Reinforcement Learning* escolhido foi o *Q-Learning*, um algoritmo baseado numa matriz Q, composta por estados e ações. Este será implementado com base em dados simulados, isto é, dados que não são reais, e o principal objetivo deste algoritmo será obter resultados melhores que os algoritmos tradicionais de gestão de stock.

1.2. Estrutura da dissertação

Para além da introdução, esta dissertação é composta por mais 5 capítulos.

Inicialmente, no capítulo 2 é descrito o estado da arte. De seguida, o capítulo 3 consiste na caracterização do problema em questão. Depois, no capítulo 4 é descrita a metodologia desenvolvida. O capítulo 5 consiste na apresentação e descrição dos testes efetuados e dos resultados obtidos. Por fim, no capítulo 6 são apresentadas as principais conclusões e possíveis sugestões de melhoria.

Capítulo 2

Revisão Bibliográfica

2.1. Machine Learning

Machine Learning é uma área da inteligência artificial que permite a implementação de algoritmos e modelos estatísticos que possam prever ações futuras, com base em dados obtidos anteriormente. Esta área está subdividida em diversos tipos de aprendizagem: *Supervised Learning*, *Unsupervised Learning*, *Reinforcement Learning*, *Feature Learning*, entre outros.

2.2. Reinforcement Learning

Reinforcement Learning é uma área de *Machine Learning* e pode ser comparado com a forma como uma criança aprende a realizar uma nova tarefa. Se esta realizar a tarefa de forma adequada, é recompensada de forma positiva, senão não recebe nenhuma recompensa. Ao longo do tempo, a criança vai aprender a realizar a tarefa de modo a receber a sua recompensa. Este tipo de aprendizagem funciona dessa forma e destaca-se de outras áreas de *Machine Learning* devido a ser realizada sem supervisão e de forma distinta do *Unsupervised Learning*.

Sucintamente, o *Reinforcement Learning* baseia-se num Processo de Decisão Markov (MDP) [1], em que existe um conjunto de estados, ações e recompensas. Quando o agente se encontra num determinado estado, pode realizar um conjunto de ações diferentes e, dependendo da ação que realiza, recebe uma recompensa. Essa ação e recompensa afetam o estado seguinte, sendo que o objetivo principal é maximizar a recompensa total. Este processo pode ser observado na Figura 2.1.

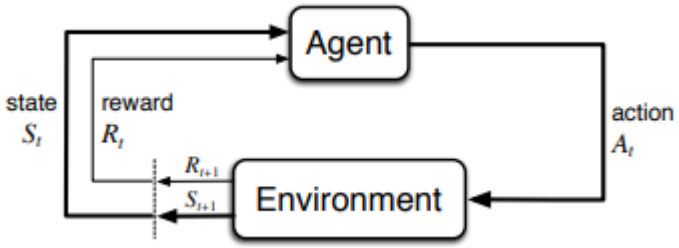

Figura 2.1 - Interação agente-ambiente num MDP [2]

O *Reinforcement Learning* apareceu inicialmente a inícios dos anos 80, tendo sido iniciado com aprendizagem através de tentativa e erro em animais. Ao longo dos anos, foram muitos os contributos para o desenvolvimento do *Reinforcement Learning*, mas destaca-se em 1972 o trabalho de Klopf que juntou a aprendizagem por tentativa e erro à aprendizagem de diferença temporal, criando a ideia de *reinforcement*, em que bons inputs são recompensados e maus inputs são punidos, tendo este trabalho sido depois mais aprofundado por Barto e Sutton em 1981 e 1982 [2].

Existem diversos algoritmos de *Reinforcement Learning*: *Q-Learning*, *State-Action-Reward-State-Action* (SARSA), *Deep Q Network* (SQN) e *Deep Deterministic Policy Gradient* (DDPG).

Após uma curta análise dos algoritmos e das características do problema em questão, o *Q-Learning* foi selecionado para ser utilizado, uma vez que as suas características se adequavam ao problema em questão, apresentando um grau de complexidade menor do que outros algoritmos como o SQN e o DDPG.

2.2.1. Q-Learning

Q-Learning é uma técnica específica de *Reinforcement Learning* que aprende um procedimento e maximiza a recompensa obtida sem necessitar de um modelo do ambiente, através do uso de uma matriz Q em que as linhas correspondem ao conjunto de estados possíveis e as colunas correspondem ao conjunto de ações disponíveis.

A fórmula de atualização dos valores da matriz Q é a seguinte, designada por equação de Bellman [3],

$$\boldsymbol{Q}(\boldsymbol{s},\boldsymbol{a}) := \boldsymbol{Q}(\boldsymbol{s},\boldsymbol{a}) + \boldsymbol{\alpha} \times [\boldsymbol{r}(\boldsymbol{t}+\boldsymbol{1}) + \boldsymbol{\gamma}\,\mathbf{max}_{a'}\,\boldsymbol{Q}(\boldsymbol{s}',\boldsymbol{a}') - \boldsymbol{Q}(\boldsymbol{s},\boldsymbol{a})], \tag{2.1}$$

onde *s* corresponde ao estado atual; *a* corresponde à ação atual; α é a taxa de aprendizagem, sendo que esta é definida entre 0 e 1. Um valor mais baixo significa que o algoritmo aprende mais lentamente e um valor elevado permite um treino mais rápido; *r* representa a recompensa que o algoritmo recebe dependendo do seu desempenho; *γ* é o fator de desconto, definido entre 0 e 1. Um fator de desconto reduzido dá menos importância a recompensas futuras,

focando-se no presente, enquanto um fator de desconto mais elevado coloca mais ênfase nas recompensas futuras; *s'* representa o estado seguinte, resultante da ação realizada no estado atual; *a'* representa a ação seguinte.

Esta fórmula é aplicada recorrendo a um algoritmo, que pode ser observado na Figura 2.2, que é corrido um grande número de vezes (por exemplo 10 000), chamada a fase de treino, após inicialização de todos os valores da matriz Q a um valor arbitrário, normalmente 0. Inicialmente, como existe pouca informação sobre qual a melhor ação dependendo do estado, é mais provável que se faça *exploration* de forma a obter mais informação sobre o ambiente. Nesse caso, é escolhida uma ação ao acaso de entre as ações disponíveis no estado atual. Assim, a probabilidade de *exploitation* é iniciada com um valor reduzido. Com o passar das iterações, essa probabilidade aumenta, uma vez que queremos melhorar a informação que já foi obtida [4] e a melhor forma de o fazer é explorar a ação mais bem treinada até ao momento. A certa altura, a probabilidade de *exploitation* atinge 1 e é sempre melhorada a melhor ação.

