LEONARDO KANASHIRO FELIZARDO

Exploring the boundaries of Deep Reinforcement Learning in simulated environments: A study on financial trading and lot-sizing

São Paulo
2023

LEONARDO KANASHIRO FELIZARDO

Exploring the boundaries of Deep Reinforcement Learning in simulated environments: A study on financial trading and lot-sizing

Thesis presented to the Polytechnic School of the University of São Paulo to obtain the Title of Doutor em Ciências and also presented to Politecnico di Torino to obtain the title of Doctoral Research degree in Pure and Applied Mathematics.

São Paulo
2023

LEONARDO KANASHIRO FELIZARDO

Exploring the boundaries of Deep Reinforcement Learning in simulated environments: A study on financial trading and lot-sizing

Original Version

Thesis presented to the Polytechnic School of the University of São Paulo to obtain the Title of Doutor em Ciências and also presented to Politecnico di Torino to obtain the title of Doctoral Research degree in Pure and Applied Mathematics.

Concentration Areas:

Electronic Systems

Pure and Applied Mathematics

Supervisors:

Prof. Dr. Emilio Del Moral Hernandez

Prof. Dr. Paolo Brandimarte

São Paulo
2023

Catalogação-na-publicação

Felizardo, Leonardo Kanashiro
Exploring the boundaries of Deep Reinforcement Learning in simulated environments: A study on financial trading and lot-sizing / L. K. Felizardo -- São Paulo, 2023.
154 p.

Tese (Doutorado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Sistemas Eletrônicos.

1.Deep Reinforcement Learning 2.Pesquisa Operacional 3.Sistemas de Negociação Autónomos 4.Approximate Dynamic Programming 5.Sistemas Multiagentes I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Sistemas Eletrônicos II.t.

Dedication

To my beloved parents, whose ceaseless dedication and selflessness paved the way for my success. To the rest of my family, specially my aunt, for their ever-present love and companionship. It is through their cycle of love and encouragement that I have found the fortitude and inspiration to pursue my dreams and achieve my goals.

ACKNOWLEDGMENTS

I am grateful to my family and friends who have supported me throughout this journey and will continue to do so. Without their unwavering support, this work would not have been possible.

I also want to thank the *Escola Politécnica da Universidade de São Paulo* for their help with my technical and administrative questions. Special thanks to my advisors, Prof. Dr. Emilio Del Moral Hernandez and Prof. Dr. Paolo Brandimarte, for guiding me through the research and development processes and helping me grow as a researcher. I also express my gratitude to Professor Dr. Edoardo Fadda for his assistance in the research carried out in Italy.

This research was partially funded by the *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior* (CAPES - Coordination for the Improvement of Higher Education Personnel, Finance Code 001, grant 88882.333380/2019-01), Brazil. The views and opinions expressed in this dissertation are solely those of the authors and do not reflect the official views of the funding organization.

Finally, I would like to thank Alice Schiavinato, Francisco Lima, Catharine Graves, Eder Urbinate, Elia Matsumoto, and all the other colleagues I have collaborated with on research and in classes. Our teamwork allowed us to explore uncharted territory fearlessly.

"It is not knowledge, but the act of learning, not possession but the act of getting there, which grants the greatest enjoyment."

-Carl Friedrich Gauss-

RESUMO

Dado o ambiente complexo e em rápida mudança de hoje, é essencial elaborar metodologias robustas para a tomada de decisões. No domínio dos processos algorítmicos de tomada de decisão, o paradigma de Reinforcement Learning (RL) tem-se afirmado progressivamente como uma metodologia preeminente. Essa abordagem é especialmente proficiente ao lidar com ambientes caracterizados por atributos dinâmicos e não determinísticos. No entanto, é fundamental analisar a adequação de RL para cada aplicação. Nesta tese, utilizamos uma estrutura matemática unificada baseada no controle estocástico que nos ajuda a identificar as principais características de um problema, permitindo a descoberta de métodos mais eficazes para melhor convergência para um espaço de solução. Com esta estrutura matemática, desenvolvemos e descrevemos as duas contribuições significativas feitas nesta tese. Primeiramente, propomos um método de classificação denominado Residual Network Long Short-Term Memory Actor (RSLSTM-A) para resolver o Active Single-Asset Trading Problem (ASATP). Nosso método supervisionado proposto apresentou resultados superiores ao estado da arte dos métodos de RL . Como o ASATP é um tipo de problema onde a matriz de probabilidades de transição não depende das ações do agente, é razoável supor que a Supervised Learning possa ser capaz de alcançar melhores resultados frente ao uso de RL. Além disso, assumindo que nesta instância do problema não enfrentamos um dilema de exploração-aproveitamento (exploration-exploitation), os métodos contextual bandit podem não ser adequados, estabelecendo-se Supervised Learning a melhor abordagem. Na segunda parte dos resultados desta tese, validamos o potencial das técnicas de RL em outra instância do problema, o Stochastic Discrete Lot-Sizing Problem (SDLSP), propondo uma abordagem multiagente que supera as principais técnicas de RL. Além disso, aplicamos estados pós-decisão para construir um método de Approximate Dynamic Programming que pode superar métodos básicos e de Deep Reinforcement Learning em várias configurações de SDLSP.

Palavras-chave – Deep Reinforcement Learning, Pesquisa Operacional, Sistemas de Negociação Autónomos, Approximate Dynamic Programming, Sistemas Multiagentes.

ABSTRACT

Given today's rapidly changing and complex environment, crafting robust methodologies for decision-making is essential. In algorithmic decision-making processes, the Reinforcement Learning (RL) paradigm has progressively asserted itself as a preeminent methodology. This approach is especially proficient when dealing with environments characterized by both dynamic and non-deterministic attributes. However, it is essential to analyze the suitability of RL for each problem application. In this thesis, we use a unified mathematical structure based on stochastic control that helps us identify the main characteristics of a problem, allowing the discovery of more effective methods for better convergence in the solution space. With this mathematical framework, we develop and describe the two significant contributions made in this thesis. Firstly, we propose a classification method named Residual Network Long Short-Term Memory Actor (RSLSTM-A) to solve the Active Single-Asset Trading Problem (ASATP). Our proposed supervised method presented results that are superior to state-of-the-art RL methods. Since the ASATP is a type of problem where the transition probability matrix is not dependent on the agent's actions, it is reasonable to assume that Supervised Learning might achieve better results than RL. Also, assuming that in this problem instance, we do not face an exploration-exploitation dilemma, the contextual bandit methods may need to be revised, and Supervised Learning establishes itself as the best approach. In the second part of the results of this thesis, we validate the potential of RL techniques in another problem instance, the Stochastic Discrete Lot-Sizing Problem (SDLSP), by proposing a multi-agent approach that outperforms the leading RL techniques. Furthermore, we apply post-decision states to build an Approximate Dynamic Programming method that can outperform baseline and Deep Reinforcement Learning methods in various SDLSP settings.

Keywords – Deep Reinforcement learning, Operations Research, Autonomous Trading Systems, Approximate Dynamic Programming, Multi-Agent System.

LIST OF FIGURES

LIST OF TABLES

LIST OF NOTATION SYMBOLS

A_t - Action variable at time t (same as a decision but employed mainly on MDP formulation)

$A(.)$ - Advantage function

A - Exponential moving estimates of the first moment of R_t - Subsection 5.1.1

B - Exponential moving estimates of the second moment of R_t - Subsection 5.1.1

$B(\theta)$ - Entropy bonus given the model parameters θ

$C(.)$ - Contribution function

C_t - Contribution variable

c_t - Contribution variable value

$c_{i,m}$ - Setup loss matrix when changing the setup of machine m to item i

c_1, c_2 - Relative importance coefficients for PPO loss

$\mathbf{D}$ - Set of trajectories used in the PPO algorithm

$D_t \in \mathcal{D}$ - Demand variable at time t

d_t - Demand variable value at time t

$f_{i,m}$ - Setup cost if machine m starts to produce item i.

$f \in \mathcal{F}$ - Approximation function employed in the decision problem mathematical definition, Section 3.1

$F(X, W)$ - State-independent objective function - employed in the decision problem mathematical definition, Section 3.1

G - Return, the accumulated discounted rewards, used in explaining RL methods

$G(S_t, X_t)$ - Stochastic component of the reward, used in explaining the ADP method

h_i - Inventory cost vector of item i

h_t - Hidden feature at time t in the RSLSTM-A architecture

$H(X^\pi(S_t))$ - Entropy of the policy

I - Maximum number if items

$I_{i,t}$ - Inventory of item i at time t

$\mathbf{I}_t$ - Inventory of all items at time t

$[l]$ - Set of items

J - Objective function

K_t - Scaling factor based on the number of time periods t used in the calculation of

the Sharpe Ratio

l_i - Lost sales cost of item i

L - Loss value calculates using the loss function

$\bar{L}_i$ - Estimated lost sales cost for item i

M - Number of machines used in the SDLSP

M - Past window time steps used in the ASAPT

M^f - Future window time steps used in the ASAPT

$\mathbf{M}_t$ - Setup of all machines at time t

$\mathbf{M}^h$ - Replay memory vector

[M] - Set of machines

$n \in \mathcal{N}$ - Node n in the set of nodes in the scenario tree $\mathcal{N}$

$n^x_{i,t}$ - Number of machines producing item i at time t after the decision x is made

$n \in \{1, 2, ...N\}$ - Iterations limited to the budget N - employed in the final reward formulation in Section 3.1

N - Number of training episodes

$O(.)$ - Order of the function

$P_t \in \mathcal{P}$ - Asset price variable at time t

p_t - Asset price variable value at time t

$p(n)$ - Parent node

$p_{i,m}$ - Number of items i produced by the machine m

$P(X)$ - Probability of an action in the RL context

$P(S_{t+1}|S_t, A_t)$ - Transition function - MDP formulation

q - Alternative notation for probability

$Q(.)$ - Q-function

$R_t \in \mathcal{R}$ - Return variable at time t

r_t - Return variable value at time t

r^f - Risk-free daily rate

$S_t \in \mathcal{S}$ - State variable at time t

s_t - State variable value at time t

S^M - Transition function

S^R - Physical state variable

S^I - Information state variable

S^B - Belief state variable

T - Number of time steps

T_D - Number of trading days in a period

$T(S, A)$ - One step transition function - MDP formulation

W_t - Exogenous information at time t

$V(.)$ - Value function

x_t - Decision variable value at time t

$x_{i,m,t}$ - Binary variable equal to 1 if machine m is producing item i at time t

X^π - Policy function with π carrying the information about the type of function f

$X_t \in \mathcal{X}$ - Decision variable at time t

$\mathbf{X}^h$ - History of policy decisions

$z_{i,t}$ - Lost sales of item i at time t

Z - Sample batch

α - Learning rate

γ - Discount factor

$\delta_{i,m,t}$ - Binary variable equal to 1 if machine m has done a setup between time $t-1$ and time t

δ - Transaction cost or commission employed in the ASATP

ϵ - The $\epsilon - greedy$ parameter - employed in the DQN methods description

ϵ - The clipping range parameter employed the PPO description

ε - Minimal number requisite for logical constraints.

$\theta \in \Theta^f$ - Tunable parameters

Θ - All tunable parameters set

λ - Regularization of the advantage

μ - Quantity of available assets for trading

π - Information about function f that approximates the policy X^π

$\pi^{[n]}$ - Unconditional probability of node n ($\pi^{[0]} = 1$)

ρ - Ratio calculates the probability of choosing a particular action

$\phi \in \Phi^f$ - Tunable parameters alternative notation

Φ - All tunable parameters set alternative notation

$\mathcal{C} \subseteq [M] \times [I]$ - Set of initial conditions. The couples $(m, i) \in \mathcal{C}$ if machine m is producing item i.

$\mathcal{D}$ - Demand space

$\mathcal{F}$ - Set of approximation functions

$\mathcal{F}(x)$ - Residual mapping

$\mathcal{H}(x)$ - Underlying mapping

$\mathcal{I}(m)$ - Set of items that can be produced by machine m

$\mathcal{N}$ - The set of nodes in the scenario tree

$\mathcal{M}(i)$ - Set of machines that can produce $i \in \mathcal{I}$

$\mathcal{P}$ - Set of production plan

$\mathcal{P}$ - Price variable space used in the ASAPT

$\mathcal{R}$ - Return space

$\mathcal{S}$ - State space

$\mathcal{T}$ - Set of time periods.

$\mathcal{X}$ - Decision space

$\hat{Y}$ - The operator "hats" denotes exogenous variable t

clip - Superscript notation for clipped surrogate objective

eval - Sperscript notation for evaluation function

b - Baseline model or bootstrapped model indicator, y_b

h - Superscript notation for storage vector of variables values

$i \in 0, 1, 2, ..., N$ - Iteration index or item index, y_i

$k \in 0, 1, 2, ..., \infty$ - Index for future time-steps in the future starting from t

$m \in \mathcal{M}$ - Machine index

N - Max number of iterations

$*$ - Superscript notation for optimal decision X^*

$+$ - Superscript notation for after demand in scenario tree, $\mathcal{N}^+$

$+$ - Superscript notation for the RL decision to select which baseline decision

$t \in 0, 1, 2, ..., T$ - Time index, y_t

T - Max value of the time index t

T - Superscript notation trader return R^T

$[n]$ - Operator to indicate to which note the variable value is associated, $y^{[n]}$

x - Superscript notation for post-decision state, value, and transition function

LIST OF ACRONYMS

A2C - Advantage Actor-Critic

ACR - Accumulated Agent Asset Price Return

ADP - Approximate Dynamic Programming

ANN - Artificial Neural Network

API - Application Programming Interface

AR - Annualized Return

ASATP - Active Single-Asset Trading Problem

B&H - Buy and Hold

BQN - Branching Dueling Q-Network

BTS - Bootstrapped Thompson Sampling

BTC - Bitcoin

DASH - Dash

DLSP - Discrete Lot-Sizing Problem

DP - Dynamic Programming

DR - Decision Rule

DRL - Deep Reinforcement Learning

DQN - Deep Q-Network

DDQN - Double Deep Q-Network

ETH - Ethereum

ETC - Ethereum Classic

ISP - Identical Parallel Sources

LSCMA - Lot-Sizing Cooperative Multi-Agent Adjustment

LSMC - Least-Squares Monte Carlo

LSTM - Long Short-Term Memory

LTC - Litecoin

LSK - Lisk

MS - Multi-Stage

NXT - Next

PG - Policy Gradient

PI - Perfect Information

PO - Policy Optimization

PPO - Proximal Policy Optimization

ReLU - Rectified Linear Unit

RF - Random Forest

RL - Reinforcement Learning

RRL - Recurrent Reinforcement Learning

ResNet - Residual Network

RSLSTM-A - Residual Network Long Short-Term Memory Actor

SDLSP - Stochastic Discrete Lot-Sizing Problem

SL - Supervised Learning

SR - Sharpe Ratio

URS - Unrelated Parallel Sources

XEM - New Economy Movement

XMR - Monero

XRP - Ripple

CONTENTS

1 INTRODUCTION

Daily problems are often defined by the sequential nature of decision-making, in which a series of choices must be made to achieve a desired outcome. For decades, artificial intelligence experts have sought to model and resolve such issues by emulating nature's learning algorithms, such as Reinforcement Learning (RL). This method leverages trial and error to inculcate autonomous agents' behaviors (or policies) as they interact with their environment, deriving rewards and incentives. Research in this field has been ongoing since the 1950s in computer science. A remarkable milestone was reached in the field of RL with the incorporation of deep learning, resulting in the advent of a new subfield called Deep Reinforcement Learning (DRL) (MNIH et al., 2015). Using function approximation methods, such as Artificial Neural Networks (ANNs), enables us to learn complex and high-dimensional state space representations, overcoming problems such as the "curse of dimensionality" in RL. Furthermore, deep learning demonstrates remarkable proficiency in approximating functions, a critical task in approximating value and policy functions. These challenges, only partially alleviated by deep learning, have been long-standing issues in the research communities of Dynamic Programming (DP) and stochastic optimal control. While there are intriguing methods for tackling DP problems, such as the curse of dimensionality (POWELL, 2007), DRL remains an affluent area of research for addressing a diverse range of such problems. The research community has, therefore, begun to develop solutions for previously intractable problems using DRL methods. Some of the most notable achievements of DRL include its exceptional performance in games such as Go (SILVER et al., 2016) and tabletop games (SILVER et al., 2018), as well as in MuJoCO physics problems (DUAN et al., 2016).