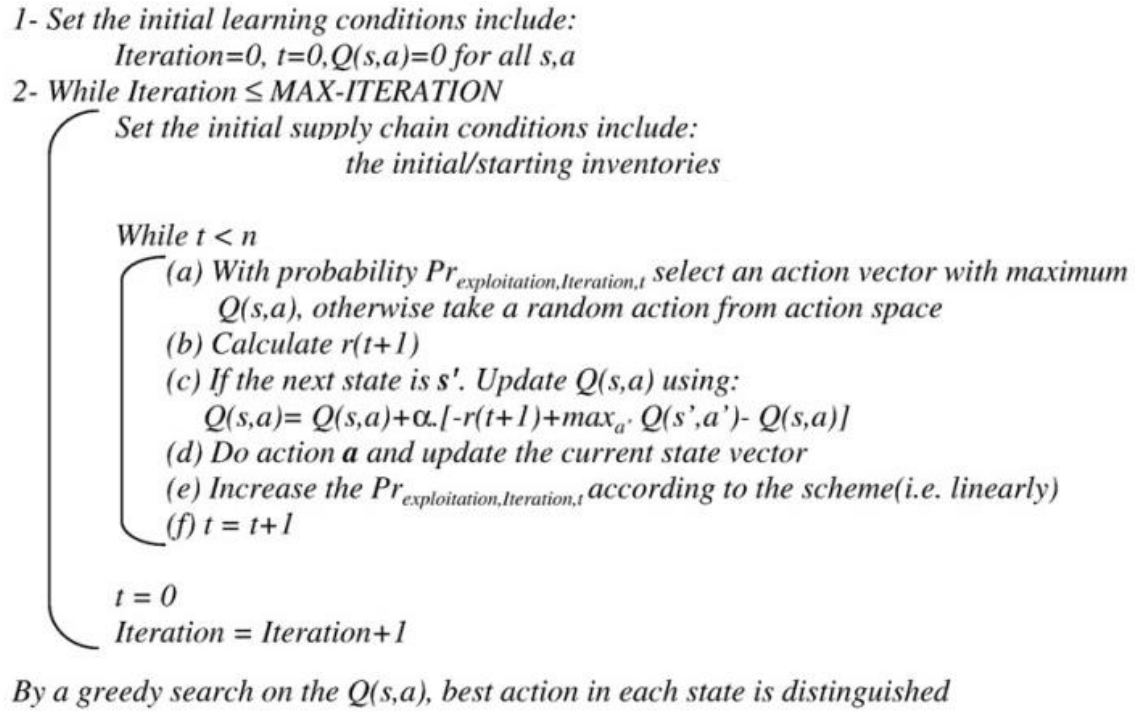

Figura 2.2 - Algoritmo utilizado para aplicação de *Q-Learning* [5]

Após terminado este algoritmo e otimizada a matriz Q, segue-se a fase de teste em que é resolvido o problema. Em qualquer estado que o sistema se encontre, é realizada uma procura de qual a ação com maior valor na matriz Q e essa é a ação a tomar, já que essa será a que otimizará a função objetivo, de acordo com o treino.

O *Q-Learning* foi inicialmente introduzido por Christopher Watkins em 1989 [6], tendo sido provado mais tarde por Tsiksiklis em 1994 [7] e Tsitsiklis e Bertsekas em 1996 [8].

Em 2014 a Google DeepMind patenteou um algoritmo de *Q-Learning* [9] que consegue jogar Atari 2600 a um nível semelhante a jogadores humanos de alto nível.

2.3. Gestão de stock

A gestão de stock consiste em gerir o stock disponível de modo a minimizar os custos e maximizar o lucro da empresa em questão. Para isso, é necessário calcular a altura ideal, bem como a quantidade ideal de stock a encomendar, de forma a minimizar o custo total, sabendo o preço de armazenamento de stock e o preço dos atrasos devido a falta de stock (rutura). Explicado simplesmente, "a gestão de stock é ter a quantidade certa, do stock certo, no local certo, na altura certa e ao custo certo" [10].

Tal é um problema uma vez que o gestor está dependente da procura do consumidor, muitas vezes bastante irregular, o que leva a incertezas no momento de decidir quando e quanto encomendar.

Existem vários tipos de modelos para a resolução de problemas deste tipo. Dois desses modelos, sendo ambos bastante comuns, são o modelo (s,Q) [11] e o modelo (R,S) [12].

O modelo (s,Q), representado na Figura 2.3 é um modelo de análise contínua, isto é, o nível de stock é monitorizado constantemente, e consiste em encomendar uma quantidade Q quando o nível de stock atinge o ponto de encomenda s. A encomenda é sempre recebida um tempo constante (*lead time*) depois de ter sido realizada.

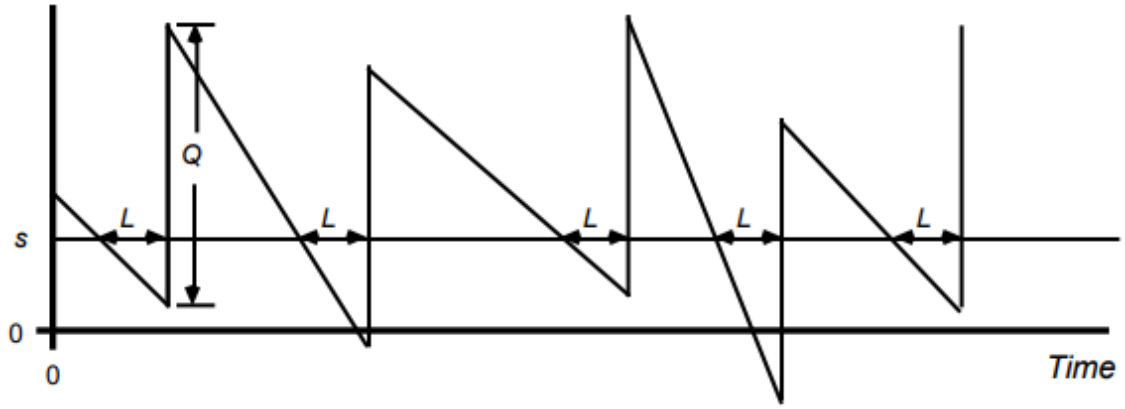

Figura 2.3 - Modelo de gestão de stock (s,Q)

O modelo (R,S) é um modelo de análise periódica, já que o nível de stock apenas é verificado entre certos intervalos de tempo R. Este consiste em realizar encomendas que reabastecem o stock até ao nível de encomenda S. Ou seja, se o nível de stock estiver a um certo valor x (posição de stock), a quantidade encomendada nessa altura será S-x. Este comportamento pode ser observado na Figura 2.4.