Traditionally, the ability to outperform human players in games has served as a benchmark for computer science experiments and has often been a formidable challenge. As we observe with increasing frequency (MNIH et al., 2013; MNIH et al., 2015; SILVER et al., 2016; SILVA; COSTA, 2019), DRL methods attain performance levels in various tasks that exceed human capability. Due to this, the research community has redirected its focus to addressing other real-world problems. DRL has gained widespread use across a variety

of industries, including operations research (BOUTE et al., 2021; GIOIA; FELIZARDO; BRANDIMARTE, 2022), finance (PAIVA et al., 2022; FELIZARDO et al., 2022a; FELIZARDO; MATSUMOTO; DEL-MORAL-HERNANDEZ, 2022b), autonomous vehicles (KIRAN et al., 2022), healthcare (YU et al., 2021), and natural language processing (BAI et al., 2022), to name a few. Real-world problems can pose unique difficulties for the implementation of DRL, as they often have more significant complexities that require specific adaptations of DRL methods. It is important to note that the benchmark for formidable performance in real-world problems is not limited to human ability but can also encompass the capabilities of autonomous systems that are specifically designed to solve a particular problem. As a result, DRL must meet higher performance standards and require adaptations to meet the specific requirements of each problem. Despite these challenges, there has been a significant increase in research exploring applying DRL techniques to real-world problems (SARKER, 2021). Various solutions have been proposed to overcome these challenges, such as the use of multi-agent systems (NGUYEN; NGUYEN; NAHAVANDI, 2020), transfer-learning (SILVA; COSTA, 2019), the integration of conventional techniques with RL (PARBHOO et al., 2017), and the development of specialized frameworks (POWELL, 2021). This work expands the contributions to real-world problems by proposing DRL solutions and related alternatives to two real-world applications.

To conduct a progressive investigation, we begin by exploring the Active Single-Asset Trading Problem (ASATP), which is comparatively simpler due to its inherent characteristics that we will elaborate on. Although recent literature (DENG et al., 2017; PARK; SIM; CHOI, 2020; ALMAHDI; YANG, 2019; ABOUSSALAH; LEE, 2020; CARTA et al., 2021) has tackled this similar problem using various DRL methods, we propose that these approaches might not be ideally suited for this specific problem, despite their seemly capacity to address it. Next, we delve into a more complex problem (given the problem definitions we adopted) in the domain of operations research—the Stochastic Discrete Lot-Sizing Problem (SDLSP). Here, we bring some pieces of evidence of why we believe RL could be a viable and advantageous alternative for directly solving this problem. Through the exploration of the characteristics of the SDLSP, including configurations with higher dimensionality (which we will further explain), we highlight scenarios where RL proves to be a fitting solution. Our investigation seeks to shed light on the suitability of RL in these distinct problem instances and aims to provide valuable pieces of evidence for the practical application of RL techniques (or not) in real-world decision-making challenges.

Active trading is a decision problem where an agent takes various positions to maximize profit, and it can be addressed using multiple methods, including DRL. Recent

studies, such as Park, Sim and Choi (2020) and Almahdi and Yang (2019), have applied DRL techniques that were built to solve full Reinforcement Learning problems (SUTTON; BARTO, 2018, Chapter 2). However, upon analysis, it was found that most of these works are contextual bandit problems, as defined in Sutton and Barto (2018, Chapter 2.9) and discussed in our previous work (FELIZARDO et al., 2022a). The contextual bandit problem is a variant of the Reinforcement Learning problem that assumes that the agent's decisions have no impact on future state variables related to the reward. In other words, the transition function provides a probability distribution over the next possible states given only the current state variables and not the agent's decision. One important characteristic of active trading is that the best possible action can be determined if the next price is known. This phenomenon occurs because the decision-making process is based on predicting future price movements and taking appropriate positions. As a result, the trading task can be reduced to a time series classification problem, where the agent's goal is to predict the next price movement based on historical data and choose the appropriate action from a limited set of discrete options. The presence of transaction costs is the main factor that would justify using a more complex policy finder such as DRL. However, as we present in this work, a simple heuristic can deal with the complexity introduced by the transaction cost. We show the RL results in a scenario with transaction costs and compare them against our proposed approach, the Residual Network Long Short-Term Memory Actor (RSLSTM-A) (FELIZARDO et al., 2022a). We outperformed near-state-of-the-art DRL methods by using state-of-the-art time series classification and a simple heuristic.

In addition to active trading, we have investigated Reinforcement Learning and related methods in operations research. Traditionally, many methods used in operations research have come from the stochastic optimal control community, with DP methods often used to solve stochastic optimal control problems. RL and DP share a common theoretical background based on Hamilton-Jacobi-Bellman equations, but both communities have different mathematical frameworks, as we further describe. The mathematical framework employed in the DP uses control theory that provides several techniques to solve inventory management problems, such as the Discrete Lot-Sizing Problem (DLSP), In the SDLSP, DP and Mixed Integer Programming (MIP), among other techniques in stochastic optimal control, are commonly employed. Motivated by the recent achievements in the RL computer science field, RL methods are gaining prominence in operations research. In recent years, many works have been employing DRL methods (WANG; LI; ZHU, 2012; RUMMUKAINEN; NURMINEN, 2019; LI et al., 2020; GIJSBRECHTS et al., 2022) that

deal with the SDLSP specifically. One main advantage of the recently developed DRL methods is the capacity to find good policies only by interacting with an environment simulation. RL is more flexible regarding the problem constraints and may not require prior knowledge of the problem or any handmade heuristic. Another interesting advantage of DRL in the SDLSP is the capacity to approximate functions employing ANNs. The use of ANNs as approximation methods in Hamilton-Jacobi-Bellman-based techniques is not exclusive to the RL community but is also present in the field of stochastic optimal control as Approximate DP (ADP).

In the SDLSP, many problem constraints can be changed to justify using a more general solution. Examples include using a complex production model with a sequence-independent matrix of setup costs and machine production under different time-horizon regimes. On the other hand, most of the literature explores solutions under stochastic optimal control, which can be efficient depending on the problem configuration. We aim to give, as a contribution, a more comprehensive comparison between stochastic optimal control techniques, DP (in this case ADP), and DRL. Here, the contribution is two-fold. Firstly, we propose an Approximate Dynamic Programming technique that employs a post-decision value function using a branch and bound tree-like heuristic. Secondly, we propose a Lot-Sizing Cooperative Multi-Agent Adjustment (LSCMA) approach that uses a baseline policy to generate a better policy. Our approach leverages the advantages of multi-agent systems and Deep Reinforcement Learning to find a better policy than a baseline.

In both real-world applications, we have adopted a uniform notation drawn from stochastic optimal control. To unify the notation and enhance the mathematical characterization of the problems, we adopt the mathematical framework suggested by Powell (2021). This framework enables a comprehensive understanding of the elements used by both research communities (stochastic optimal control and Reinforcement Learning) while providing sufficient mathematical detail to model the problems effectively.

In the active trading problem, we make the following contributions, building on previous results obtained in Felizardo et al. (2022a):

1. The proposal of RSLSTM-A, a tailored heuristic with a Supervised Learning approach for the trading task that outperforms other methods and the Buy and Hold (B&H) strategy.

2. Evidence that the selected Supervised Learning technique provides stable and competitive performance compared to the most recent RL approaches in the literature.