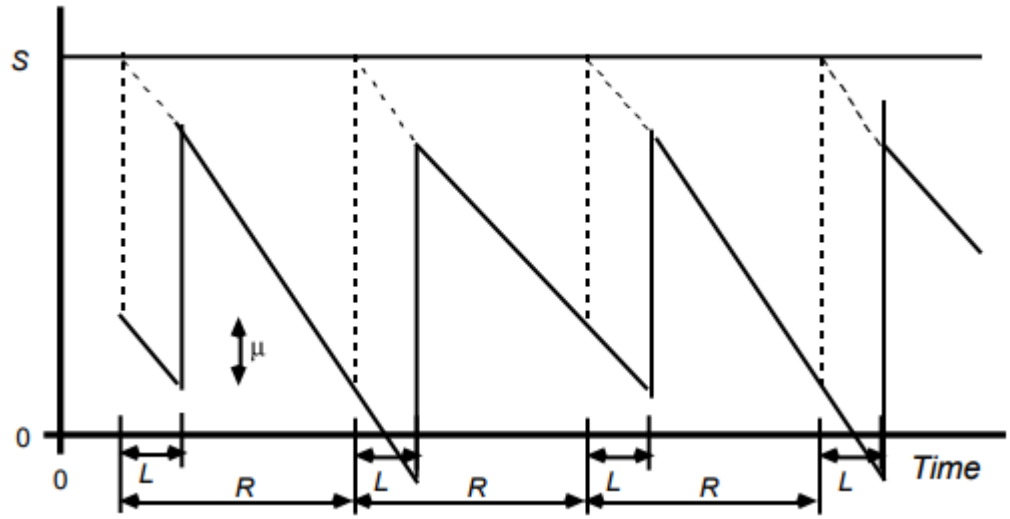

Figura 2.4 - Modelo de gestão de stock (R,S)

2.3.1. Stock de segurança

O ponto de encomenda s é composto pelo stock de ciclo e pelo stock de segurança e é calculado da seguinte forma:

$$s = (RT + LT) \times \bar{d} + ss \tag{2.2}$$

Para calcular o stock de segurança, existem o *Alpha Service Level* e o *Beta Service Level* [13]. A formula para calcular o stock de segurança (*ss*) é a seguinte:

$$ss = \sqrt{RT + LT} \times \sigma_d \times k \tag{2.3}$$

O *Alpha Service Level* determina o número mínimo de encomendas que são satisfeitas. Isto é, se existir um *Alpha Service Level* de 95%, no máximo apenas 5% das encomendas podem entrar em rutura. Para atingir um *Alpha Service Level* de 95%, teria que ser utilizado k = 1.96.

O *Beta Service Level* determina a quantidade máxima que entra em rutura. Se existir um *Beta Service Level* de 90%, no máximo apenas 10% da quantidade de stock pode entrar em rutura. Para calcular o k necessário para obter um *Beta Service Level* desejado, o processo é um pouco diferente, não sendo tão direto. Primeiro, calcula-se o valor de G(k), onde F é o valor de Beta pretendido, a partir da seguinte fórmula:

$$G(k) = \frac{\bar{d}}{\sigma_d} \times \frac{1 - F}{F} \tag{2.4}$$

De seguida, é necessário ir à tabela de G(k) [14] e verificar qual o valor de k correspondente ao valor obtido. Esse k será depois utilizado na fórmula de cálculo do stock de segurança.

No entanto, estes e outros modelos não avaliam custos do sistema tendo apenas a preocupação em garantir o nível de serviço desejado. Assim, estas abordagens mais tradicionais levam a momentos em que o stock existente é demasiado alto ou demasiado baixo, acarretando custos elevados para a empresa em questão. Neste contexto, torna-se útil a utilização de *Reinforcement Learning* para realizar esta otimização.

2.4. Reinforcement Learning na gestão de stock

Para aplicar o *Reinforcement Learning* na gestão de stock, começa-se por definir a cadeia de abastecimento normalmente, incluindo as três atividades principais mencionadas anteriormente: Fornecimento, Produção e Distribuição.

Após concluído esse processo, esse modelo é codificado num MDP, sendo possível abordar esse processo recorrendo ao algoritmo apresentado anteriormente.

Para aplicar este algoritmo, na criação do processo de decisão Markov é necessário definir a função de recompensa e codificar os estados e ações possíveis. Estes podem ser definidos de várias formas.

A forma mais comum de definir a função de recompensa é torná-la igual à função objetivo, visto que é o que deve ser otimizado, recompensando assim o algoritmo por bons resultados da função objetivo e penalizando-o por maus resultados.

Já os estados e as ações são mais dependentes do problema em questão e devem ser bem definidos, uma vez que se não for esse o caso os resultados obtidos não serão os melhores. Podem ser efetuados testes onde são comparadas diversas definições dos estados da matriz, permitindo obter assim a melhor definição que leva aos melhores resultados.

Existem vários exemplos de aplicações de *Reinforcement Learning* na gestão de stock. Um deles pode ser observado no artigo "*Application of Multi-agent Reinforcement Learning to Supply Chain Ordering Management*" [15]. Nesse artigo é aplicado um algoritmo de *Q-Learning* idêntico ao apresentado a uma cadeia de abastecimento. A principal diferença relativamente ao problema proposto é a existência das várias atividades da cadeia de abastecimento, enquanto no nosso caso apenas é considerada uma atividade, não existindo interação com outras atividades.

Capítulo 3

Caracterização do problema

3.1. Definição do problema

O problema em questão consiste num problema de otimização de gestão de stock em que o objetivo é minimizar o custo total, composto pelo custo de armazenamento e pelo custo de rutura, decidindo as alturas e quantidades ideais de stock a encomendar.

Após a implementação de um algoritmo de *Reinforcement Learning*, é fundamental a comparação do mesmo com os métodos tradicionais de gestão de stock, de modo a estudar a viabilidade/adequabilidade do algoritmo implementado.