3. Insightful interpretations of the Residual Neural Network learning process for a time series sequential decision process through visual representations of feature extraction and comparing the original time series and channel outputs.

In the SDLSP, we present the following contributions:

1. An environment model of inventory management simulation for RL application that can support future research, benefit research communities, and advance computer science and operations research.
2. A priority-based decision-making technique.
3. A branch and bound tree heuristic for Approximate Dynamic Programming.
4. The integration of baseline techniques with RL using our custom-made Lot-Sizing Cooperative Multi-Agent Adjustment (LSCMA) proposal.
5. A comparison and analysis of these techniques, exploring their advantages, and a comparison against multistage stochastic programming at different complexity levels.

To assist the reader in navigating this thesis, we present an outline of its structure, accompanied by succinct summaries of each chapter's content:

- **Literature review**

  This chapter is divided into three main sections:

  Dynamic Programming and Reinforcement Learning: This section provides an overview of the fundamental concepts of DP and RL. It discusses the principles of these methods and their applications in various fields. It also delves into some of these techniques' mathematical aspects, which are further explored in the thesis.

  RL in Active Asset Trading Problems: This section focuses on applying Reinforcement Learning in active asset trading problems. It discusses the challenges and complexities associated with these problems and how Reinforcement Learning can be used to address them. It also provides a detailed analysis of the existing literature on this topic, highlighting the strengths and weaknesses of different approaches.

  Reinforcement Learning in Stochastic Discrete Lot-Sizing Problem: The final section of the literature review chapter is dedicated to applying Reinforcement Learning in

Stochastic Discrete Lot-Sizing Problems. It provides a comprehensive review of the existing literature on this topic and discusses the potential of Reinforcement Learning in addressing these problems. It also highlights the gaps in the current research and justifies the research conducted in this thesis.

These sections set the stage for the subsequent chapters by highlighting the gaps in the current research and establishing the need for the research conducted in the thesis.

- **Theoretical background**

  The chapter begins by discussing the limitations associated with the Markov Decision Process (MDP) mathematical framework. It then discusses the potential benefits of the mathematical framework used in your study, highlighting its advantages over the MDP. The chapter also details the decision process for the two problems being examined.

  The chapter notes that the mathematical framework suggested by Powell (2021) supports explaining all employed methods in your work. This framework primarily employs stochastic optimal control notations, and the chapter draws parallels between the notations used in Sutton (1988) and Sutton and Barto (2018) and Powell (2021). The chapter also models the problem using the mathematical framework suggested by Powell (2021) and describes the solutions within this framework. This approach ensures that the solutions are not specific to a particular mathematical framework but are broadly applicable and can be compared with solutions from different communities.

  The chapter then delves into the specifics of the two problems being examined in your thesis, namely the Active Single-Asset Trading Problem and the Stochastic Discrete Lot-Sizing Problem. Each problem is discussed in detail, focusing on their mathematical descriptions and the decision problem mathematical framework for each problem.

  The chapter also provides a detailed explanation of the state variables used in the mathematical framework. These state variables are divided into three categories: physical state, information state, and belief state. The physical state refers to the number of assets the agent owns and whether the agent is long or short in that asset. The information state refers to any deterministic information that can evolve exogenously or be controlled by decisions, such as price changes. The belief state refers to the information about the distributional information of the unknown

parameters, such as the mean and the covariance matrix of a multivariate normal distribution.

- **Reinforcement Learning Methods**

  The chapter is divided into several sections, each focusing on a different Reinforcement Learning method. Here are the details:

  Bootstrapped Thompson Sampling (BTS): This section discusses the BTS method. It is a Reinforcement Learning method that uses a form of uncertainty estimation to guide the exploration process. This method is particularly useful when the agent needs to balance the trade-off between exploration and exploitation.

  Deep Q-Network (DQN): This section delves into the DQN method. DQN is a value-based Reinforcement Learning algorithm that uses a neural network as a function approximator to estimate the Q-value function. The Q-value function determines the quality of actions given a particular state.

  Actor-critic: This section explores the actor-critic method. In this method, two neural networks are used: one (the actor) is used to select actions, and the other (the critic) is used to evaluate the selected action. The critic's feedback is then used to update the actor's policy.

  Advantage Actor-Critic (A2C): This section focuses on the advantage A2C method. A2C is an actor-critic method that uses the advantage function concept to reduce the gradient estimate's variance. The advantage function measures how much better an action is compared to the average action for a particular state.

  Proximal Policy Optimization (PPO): The final section discusses the PPO method. PPO is a Policy Gradient method that uses a surrogate objective function to improve the stability of the learning process. It is designed to address the challenges of other Policy Gradient methods, such as the difficulty of choosing a suitable step size.

  Each section provides a detailed explanation of the respective method, including its theoretical underpinnings, advantages and disadvantages, and applications in various domains. The chapter also includes comparisons between the methods, highlighting their differences and similarities.

- **Active Single-Asset Trading Problem**

The chapter introduces two distinct methods explicitly crafted to tackle the ASATP. The first method is the Recurrent Reinforcement Learning (RRL) methodology, which was first introduced by Moody and Wu (1997) and employed as an RL method purposed to solve trading problems within the scope of portfolio management. This method has since been adapted for single asset trading.

The second method is the Residual Network Long Short-Term Memory Actor (RSLSTM-A) (FELIZARDO et al., 2022a), primarily utilizing a Supervised Learning approach to address the decision problem. This method employs a modified version of the Residual Network (ResNet) architecture, functioning as a time series classifier for decision problems. This exploration into the unique features of the ResNet model enables us to illustrate how our approach could surpass traditional RL techniques in solving active trading problems.

The chapter then presents the experimental results of the research, providing a brief overview of the experimental setups and analyzing the pertinent market data and performance metrics. The performance of the proposed RSLSTM-A method is revealed, investigating two fundamental problem setups: those inclusive and exclusive of transaction costs. The critical influence of transaction costs on policy formulation is underscored, highlighting its significant role within this context. The investigation extends beyond the mere exploration of the proposed RSLSTM-A algorithm. It encompasses a comparative analysis of the current milieu of solutions available for the ASATP, including both RL algorithms and the contextual bandit solution BTS. The intention is to comprehensively review existing techniques, emphasizing those already implemented within trading scenarios. As we explore these various methodologies, our focus remains steadfastly on RL algorithms that have been comprehensively tested within the trading context, thus ensuring an accurate and enlightening analysis.

The chapter also includes an appendix with auxiliary material related to the ASATP, including environment setting tables.

- **Stochastic Discrete Lot-Sizing Problem**

  The chapter delves into three methods to address the Stochastic Discrete Lot-Sizing Problem (SDLSP). These methods include the Branch and Bound Approximate Dynamic Programming (BBADP) method, the Decision Rule (DR) method, and the Lot-Sizing Cooperative Multi-Agent Adjustment (LSCMA) method. Each method contributes a unique approach to the problem, promising distinct advantages and

opportunities. The BBADP method is an approach rooted in Dynamic Programming to calculate the optimal solution, assuming that the demand distribution is known. The DR method uses eligibility and priority rules to guide the decision-making process. The LSCMA method relies on a cooperative multi-agent approach to improve the recommendations from a baseline agent using multiple RL agents.

The chapter presents the experimental results of employing these methods and other RL solutions. A comprehensive analysis of these results sheds light on their performance under various conditions, offering insights into their potential real-world applications. The aim is to understand the landscape of solutions available for the SDLSP, identifying the most effective solutions within the context of this domain.

The chapter presents the experimental results of employing these methods and other RL solutions and mainstream methods such as rule-based methods. A comprehensive analysis of these results sheds light on their performance under various conditions, offering insights into their potential real-world applications. The analysis includes a comparative study where our proposed methods are benchmarked against the RL and other mainstream methods. This comparison allows us to evaluate the effectiveness of our methods concerning established techniques in the field.

The chapter includes an appendix with auxiliary material related to the SDLSP, including environment setting tables.