Relativamente aos dados do problema, assumiu-se uma procura com distribuição normal e valores de RT e LT constantes ao longo de cada instância.

3.2. Parâmetros

Tiveram que ser definidos alguns parâmetros comuns a todos os métodos de gestão de stock utilizados, sendo eles os seguintes:

- T – nº de dias analisados
- RT - tempo entre duas avaliações sucessivas do nível de stock (dias)
- LT - tempo entre realizar e receber uma encomenda (dias)
- h – custo de armazenamento de stock (unidade)
- b – custo de rutura (unidade)
- $\theta = \frac{b}{h}$
- I0 – stock inicial (unidades)
- $\bar{d}$ – procura média (unidades por dia)

- σ_d – desvio padrão da procura (unidades por dia)
- CV = $\frac{\sigma_d}{\bar{d}}$

Sabendo estes valores, foram calculados para cada dia I_t, que corresponde ao stock disponível no final do dia t; NI_t, que corresponde ao stock líquido (*on-hand* + *in-transit*) no início do dia t; Q_t, correspondente à quantidade de stock a encomendar no dia t. Estes serão os valores necessários para calcular o valor da função objetivo. Esta é dada pela fórmula

$$\min \sum_{t=0}^{T-1} (h \times I_t^+ + b \times I_t^-) , \tag{3.1}$$

onde

$$I_t^+ = \begin{cases} I_t, & I_t \geq 0 \\ 0, & I_t < 0 \end{cases} \tag{3.2}$$

$$I_t^- = \begin{cases} 0, & I_t \geq 0 \\ -I_t, & I_t < 0 \end{cases} \tag{3.3}$$

e I_t^+ é o stock *on-hand* no dia t e I_t^- é a rutura no dia t. Foi definido ainda que quando existe rutura, as unidades de stock correspondentes são perdidas completamente, não existindo a possibilidade de se verificarem atrasos.

Capítulo 4

Metodologia desenvolvida

4.1. Solução proposta

A solução proposta para o problema em questão passa por aplicar *Reinforcement Learning*, mais concretamente *Q-Learning*, para otimizar as quantidades encomendadas, de modo a minimizar a função objetivo.

Inicialmente, calculam-se as quantidades a encomendar sem utilizar qualquer tipo de *Reinforcement Learning* e depois calculam-se essas quantidades com o uso de *Reinforcement Learning*, de modo a obter uma comparação entre os dois métodos e verificar se, de facto, o algoritmo implementado leva a uma melhor solução.

Tendo em conta a solução em si, foi seguido o algoritmo de *Q-Learning* apresentado na Figura 2.2.

4.2. Gestão de Stock

O método de gestão de stock utilizado foi o (R,S), recorrendo a um *Alpha Service Level* de 95% e um *Beta Service Level* de 90% para o cálculo do stock de segurança das técnicas tradicionais utilizadas para comparação com o *Reinforcement Learning*.

Para obter um *Alpha Service Level* de 95% foi utilizado k = 1.96 e para obter um *Beta Service Level* de 90% foi utilizado k = -1.03 quando CV = 0.1 e k = 0.06 quando CV = 0.3. Estes valores foram calculados a partir das Equações 2.3 e 2.4.

4.3. Reinforcement Learning

O método *Q-Learning* segue o algoritmo apresentado na Figura 2.2. Como tal, foi também necessário definir alguns parâmetros específicos a este algoritmo:

- α – Taxa de aprendizagem = 0.1
- γ – Fator de desconto = 0.2
- Probabilidade de *exploitation* = 0.15 (incrementada após cada iteração até o seu valor atingir 1)
- Número de iterações (treino da matriz Q) = 1500

Para efetuar a aplicação deste algoritmo, teve ainda de ser criada uma matriz (Q) composta por estados e ações.

Os estados são as linhas da matriz e dependem do nível de stock líquido NI_t no momento de revisão de stock e foram definidos com base na procura média, RT e LT. Os intervalos de estado para estado foram definidos após tentativa e erro, sendo testados mais tarde de modo a verificar se existiam intervalos que permitissem obter melhores resultados.

- Estado 0: $NI_t \leq \bar{d} \times LT + \bar{d} \times RT \times 0.05$
- Estado 1: $\bar{d} \times LT + \bar{d} \times RT \times 0.05 < NI_t \leq \bar{d} \times LT + \bar{d} \times RT \times 0.1$
- Estado 2: $\bar{d} \times LT + \bar{d} \times RT \times 0.1 < NI_t \leq \bar{d} \times LT + \bar{d} \times RT \times 0.15$
- Estado 3: $\bar{d} \times LT + \bar{d} \times RT \times 0.15 < NI_t \leq \bar{d} \times LT + \bar{d} \times RT \times 0.2$
- Estado 4: $\bar{d} \times LT + \bar{d} \times RT \times 0.2 < NI_t \leq \bar{d} \times LT + \bar{d} \times RT \times 0.25$
- Estado 5: $\bar{d} \times LT + \bar{d} \times RT \times 0.25 < NI_t \leq \bar{d} \times LT + \bar{d} \times RT \times 0.3$
- Estado 6: $\bar{d} \times LT + \bar{d} \times RT \times 0.3 < NI_t \leq \bar{d} \times LT + \bar{d} \times RT \times 0.5$
- Estado 7: $\bar{d} \times LT + \bar{d} \times RT \times 0.5 < NI_t \leq \bar{d} \times LT + \bar{d} \times RT \times 0.75$
- Estado 8: $NI_t > \bar{d} \times LT + \bar{d} \times RT \times 0.75$

As ações correspondem às colunas da matriz e determinam a quantidade a encomendar Q_t no momento de revisão de stock e foram distribuídas por 25 ações diferentes, definidas em função da procura média e RT. O valor mínimo possível a encomendar foi de 0 e o máximo foi o dobro da procura média multiplicada pelo *review time*. As ações foram ainda distribuídas de forma que mais ações se encontrem perto do valor médio, sendo mais espaçadas conforme se vão aproximando dos extremos da escala.

- Ação 0: $Q_t = \bar{d} \times RT \times 2$
- Ação 1: $Q_t = \bar{d} \times RT \times 1.75$
- Ação 2: $Q_t = \bar{d} \times RT \times 1.5$

- Ação 3: $Q_t = \bar{d} \times RT \times 1.4$
- Ação 4: $Q_t = \bar{d} \times RT \times 1.3$
- Ação 5: $Q_t = \bar{d} \times RT \times 1.25$
- Ação 6: $Q_t = \bar{d} \times RT \times 1.2$
- Ação 7: $Q_t = \bar{d} \times RT \times 1.15$
- Ação 8: $Q_t = \bar{d} \times RT \times 1.1$
- Ação 9: $Q_t = \bar{d} \times RT \times 1.075$
- Ação 10: $Q_t = \bar{d} \times RT \times 1.05$
- Ação 11: $Q_t = \bar{d} \times RT \times 1.025$
- Ação 12: $Q_t = \bar{d} \times RT$
- Ação 13: $Q_t = \bar{d} \times RT \times 0.975$
- Ação 14: $Q_t = \bar{d} \times RT \times 0.95$
- Ação 15: $Q_t = \bar{d} \times RT \times 0.925$
- Ação 16: $Q_t = \bar{d} \times RT \times 0.9$
- Ação 17: $Q_t = \bar{d} \times RT \times 0.85$
- Ação 18: $Q_t = \bar{d} \times RT \times 0.8$
- Ação 19: $Q_t = \bar{d} \times RT \times 0.75$
- Ação 20: $Q_t = \bar{d} \times RT \times 0.7$
- Ação 21: $Q_t = \bar{d} \times RT \times 0.6$
- Ação 22: $Q_t = \bar{d} \times RT \times 0.5$
- Ação 23: $Q_t = \bar{d} \times RT \times 0.25$
- Ação 24: $Q_t = 0$

A função de recompensa utilizada corresponde à função objetivo apresentada na Equação 3.1. Esta é calculada em cada iteração durante a fase treino e é resultante da ação tomada no estado atual, que resulta na transição para um novo estado.

De forma a clarificar melhor como é definida a matriz Q, apresenta-se aqui um exemplo com valores concretos. Os valores utilizados foram $\bar{d}$ = 100, RT = 1 e LT = 1. Com estes valores, os estados e ações obtidos são os seguintes:

- Estado 0: $NI_t \leq 105$
- Estado 1: $105 < NI_t \leq 110$
- Estado 2: $110 < NI_t \leq 115$
- Estado 3: $115 < NI_t \leq 120$
- Estado 4: $120 < NI_t \leq 125$
- Estado 5: $125 < NI_t \leq 130$
- Estado 6: $130 < NI_t \leq 150$
- Estado 7: $150 < NI_t \leq 175$

- Estado 8: $NI_t > 175$

- Ação 0: $Q_t = 200$
- Ação 1: $Q_t = 175$
- Ação 2: $Q_t = 150$
- Ação 3: $Q_t = 140$
- Ação 4: $Q_t = 130$
- Ação 5: $Q_t = 125$
- Ação 6: $Q_t = 120$
- Ação 7: $Q_t = 115$
- Ação 8: $Q_t = 110$
- Ação 9: $Q_t = 107.5$
- Ação 10: $Q_t = 105$
- Ação 11: $Q_t = 102.5$
- Ação 12: $Q_t = 100$
- Ação 13: $Q_t = 97.5$
- Ação 14: $Q_t = 95$
- Ação 15: $Q_t = 92.5$
- Ação 16: $Q_t = 90$
- Ação 17: $Q_t = 85$
- Ação 18: $Q_t = 80$
- Ação 19: $Q_t = 75$
- Ação 20: $Q_t = 70$
- Ação 21: $Q_t = 60$
- Ação 22: $Q_t = 50$
- Ação 23: $Q_t = 25$
- Ação 24: $Q_t = 0$

É de realçar que caso se tratasse de um problema real, não deveria haver quantidades decimais, mas como se tratam de dados simulados, assume-se a possibilidade de isso acontecer. Se tal não fosse possível, as ações teria que ser definidades de outra forma, de modo a impossibilitar a existência de quantidades decimais.

Capítulo 5

Testes e Resultados

5.1. Bibliotecas utilizadas

De forma a poder realizar os testes, foi necessário utilizar algumas bibliotecas com funções já existentes que permitissem a realização do trabalho dentro do tempo esperado. As bibliotecas de Python [16] utilizadas no desenvolvimento do algoritmo foram a Random, que permite retornar números aleatórios e foi fundamental na geração da procura normal que foi utilizada neste projeto; Numpy, que permite realizar operações com matrizes e *arrays*, sem a qual não seria possível implementar o algoritmo de *Reinforcement Learning*; Matplotlib, utilizada na criação de gráficos que permitiram analisar a evolução do valor da função objetivo ao longo do tempo; Math, composta por diversas funções matemáticas, sendo que esta foi utilizada uma vez que certas fórmulas utilizadas exigiam a utilização de raíz quadrada.

5.2. Definição das instâncias de teste

De formar a testar o algoritmo implementado, foi necessário atribuir valores aos parâmetros apresentados na Secção 3.2. Assim, os valores utilizados durante os testes efetuados foram os observados na Tabela 5.1. A combinação destes valores totalizou 504 instâncias diferentes a serem testadas.

Tabela 5.1 - Valores utilizados para as instâncias de teste

Parâmetro	Lista de valores testados
T	360 (180 treino, 180 teste)
RT	1, 2, 3, 5, 7, 15, 30
LT	1, 2, 3, 5, 7, 15
h	1
θ	0.5, 5, 20, 50, 100, 300
I0	100
$\bar{d}$	100
CV	0.1, 0.3

Foram geradas diversas procuras, todas elas com distribuição normal, $\bar{d}$ = 100 e CV = 0.1 ou CV = 0.3

5.3. Teste do algoritmo com matriz inicial

Utilizando a matriz composta pelos estados e ações apresentados na Secção 4.3 e os parâmetros apresentados na secção anterior, foram comparados os valores da função objetivo obtidos pelo *Reinforcement Learning* com os valores obtidos pelos métodos tradicionais (*Alpha Service Level* e *Beta Service Level*) em função do valor de θ. O algoritmo de *Reinforcement Learning* foi corrido 10 vezes para cada instância e foi depois efetuada a média, já que este algoritmo é estocástico.