- **Conclusion**

  The concluding chapter of the thesis revisits the contributions made throughout the research. It emphasizes the effectiveness of the proposed RSLSTM-A in solving the ASATP, outperforming state-of-the-art Reinforcement Learning methods. It also highlights the successful application of Reinforcement Learning techniques to the SDLSP, where multi-agent and ADP methods were proposed and validated. The chapter concludes with a discussion of potential future research directions, including exploring other Reinforcement Learning methods and applying the proposed methods to other decision-making problems.

- **Appendix A, B, C**

  The first two appendices of the thesis provide details on the methods implemented to address the ASATP and the SDLSP, as outlined in Appendix B and A, respectively. These sections serve as complementary material to help the reproducibility of the experiments conducted in the study, giving some tools to validate and build upon

this work in future research. The Appendix C provides the list of published academic papers during the Ph.D. program with the abstract of each paper.

In this thesis, it is important to note that the list of acronyms and symbols is positioned before the introduction chapter.

2 LITERATURE REVIEW

This literature review is structured into three parts. The first part gives an overview of recent advances in RL and DP and a brief historical account of their evolution to their current states. The second section discusses recent research that has utilized RL techniques to perform active trading. In the last section, we provide a comprehensive review of solutions for the SDLSP, with a specific focus on recent applications of RL.

There has been a surge in the use of RL, particularly DRL, in the two problem domains mentioned. In light of this trend, it is important to consider the prerequisites for successfully implementing RL methods. Despite the growing interest in RL, its application appears less prevalent in the SDLSP due to the long-standing use of traditional DP methods in this field. On the other hand, DRL has been frequently applied to active trading. However, the comparison with standard methods is often lacking. Hence, there is a pressing need for further research in both domains to realize the capabilities of RL and DRL fully. In this review, we aim to shed light on the potential of RL in these problem domains.

2.1 Dynamic Programming and Reinforcement Learning

In the field of optimal control, the solution to deterministic control problems dates back to the early 1900s[1]. However, as decision problems incorporating stochastic variables emerged, stochastic optimal control became more relevant in the 1950s, and new methods were developed to solve them. One such method was DP (BELLMAN, 1957), which could solve single and multi-stage decision problems. DP utilizes the principle of optimality, breaking down a complex problem into simpler subproblems, allowing for the efficient calculation of optimal policies that are a sequence of optimal decisions independent of

[1] The literature on deterministic control is rich, see Kirk (2004), Lewis, Vrabie and Syrmos (2012), Stengel (1994)

the initial state. The Hamilton-Jacobi-Bellman equation (see Lewis, Vrabie and Syrmos (2012) for more details) offers a general solution to nonlinear optimal control problems, which are often intractable to solve analytically. Compared to other techniques, such as direct enumeration, DP significantly reduces computational complexity. Specifically, the number of calculations required by direct enumeration increases exponentially with the number of stages of the decision process, while the computational requirements of DP increase linearly. As the number of stages increases, DP becomes increasingly more efficient than direct enumeration. In terms of computational complexity, DP is generally considered to be $O(n^2)$ or $O(n^3)$, while a direct enumeration is $O(n^n)$ where n is the number of stages.

Dynamic Programming is a powerful technique for solving multi-stage decision problems, but its effectiveness can be hindered by the curse of dimensionality (BELLMAN, 1957). The curse of dimensionality refers to the fact that as the number of dimensions in a system increases, the amount of computer memory required to store the values of the value function becomes impractical. For example, in a robotic arm with ten joints, each with a range of motion of 180 degrees and a discretization of 1 degree, the number of possible states is $180^{10} = 2.7 \times 10^{14}$, requiring an unfeasible amount of storage to store the value function. Other methods, such as state aggregation and Approximate/Adaptive Dynamic Programming (ADP) (WERBOS, 1977; WERBOS, 2007), have been proposed as alternatives to address this limitation. As another alternative to overcome the curse of dimensionality, Powell (2007) presents a model-based[2] method that can deal with high dimensional problems using ADP with radial-based function and post-decisions states. Techniques such as the Taylor series and Artificial Neural Networks can approximate the policy and value functions using the Bellman equation. With the large availability of computational resources, researchers have been exploring the use of function approximators in recent RL methods (MNIH et al., 2013; MNIH et al., 2015), which have shown promising results.

There has been a surge in the use of RL, particularly DRL, in the two problem domains mentioned. In light of this trend, it is important to consider the prerequisites for successfully implementing RL methods. Despite the growing interest in RL, its application appears less prevalent in the SDLSP due to the long-standing use of traditional DP methods in this field. On the other hand, DRL has been frequently applied to active trading. However, the comparison with standard methods is often lacking. Hence, there

[2] Model-based is the term used to describe the techniques employed to solve decision processes using the transition function

is a pressing need for further research in both problem fields to realize the capabilities of RL and DRL fully. In this review, we aim to shed light on the potential of RL in these problem domains.

The field of RL has evolved, incorporating a blend of concepts and techniques. The RL community has embraced techniques such as temporal-difference learning (SAMUEL, 1959), value functions (HOLLAND, 1976), and the combination of optimal control with Q-learning (WATKINS, 1989). Additionally, RL has combined elements of Dynamic Programming and trial-and-error learning (WERBOS, 1977; WERBOS, 1987). One significant advancement in RL was the successful application of temporal-difference learning and ANNs as approximation functions for value functions in backgammon games (TESAURO, 1992; TESAURO, 1995; TESAURO; GALPERIN, 1996). ANNs have since been used as more advanced approximation methods to model complex state features (MNIH et al., 2013; MNIH et al., 2015; MNIH et al., 2016), often in the context of playing games, which has long been a benchmark in the RL community starting with Samuel (1959) and continuing with more recent works such as Schulman et al. (2017). Meanwhile, the stochastic optimal control community has focused on real-world applications, employing techniques that share the same roots.

In recent years, RL has emerged as a powerful tool in real-world applications, enabling significant advances in diverse areas such as operations research (RUMMUKAINEN; NURMINEN, 2019; GIJSBRECHTS et al., 2022), molecular optimization (ZHOU et al., 2019), autonomous vehicles (KIRAN et al., 2022), and robotics (ZHAO; QUERALTA; WESTERLUND, 2020). This literature review focuses on two specific real-world types of applications of RL and ADP: active asset trading and lot-sizing problems.

2.2 Reinforcement Learning in active asset trading problems

This section of the literature review aims to provide an in-depth analysis of the recent advancements in the utilization of RL in trading systems. We begin by briefly reviewing technical analysis methods and then explore the progress made in the field of RL as an autonomous active asset trading method.

The practice of technical analysis in finance involves using quantitative and statistical methods to identify patterns and trends in asset price movements. This approach has a long history, dating back to the late 1800s, with techniques attributed to Charles Dow, as

discussed in Hamilton (1922). However, the efficient market hypothesis, first proposed by Fama (1965), suggests that financial markets follow a random walk, making it impossible to predict future asset prices. Despite the efficient market hypothesis, several studies have challenged it by demonstrating that market inefficiencies can occur. For example, Fama and French (1988) found evidence of a small firm effect and a value effect inconsistent with the random walk assumption. Chopra, Lakonishok and Ritter (1992) shows that the returns on stocks with low price-to-earnings ratios and high book-to-market values are inconsistent with the efficient market hypothesis. Furthermore, Brock, Lakonishok and LeBaron (1992) investigates popular trading rules based on moving averages and trading range breaks and finds returns incompatible with the random walk assumption. Similarly, Bessembinder and Chan (1995) employed a range of popular trading rules, including trading range break and fixed and variable-length moving averages, reaching conclusions contradicting the efficient market hypothesis.

In recent years, there has been a growing interest in utilizing machine learning methods to discover trading rules in the financial market. Machine learning is a subset of artificial intelligence that involves developing algorithms and statistical models that allow computer systems to learn from and make predictions or decisions based on data without being explicitly programmed. This approach has gained considerable interest in recent years, particularly in the financial industry, due to the vast amount of available data and the potential for machine learning to identify complex patterns and relationships that may only be apparent to human analysts. Traditionally, mathematical techniques such as moving averages-based methods Geurts, Box and Jenkins (1977) have been used to identify trends and patterns in the financial market. However, these techniques may only be effective in some situations, failing as the market becomes complex and the number of factors influencing asset prices increases. Machine learning methods can be trained on large amounts of historical market data to identify patterns in these complex combinations of factors, find relationships, and develop trading rules that may be more effective for asset allocation.