Tabela 5.2 - Comparação dos resultados obtidos com a matriz inicial

	θ					
Método	**0.5**	**5**	**20**	**50**	**100**	**300**
Alpha SL	124,618.0	127,408.9	136,711.6	155,317.2	186,326.4	310,363.3
Beta SL	98,897.9	105,583.0	127,866.7	172,434.0	246,712.9	543,828.4
RL	14,292.8	83,123.1	115,324.6	144,077.6	181,416.1	315,595.4
Diferença	592%	27%	11%	7.8%	2.7%	-1.7%

Na Tabela 5.2 foi comparado o desempenho dos diferentes métodos para todas as instâncias. A última linha demonstra a diferença do desempenho do algoritmo de *Reinforcement Learning* comparativamente ao método tradicional com o resultado mais próximo. Foram ainda comparados os resultados obtidos para instâncias com LT > RT na Tabela 5.3 e os resultados obtidos para instâncias com LT ≤ RT na Tabela 5.4, de modo a compreender

se o algoritmo de *Reinforcement Learning* apresenta melhores resultados para algum desses cenários.

Tabela 5.3 - Comparação dos resultados obtidos com a matriz inicial para LT > RT

	θ					
Método	**0.5**	**5**	**20**	**50**	**100**	**300**
Alpha SL	50,474.7	54,963.7	69,927.0	99,853.5	149,731.0	349,241.0
Beta SL	30,483.4	39,707.3	70,453.5	131,946.0	234,433.5	644,383.5
RL	8,018.2	30,831.2	56,010.4	92,688.4	147,461.3	353,134.3
Diferença	280%	28.8%	24.8%	7.7%	1.5%	-1.1%

Tabela 5.4 - Comparação dos resultados obtidos com a matriz inicial para LT ≤ RT

	θ					
Método	**0.5**	**5**	**20**	**50**	**100**	**300**
Alpha SL	165,808.8	167,656.2	173,814.2	186,130.3	206,657.2	288,764.6
Beta SL	136,906.0	142,180.7	159,762.9	194,927.3	253,534.8	487,964.4
RL	17,778.6	112,174.2	148,276.9	172,627.2	200,279.9	294,740.5
Diferença	670%	26.7%	7.7%	7.8%	3.2%	-2.1%

Tendo em conta as tabelas apresentadas, conclui-se que o *Reinforcement Learning* se destaca sendo o melhor para valores baixos de θ, quer num caso, quer noutro. Verifica-se que um *Beta Service Level* de 90% é mais competitivo também para valores baixos de θ, apresentando resultados demasiado elevados para valores de θ mais altos. O *Alpha Service Level* de 95% apresenta resultados competitivos para a maioria dos casos, comparativamente ao *Reinforcement Learning*. Para θ = 300, o *Alpha Service Level* apresentou o melhor resultado. Assim, foram identificados casos específicos em que o *Reinforcement Learning* apresentou resultados menos bons e foram testadas diferentes matrizes e durações de período de treino.

5.4. Testes à matriz Q

Como existiram bastantes casos em que os resultados obtidos não foram satisfatórios, decidiu-se que se deviam testar diversas matrizes de forma a verificar qual delas apresentava melhor desempenho, sendo essa depois utilizada no algoritmo. De forma a testar a matriz foram efetuados testes com diversas procuras para os seguintes valores, já que estas foram algumas das instâncias com a pior performance do algoritmo de *Reinforcement Learning*:

- $\bar{d}$ = 100

- RT = 2, 30
- LT = 1, 2
- θ = 300
- CV = 0.1

Foram testadas oito matrizes diferentes, sendo que as ações foram sempre as mesmas em todas, iguais às ações iniciais, mas os estados definidos foram diferentes em todas as matrizes utilizadas. O algoritmo de *Reinforcement Learning* foi desta vez corrido 20 vezes, tendo sido efetuada a média de todos os valores resultantes, sendo esta depois comparada com o valor do *Alpha Service Level*, já que era o método mais competitivo. A matriz 1 corresponde à matriz inicial, definida na Secção 4.3. Os resultados obtidos foram os seguintes:

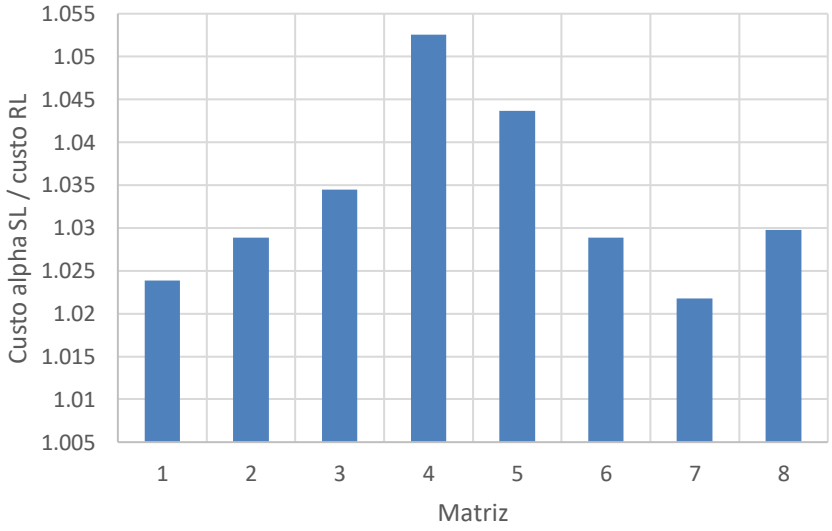

Figura 5.1 - Comparação do desempenho das diversas matrizes em relação ao custo total

A matriz que apresentou a melhor média de resultados foi a matriz designada de matriz 4 na Figura 5.1, já que foi com essa que se verificou a maior diferença entre o resultado do *Alpha Service* Level e o resultado do *Reinforcement Learning*. Os estados dessa matriz são os seguintes:

- Estado 0: $NI_t \leq \bar{d} \times LT$
- Estado 1: $\bar{d} \times LT < NI_t \leq \bar{d} \times LT + \bar{d} \times RT \times 0.1$
- Estado 2: $\bar{d} \times LT + \bar{d} \times RT \times 0.1 < NI_t \leq \bar{d} \times LT + \bar{d} \times RT \times 0.2$
- Estado 3: $\bar{d} \times LT + \bar{d} \times RT \times 0.2 < NI_t \leq \bar{d} \times LT + \bar{d} \times RT \times 0.3$
- Estado 4: $\bar{d} \times LT + \bar{d} \times RT \times 0.3 < NI_t \leq \bar{d} \times LT + \bar{d} \times RT \times 0.4$
- Estado 5: $\bar{d} \times LT + \bar{d} \times RT \times 0.4 < NI_t \leq \bar{d} \times LT + \bar{d} \times RT \times 0.5$
- Estado 6: $\bar{d} \times LT + \bar{d} \times RT \times 0.5 < NI_t \leq \bar{d} \times LT + \bar{d} \times RT \times 0.6$

- Estado 7: $\bar{d} \times LT + \bar{d} \times RT \times 0.6 < NI_t \leq \bar{d} \times LT + \bar{d} \times RT \times 0.75$
- Estado 8: $\bar{d} \times LT + \bar{d} \times RT \times 0.75 < NI_t \leq \bar{d} \times LT + \bar{d} \times RT \times 0.9$
- Estado 9: $NI_t > \bar{d} \times LT + \bar{d} \times RT \times 0.9$

Foi também adicionado um estado adicional à nova matriz, existindo agora 10 estados, um a mais do que os 9 iniciais.

5.5. Teste do período de treino

Além de testar a matriz utilizada, foi ainda testado o período de treino de forma a perceber a importância desse período e verificar qual o número de dias durante o qual a matriz Q deve ser treinada. Em princípio, esperava-se que uma duração mais longa do período de treino levasse não só a resultados melhores, em média, mas também a menos ocorrências de resultados discrepantes. A matriz utilizada durante este teste foi a matriz resultante do teste efetuado na secção anterior.

Para efetuar este teste foram utilizados os seguintes valores apresentados na Tabela 5.5, com os restantes valores mantendo-se iguais aos apresentados na Tabela 5.1.

Tabela 5.5 - Valores utilizados para as instâncias de teste do período de treino

Parâmetro	Lista de valores testados
Nº de dias de treino	30, 60, 90, 120, 150, 180, 210, 240, 270, 300, 330, 360
RT	1, 2, 5, 30
LT	1, 2, 5, 15
θ	0.5, 20, 300
CV	0.1

Para cada instância o *Reinforcement Learning* foi corrido novamente 20 vezes e depois efetuada a média dos valores resultantes. Os resultados obtidos por este método foram de seguida comparados com os resultados obtidos pelo *Alpha Service Level*.

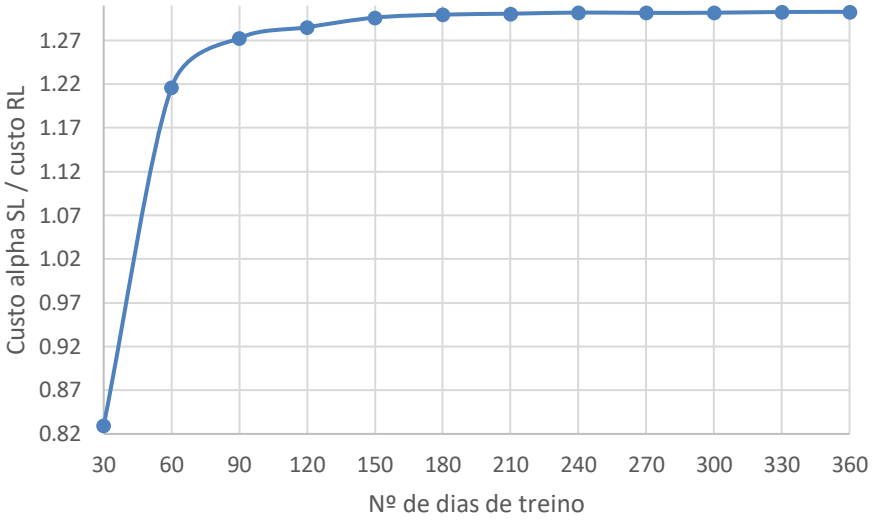

Figura 5.2 - Comparação do desempenho dos diversos períodos de treino em relação ao custo total

Da Figura 5.2 conclui-se que, em média, o treino da matriz Q é melhor quando é treinada durante mais tempo. Assim, o valor a utilizar para a duração do treino da matriz Q será de 360 dias. De notar que foi delimitada uma duração máxima de aproximadamente 1 ano para o treino da matriz. Se não fosse o caso, a matriz poderia ser treinada durante um período ainda mais longo.

5.6. Teste do algoritmo final

Terminada a escolha da matriz e do melhor período de treino, foi necessário realizar novos testes ao algoritmo de *Reinforcement Learning* de forma a verificar se os resultados obtidos de facto melhoraram. Os parâmetros utilizados foram os mesmos que foram apresentados na Secção 5.2, com exceção do valor de T que foi agora igual a 540, devido a ter sido escolhido um período de treino de 360 dias. O período de teste continua a ser de 180 dias.

Tabela 5.6 - Comparação dos resultados obtidos com nova matriz e duração de treino

	θ					
Método	**0.5**	**5**	**20**	**50**	**100**	**300**
Alpha SL	102,944.8	105,179.2	112,627.4	127,523.9	152,351.3	251,660.8
Beta SL	79,975.9	84,457.9	99,398.1	129,278.5	179,079.0	378,281.4
RL	11,168.3	65,386.0	92,117.8	115,119.0	145,913.9	252,073.2
Diferença	616%	29.2%	7.9%	10.8%	4.4%	-0.2%

Na Tabela 5.6 foi comparado o desempenho dos diversos algoritmos com a nova matriz e duração do período de treino. Tal como no procedimento efetuado para a matriz inicial,

foram ainda comparados os resultados obtidos para LT > RT na Tabela 5.7 e os resultados obtidos para LT ≤ RT na Tabela 5.8.

Tabela 5.7 - Comparação dos resultados obtidos com nova matriz e duração de treino para LT > RT

	θ					
Método	**0.5**	**5**	**20**	**50**	**100**	**300**
Alpha SL	43,837.5	47,524.6	59,815.1	84,396.1	125,364.5	289,237.8
Beta SL	25,589.7	32,120.6	53,890.1	97,429.1	169,994.1	460,254.1
RL	6,420.9	24,937.0	46,076.9	76,129.3	121,157.2	292,079.7
Diferença	298.5%	28.8%	17%	10.9%	3.5%	-1%

Tabela 5.8 - Comparação dos resultados obtidos com nova matriz e duração de treino para LT ≤ RT

	θ					
Método	**0.5**	**5**	**20**	**50**	**100**	**300**
Alpha SL	135,782.1	137,209.6	141,967.6	151,483.7	167,343.9	230,784.6
Beta SL	110,190.4	113,534.2	124,680.3	146,972.6	184,126.3	332,741.1
RL	13,805.8	87,857.7	117,696.1	136,780.0	159,667.6	229,847.4
Diferença	698%	29.2%	6%	7.5%	4.8%	0.4%

Verifica-se novamente que para θ = 300 o resultado do *Reinforcement Learning* é pior que o do *Alpha Service Level*. No entanto, para LT ≤ RT isto já não acontece. Assim, torna-se útil efetuar uma comparação entre os resultados obtidos nesta secção e aqueles obtidos na Secção 5.3. Para isso, como as procuras utilizadas nos dois casos foram distintas, serão comparados os rácios entre a média total dos custos do *Alpha Service Level*, *Beta Service Level* e do *Reinforcement Learning*.