While machine learning is frequently used to estimate future asset prices, another vital element is to discover trading rules that can operate even with the uncertainty of prices. Trippi and DeSieno (1992), for instance, developed a neural network system that could assume *long* or *short* positions based on the asset's market price, beating random trading. Other works, such as Allen and Karjalainen (1999), have used machine learning to extract trading rules without explicitly describing them, similarly to Brock, Lakonishok and LeBaron (1992). Using a genetic algorithm to determine trading policies

did not beat the Buy and Hold (B&H) strategy when transaction costs and out-of-sample data with daily frequency were accounted for. As a subsequent stage, Kuo, Chen and Hwang (2001) suggested a genetic algorithm-based fuzzy neural network that employs a genetic algorithm for initial weights selection and fuzzy logic to capture qualitative characteristics of the stock market for buy and sell choices. In recent years, articles such as Nakano, Takahashi and Takahashi (2018) and Sang and Pierro (2019) employed Supervised Learning approaches to train an ANN to trade on the financial market.

In portfolio optimization, early attempts to apply RL were made by Neuneier (1995) and Moody and Wu (1997). Neuneier used the Q-learning technique to allocate assets at each time step to optimize portfolio performance, while Moody proposed using direct recurrent reinforcements to approximate an optimal asset allocation policy. These two approaches can be categorized as critic-based and actor-based, respectively. Critic-based methods indirectly estimate the optimal policy by estimating each state's value, while actor-based methods estimate the optimal policy directly using only the objective function (we provide more details in Chapter 4). Since then, researchers have continued to improve these methods by introducing new techniques. For example, Maringer and Ramtohul (2010) enhanced the Recurrent Reinforcement Learning strategy by introducing a regime-switching mechanism to better adapt to the various regimes that a price time series may assume. Recently, RL methods such as Asynchronous Advantage Actor-Critic (A3C) and Proximal Policy Optimization (PPO) (KANG; ZHOU; KANG, 2018; YU et al., 2019; LI; ZHENG; ZHENG, 2019; PONOMAREV; OSELEDETS; CICHOCKI, 2019; YE et al., 2020; PAIVA et al., 2022; HIRCHOUA; OUHBI; FRIKH, 2021), have been more used, specially because they empower the advantages of ANNs. Recent studies have explored combining Deep Q-Networks (DQN) methods with recurrent neural networks for asset trading. This approach has been employed in studies such as Deng et al. (2017), Wu et al. (2020), Lei et al. (2020). In some cases, improved results have been achieved by utilizing enhanced feature extraction and information fusion techniques based on multiple asset price histories, as shown in Lei et al. (2020). For a more comprehensive overview of the literature, we recommend consulting the works of Dang (2020), An, Sun and Wang (2022), Felizardo et al. (2022c).

One ongoing debate in the literature is which approach is best for active asset trading. Pendharkar and Cusatis (2018) classified the problem of active trading in portfolio optimization as a multi-armed bandit problem, assuming that asset trades do not impact market prices. Under this premise, a simple rule can be established to create a label, making the problem a Supervised Learning. RL becomes more relevant to trade financial

assets when considering introducing transaction costs. While the use of Reinforcement and Supervised Learning in active asset trading has been explored in the literature, there has been limited comparison between the two approaches with different transaction costs. Only a few studies, such as Deng et al. (2017), Almahdi and Yang (2017), Sun et al. (2022), have briefly compared the effectiveness of Reinforcement and Supervised Learning methods in these different contexts. Whether Reinforcement Learning algorithms perform better in both settings - with and without transaction costs remains unclear. To address this gap, we propose to compare recent DRL approaches and our state-of-the-art Supervised Learning method with a simple heuristic in the Active Single-Asset Trading Problem (ASATP). In the past two years, RL approaches have typically leveraged effective feature acquisition techniques (BORRAGEIRO; FIROOZYE; BARUCCA, 2022). These methods have been increasingly adapted to the specific characteristics of the problem, such as the implementation of branching dueling Q-Networks (TAVAKOLI; PARDO; KORMUSHEV, 2017). Still, working better on the features given by the price time-series, (MILLEA, 2023) employs a multi-agent approach to account for the heterogeneous behavior of different periods.

We conclude this part by presenting Table 1, which provides an overview of the main works from 2017 to the beginning of 2023 that employed RL techniques to solve the ASATP. The table classifies the works based on the type of asset traded, the compared techniques/benchmarks, and the proposed technique. Table 1 summarizes the state-of-the-art RL techniques used to solve the ASATP and positions our proposed RSLSTM-A (SP) method within the existing literature. The table highlights that DQN (MNIH et al., 2013) is a commonly used technique for trading various types of assets, followed by Double Deep Q-Network (DDQN) (HASSELT; GUEZ; SILVER, 2016), Asynchronous Advantage Actor-Critic (MNIH et al., 2016), Policy Gradient (PG) (SUTTON et al., 1999), and PPO (SCHULMAN et al., 2017), which have also been employed in a few works. Although the B&H benchmark is popular for evaluating trading strategies, only some works use it. Additionally, only a few works compare their proposed approach against a rule-based or Supervised Learning approach. In Chapter 5, we better explain some of the methods cited in the table 1 that we employ in this work.

Table 1: This table presents the main works employing RL techniques to trade a single asset using price-related features. This table provides the type of asset employed in the experiments and compares techniques against the proposed method.

Work	Asset	Compared techniques/benchmarks	Proposed technique
(SI et al., 2017)	Futures	RRL, B&H	MODRL
(DENG et al., 2017)	Futures, Indexes	DDR variations, SL, B&H	Fuzzy DDR
(CARAPUÇO; NEVES; HORTA, 2018)	Currency	DQN	DQN
(DANTAS; SILVA, 2018)	Stocks	Q-learning variations, B&H	Q-learning
(LI; ZHENG; ZHENG, 2019)	Stocks	A3C, DQN, SL, B&H	SDAE - DQN
(WU et al., 2019)	Stocks	PG variations	PG
(JEONG; KIM, 2019)	Index	DQN variations, B&H	DQN
(PONOMAREV; OSELEDETS; CICHOCKI, 2019)	Index	A3C variations	A3C
(ZARKIAS et al., 2019)	Currency	DDQN, Fuzzy DDR	DDQN
(WU et al., 2020)	Stocks	DQN, PG, Rule based	PG
(LIU et al., 2020)	Futures	DPG variations, Rule based, B&H	DPG
(LI; NI; CHANG, 2020)	Stocks	DQN, Dueling DDQN	DDQN
(LEI et al., 2020)	Stocks	DDR, RRL, SFM, B&H	TFJ-DRL (PG)
(FENGQIAN; CHAO, 2020)	Futures	PG variations	PG
(LEI et al., 2020)	Stocks	RF, B&H	Q-learning
(THÉATE; ERNST, 2021)	Stocks	Rule based, B&H	DQN
(TSANTEKIDIS; PASSALIS; TEFAS, 2021)	Currency	PPO (different transfer learning approaches)	PPO
(HIRCHOUA; OUHBI; FRIKH, 2021)	Stocks, Indexes	DQN	PPO + rule based
(MA et al., 2021)	Stocks	DDQN variations, B&H	DDQN
(CARTA et al., 2021)	Index	Rule based, B&H	DQN
(SHAVANDI; KHEDMATI, 2022)	Currency	Rule based, B&H	Multi-agent DQN
(SUN et al., 2022)	Stocks, T-bound	Rule based, SP, DQN,	BQN
(BORRAGEIRO; FIROOZYE; BARUCCA, 2022)	Cryptocurrency	None	PG
(MILLEA, 2023)	Cryptocurrency	Rule based	Multi-agent PPO
This work/thesis	Cryptocurrency	DQN, A2C, PPO, RRL, BTS, B&H	RSLSTM-A (SP)

2.3 Reinforcement Learning in Stochastic Discrete Lot-Sizing Problem

This section of the literature review aims to provide an overview of the recent advances in applying RL to the Stochastic Discrete Lot-Sizing Problem (SDLSP), which is a problem located in production planning and inventory management. The SDLSP is a complex problem that involves uncertainty in demands and processing times and is traditionally addressed using scenario trees and Mixed-Integer Programming (MIP) methods. However, scenario trees have limitations in capturing uncertainty while maintaining computational tractability, so heuristics are often implemented to achieve solutions in a reasonable time frame, as noted in Beraldi et al. (2005).

Various solutions have been proposed to address high-dimensional problems and constraints in production planning and inventory management. These solutions leverage mathematical optimization methods, such as MIP, and artificial intelligence techniques, as suggested in studies such as Gicquel, Minoux and Dallery (2011), Jans and Degraeve (2008), Powell (2007), Clark and Clark (2000), Mula et al. (2006). Moreover, Bellman-based approaches like RL and DP, along with approximate techniques such as DRL and ADP, have shown potential in addressing the curse of dimensionality and solving intricate

production planning problems, as emphasized in the study by Mula et al. (2006).

The first paper to use RL for the implemented method that solves the SDLSP was Paternina-Arboleda and Das (2005). They proposed a multi-agent RL method that used the proposed method from Das et al. (1999) with an ANN as a value function approximation method for each agent, where each agent chose which item to produce or the to idle setup. Another example of RL usage is Wang, Li and Zhu (2012), which proposed a Q-learning algorithm improved through a heuristic to solve a three-item with a single-machine problem with uncertainty affecting both demands and processing times. The heuristic modified the ϵ-greedy method to optimize the learning of the RL agent. Boute et al. (2021) suggests that further investigation could be done using larger state spaces and a more significant number of deliveries in lot-sizing problems solved by RL, highlighting that these investigations lack extensive stress in larger actions and state spaces in lot-sizing applications. The same work suggests that the ADP approach for larger action spaces could overcome the dimensionality problem and solve the larger action spaces inventory problem.

As explained by Powell (2007), one way to approach the dimensionality problem is to employ the post-decision state-value function, breaking down the large state space of a Markov Decision Process (MDP) into smaller, more manageable subspaces. In this process, the state space is partitioned based on a set of post-decision states, which are the states that can be reached after taking action in a given state. By using a post-state value function to estimate the value of each subspace, the computational complexity of solving the MDP is greatly reduced. The post-state value function is based on Dynamic Programming and can be used with various solution methods, such as linear programming or Reinforcement Learning, to find an optimal policy for the MDP.

DLR methods in the SDLSP have recently been the focus of considerable attention. For instance, a study by Li et al. (2020) aimed to minimize the tardiness of release dates in a single-machine setting. This was achieved by comparing a variety of model-free[3] RL methods, including Q-learning, Sarsa (RUMMERY; NIRANJAN, 1994), Watkins's Q(λ), Sarsa(λ) (TESAURO, 1995), and the Deep Q-Network (DQN). The study results showed that Watkins's Q(λ) outperformed the other methods, but careful algorithm selection is necessary. Furthermore, Gijsbrechts et al. (2022) demonstrated that Asynchronous Advantage Actor-Critic is a viable method for solving classical inventory problems. While the results suggested that Asynchronous Advantage Actor-Critic has the potential for real-world adoption, further research is needed to investigate its performance in non-stationary

[3] For an elaboration on the terminology, refer to Section 3.1.

environments and initial tuning. Some studies have focused on using PPO to solve the SDLSP, such as Rummukainen and Nurminen (2019) and Hezewijk et al. (2022). The first of these two studies found that PPO outperformed previous RL implementations for the SDLSP (PATERNINA-ARBOLEDA; DAS, 2005) but required more training steps and hyperparameter tuning. In contrast, the latter study explored the scalability of PPO for larger problem instances, even compared to industry benchmarks. PPO may perform well in various complex problems without extensive hyperparameter tuning.

This literature review has highlighted the increasing application of DRL in addressing the complex and high-dimensional challenges presented by the SDLSP in production planning and inventory management. While traditional methods such as scenario trees and Mixed-Integer Programming have limitations, using RL and related techniques such as ADP has shown the potential to provide more effective solutions. Moreover, approximation techniques like DRL and ADP can help overcome the curse of dimensionality and provide more efficient solutions. Despite the significant progress made, further research is required to explore larger state spaces and compare the effectiveness of stochastic programming solutions with ADP and DRL solutions in addressing the SDLSP. Such research can facilitate a better understanding of the potential and limitations of DRL and ADP approaches in solving real-world production planning and inventory management challenges.

We conclude this section by summarizing the works employed by DRL in the SDLSP. This summary will be presented in Table 2 to help contextualize our thesis contribution, positioning this work in the literature. Some of the methods presented in Table 2 and in this chapter are explained in detail in Chapter 6.

Table 2: This table is the literature summary of the application of DRL in the SDLSP. We mark what type of machine production replenishment: Single machine use, multiple machines with Identical Parallel Sources (IPS), or Unrelated Parallel Sources (URS). Finally, we mark if the work analyzed deals with multiple items. The magnitude, measured in terms of a production loss, is the maximum value proportional to the time bucket used due to setup change.

Reference	**RL technique**	**Setup/order time**	**Machine/Replenishment**		
		Magnitude	**Single**	**IPS**	**URS**
(PATERNINA-ARBOLEDA; DAS, 2005)	relaxed-SMART	< 0.2	✓		
(WANG; LI; ZHU, 2012)	Q-learning	< 0.3	✓		
(RUMMUKAINEN; NURMINEN, 2019)	PPO (modified)	$0.1, 0.2, 0.15$	✓		
(LI et al., 2020)	Q-learning, DQN, Sarsa		✓		
(GIJSBRECHTS et al., 2022)	A3C,PPO		✓	✓	
(HEZEWIJK et al., 2022)	PPO	$0.3, 0.5$	✓	✓	
This work/thesis	PPO,A2C,BBADP,LSCMA	> 1			✓

3 THEORETICAL BACKGROUND

This chapter begins by outlining some of the limitations of the Markov Decision Process (MDP) mathematical framework. It then discusses the potential benefits of the mathematical framework used in this thesis, highlighting its advantages over the Markov Decision Process. We also detail the decision process mathematical framework for the two problems being examined in this thesis.

In operations research, the mathematical frameworks stem from the stochastic optimal control and Dynamic Programming fields (BELLMAN, 1957; POWELL, 2007; LEWIS; VRABIE; SYRMOS, 2012). The MDP mathematical framework is widely used in Reinforcement Learning literature, as referenced by (PUTERMAN, 1994; SUTTON; BARTO, 2018). However, as noted by Powell (2021), the MDP framework has its own set of challenges, which include:

- *State and action spaces*: Both may not clearly describe the actual variables of the problems, which does not directly map to the software implementation of the model. This incompatibility can create confusion or ambiguity about how the problem should be implemented and may require additional effort to reconcile the two. Overall, it is important for problem descriptions to be clear and concise and to use variables and terminology that are consistent with the intended software implementation. Employing the variable notation corresponding to the software implementation can help ensure the problem is well-defined and effectively solved using available methods and tools.

- *Exogenous variables*: In Reinforcement Learning, the transition function $P(S_{t+1}|S_t, A_t)$ is used to specify the probability distribution over the next state[1] S_{t+1} given the cur-

[1] In this thesis, we adopt the convention of using t as a subscript in some situations, while in others, we do not. The t subscript indicates the temporal aspect of a variable, i.e., it denotes the variable's value at time t. This notation is typically used when dealing with time series data or models where the evolution of variables over time is critical. On the other hand, when t subscript is absent, we consider the variable in a time-agnostic or time-invariant context. In such cases, we refer to a stationary property of the variable, an averaged value over time, or a snapshot at an unspecified point in time.

rent state S_t and action A_t. This function includes all the information not captured by the state and action alone, including any exogenous information that affects the environment dynamics. In other words, the exogenous information is implicitly represented in the transition function. To implement the Reinforcement Learning algorithm in software, we must simulate the exogenous process W_t that affects the environment dynamics. The implementation requires a model of the exogenous process, which can be a challenge for some sequential decision problems. Sometimes, the exogenous process may be completely unknown or highly stochastic, making it difficult to model accurately. This characteristic can result in instability and poor performance of the Reinforcement Learning algorithm. Therefore, careful consideration should be given to modeling the exogenous process and its impact on the environmental dynamics to improve the algorithm's performance and stability.