Tabela 5.9 - Comparação dos resultados iniciais com os resultados finais

Método	**Inicial**	**Final**
Alpha SL	173,457.6	142,047.9
Beta SL	215,887.2	158,411.8
RL	142,304.9	113,629.7
Alpha SL / RL	1.219	1.250
Beta SL / RL	1.517	1.394

Como se pode verificar na Tabela 5.9, os resultados do *Reinforcement Learning* melhoraram 3.1%, em média, comparativamente aos resultados do *Alpha Service Level*, embora tenham piorado 12.3% face aos resultados do *Beta Service Level*. Isto seria de esperar e deve-se ao facto dos testes da matriz e da duração de treino terem sido efetuados com base no rácio entre

o custo do *Alpha Service Level* e o custo do *Reinforcement Learning*, tendo nesses casos sido ignorado o *Beta Service Level*.

Capítulo 6

Conclusões

6.1. Satisfação dos objetivos

O objetivo principal desta dissertação era implementar um algoritmo de *Reinforcement Learning* de forma a perceber se com este seria possível obter melhores resultados do que com os métodos tradicionais de gestão de stock.

No geral, este objetivo foi satisfeito, tendo sido implementado um algoritmo que, em média, apresentou resultados 25% melhores que o *Alpha Service level* e 39.4% melhores que o *Beta Service Level*.

No entanto, existem casos em que os algoritmos tradicionais ainda apresentam melhores resultados do que o algoritmo implementado, pelo que seria possível melhorar o trabalho efetuado até este momento.

6.2. Sugestões de melhoria

Nesta secção serão apresentadas algumas sugestões que poderiam melhorar os resultados obtidos pelo algoritmo implementado ou que o tornassem mais completo de forma a este poder ser aplicado a situações distintas. Assim, algumas sugestões seriam acrescentar outras variáveis aos estados da matriz Q, como por exemplo a previsão da procura para os dias seguintes, de forma a obter um melhor treino da matriz; incluir custos de encomenda, tornando o algoritmo mais abrangente; testar o algoritmo com outros tipos de procura, isto é, procura que não siga uma distribuição normal, de modo a verificar o seu desempenho com outras distribuições de procura e, caso necessário, efetuar alterações que melhorem os resultados obtidos; afinação dos parâmetros do algoritmo, de forma a otimizar os resultados; utilização de estados e ações mais específicos para cada conjunto de parâmetros, ao invés da utilização de parâmetros

abrangentes, de forma a obter melhores resultados para cada conjunto específico de parâmetros; introdução de perecibilidade de stock, que levaria à necessidade de implementação de especificações adicionais no algoritmo, mas que o tornariam mais completo e aplicável a diferentes situações.

É de realçar ainda que outra possível melhoria seria a utilização de um número mais elevado de estados e ações. No entanto, para garantir um treino suficiente da matriz Q nesse caso, seria necessário realizar um elevado número de iterações, de forma a garantir que todos os estados e ações sejam treinados suficientemente. Para isso, seria necessária uma elevada capacidade computacional que neste caso não existia, impossibilitando a utilização de um número muito elevado de estados e ações.

Referências

[1] C. H. Papadimitrioul and J. N. Tsitsiklis, “THE COMPLEXITY OF MARKOV DECISION PROCESSES,” 1985.

[2] R. S. Sutton and A. G. Barto, “Reinforcement learning: An Introduction (2nd edition draft; 2012),” *Learning*, 2012. [Online]. Available: http://incompleteideas.net/book/bookdraft2017nov5.pdf. [Accessed: 04-Feb-2019].

[3] “Diving deeper into Reinforcement Learning with Q-Learning.” [Online]. Available: https://medium.freecodecamp.org/diving-deeper-into-reinforcement-learning-with-q-learning-c18d0db58efe. [Accessed: 04-Feb-2019].

[4] “Reinforcement Learning Demystified: Exploration vs. Exploitation in Multi-armed Bandit setting.” [Online]. Available: https://towardsdatascience.com/reinforcement-learning-demystified-exploration-vs-exploitation-in-multi-armed-bandit-setting-be950d2ee9f6. [Accessed: 04-Feb-2019].

[5] S. K. Chaharsooghi, J. Heydari, and S. H. Zegordi, “A reinforcement learning model for supply chain ordering management: an application to the beer game,” *Decis. Support Syst.*, vol. 45, no. 4, Nov. 2008.

[6] C. Watkins, “Learning from Delayed Rewards.”, 1989.

[7] Tsitsiklis, “Asynchronous Stochastic Approximation and Q-learning.”, 1994.

[8] Tsitsiklis and Bertsekas, “Neuro-Dynamic Programming.”, 1996.

[9] “Methods and Apparatus for Reinforcement Learning, US Patent.”, 2014.

[10] “What is Inventory Management? - Know the Basics | TradeGecko.” [Online]. Available: https://www.tradegecko.com/learning-center/what-is-inventory-management. [Accessed: 04-Feb-2019].

[11] P. A. Jensen and J. F. Bard, “Operations Research Models and Methods.”, 2002.

[12] P. A. Jensen and J. F. Bard, “Operations Research Models and Methods Inventory Theory.S7 The (R, S) Inventory Policy.”, 2002.

[13] “(Cycle) Service Level Definition - Inventory Optimization Software - Lokad.” [Online]. Available: https://www.lokad.com/service-level-definition. [Accessed: 17-Jun-2019].

[14] “Unit Normal distribution.” [Online]. Available: https://courses.edx.org/asset-v1:MITx+CTL.SC1x_1+2T2015+type@asset+block/UnitNormalTable.pdf [Accessed: 05-May-2019]

[15] G. Zhao and R. Sun, “Application of Multi-agent Reinforcement Learning to Supply Chain Ordering Management.”, 2010.

[16] “Python.” [Online]. Available: https://www.python.org/. [Accessed: 09-Feb-2019].