- *Transition function*: In the RL community, the transition function represents the probability of moving from one state to another state given an action, and this function is typically denoted as $P(S_{t+1}|S_t, A_t)$. This notation assumes that the transition function can be computed for any state, action, and observation of any exogenous information. However, in the optimal control community, the transition function is often represented as a one-step transition matrix or kernel, denoted as $T(S_t, A_t)$. This notation assumes that the transition function can be computed for a single state, action, and observation of any exogenous information. In optimal control, the state and action spaces are often continuous and infinite-dimensional, making it difficult to represent the one-step transition dynamics as a finite-dimensional matrix or kernel. Additionally, the dynamics in optimal control problems are often complex and may involve differential equations, making it challenging to obtain an analytical solution. Therefore, it is often impossible to compute one-step transition matrices or kernels in practice. On the other hand, in Reinforcement Learning, the state and action spaces are often discrete or low-dimensional, and the dynamics can be modeled as a simple function that maps the current state, action, and observation to the next state. This function can be easily computed and represented as a lookup table or a neural network. Therefore, RL transition functions are typically computable in practice.

- *Objective function*: While the reward function specifies the immediate reward associated with taking action in a particular state, it does not provide a complete picture of the overall objective of the agent. In practice, the agent's objective is often implied by the reward function and the underlying problem being solved. However, there are

many ways to define the objective function for a sequential decision-making problem, and it is important to choose a suitable objective that aligns with the desired behavior of the agent.

We take guidance from Powell (2021) and adopt their proposed framework that we believe can support the explanation of all employed methods in this work. Since this mathematical framework primarily employs stochastic optimal control notations, in the subsequent sections, we draw a parallel between the notations used in Sutton and Barto (2018) and Powell (2021).

The adopted mathematical framework in this thesis is not confined to a specific RL or stochastic control model. This universality facilitates comparisons between solutions derived from these two research communities, thereby promoting a broader understanding and evaluation of the methods employed.

3.1 Decision problem mathematical framework

The framework proposed by Powell (2021) utilizes the following mathematical notation and decision model structure:

1. **State variables:** Similar to the state space used in the Markov Decision Process, the state variable S_t is a history function containing the necessary information to compute costs, rewards, and transition functions. In this framework, S_0 is distinguished from other $S_t, \forall t > 0$, because it contains all the deterministic variables, initial values of dynamic parameters and beliefs, or parameters, of the probability distribution of the unknown parameters.

   The state variable S_t comprises three types of information: Physical state[2], S^R, is the type of variable that usually appears as constrain in the problem such as resources like inventory level; Information state, S^I, is any deterministic information that can evolve exogenously or controlled by the decisions such as a change in prices; Belief state, S^B, the information about the distributional information of the exogenous variables, such as the mean and the covariance matrix of a multivariate normal distribution.

[2] Please note that in some specific instances, we have used our notation to avoid conflicts with other notations used within this text. We carefully considered the choice made to enhance the clarity and consistency of this thesis. Our notation is only implemented where it significantly aids understanding and avoids ambiguity. In other cases, we kept Powell (2021) notation. S^R is indeed one of those instances where we have opted for our notation instead of R from Powell (2021). We decided to introduce our notation here to avoid confusion with R, which is, in the ASATP, the return variable.

2. **Decision variables:** The decision variable can be discrete, continuous scalar or vector, integer vector, categorical, or binary. Instead of the action notation A_t, we use the notation X_t for decisions. To denote the policy that gives the decision, we use $X^\pi(S_t)$. The variable π carries the information about the type of function $f \in \mathcal{F}$, and its respectively tunable parameters $\theta \in \Theta^f$, if any.

3. **Exogenous information:** In contrast to the approach commonly taken in the RL community, our adopted mathematical framework explicitly treats specific variables, referred to as exogenous information, in a distinct manner. In the context of this thesis, "exogenous information" refers to variables that are outside the control of the decision-making process but can influence the state and outcome of the system. These variables are treated differently from the state variables, which are directly influenced by the actions taken in the system. This type variable is denoted as W_t, representing any new information discovered at time t. The nature of W_t can vary: it might be entirely exogenous or partially dependent on the current state or action. Furthermore, it could be stationary or non-stationary. For instance, in active trading systems, $R_t \in \mathcal{R}$ might represent the price change (or return) between times $t-1$ and t. In the context of lot-sizing problems, the exogenous variable $D_t \in \mathcal{D}$ could represent the demand from time $t-1$ to time t. The distribution of future information W_t can be described by a mathematical model or a data-driven exogenous source. It is also possible that W_t could be a function of the state and action, symbolized as $W_t(S_{t-1}, X_{t-1})$, although this dependency might sometimes be omitted for simplicity.

4. **Transition function:** The transition function is crucial in Reinforcement Learning since it describes the probability distribution of the next state variables and contribution given the current state variables and decisions. The notation adopted for the transition function is represented by S with the superscript notation M:

$$S_{t+1} = S^M(S_t, X_t, W_t) \tag{3.1}$$

When the learning agent does not rely on the transition function to learn policies, it is referred to as model-free. This approach estimates the value of states and actions based on experience or samples without explicitly using a model of the environment. In contrast, model-based approaches utilize the transition function to predict the next state, given the current state and action. In this work, we use the term "model-free" to denote any Bellman-based approach in which the transition function is assumed unknown. In contrast, we use the term "model-based" when

the learning process of the autonomous agent relies on an explicit knowledge of the transition function.

5. **Objective functions:** The objective function notation adopted is the contribution $C(S_t, X_t)$ that depends on the decision, the state, and the information. Here, we suppress the W_t notation for the objective function. Another form of the objective function is the expected sum of contributions:

$$\max_{\pi} \mathbb{E}_{S_0} \mathbb{E}_{W_1,...,W_T|S_0} \left\{ \sum_{t=0}^{T} C\left(S_t, X_t^{\pi}\left(S_t\right)\right) \mid S_0 \right\} \tag{3.2}$$

Each contribution depends directly on the initial state and the sequence of exogenous information
$(S_0, W_1, W_2, ..., W_T)$.

In this thesis, state-independent and state-dependent problems are distinguished, as in Powell (2021). The state-dependent case assumes that the transition function explicitly depends on the current state.

Powell (2021) utilizes the notation $F(X_t, W_t)$ for the state-independent objective function and $C(S_t, X_t)$ for the state-dependent objective function. However, in this work, for the sake of simplicity, we have chosen to use the $C(S_t, X_t)$ notation for both types of objective functions.

We also assume a finite time horizon, while in the Markov Decision Process, the common assumption is the infinity time horizon. In our problem instance, the finite time horizon is better suited as we further detail.

Another important differentiation is the final and cumulative reward problem. The first is the classical stochastic search problem, on which the goal is the final decision. The latter, cumulative rewards, depicts problems in which we must learn on the job, which necessitates optimizing the sum of the incentives we obtain and eliminating the necessity for a final exam[3].

[3]If we are dealing with a state-independent problem, we employ $\max_{\pi} \mathbb{E}_{S^0} \mathbb{E}_{W^1,...,W^N|S^0} \mathbb{E}_{\widehat{W}|S^0} F\left(X^{\pi}\left(S^N\right), W\right)$ for the final reward, where W is the process of exogenous information generated in the environment, and N is the budget.

For the cumulative reward of the state-independent problem objective, we have $\max_{\pi} \mathbb{E}_{S^0} \mathbb{E}_{W^1,...,W^N|S^0} \sum_{n=0}^{N-1} F\left(X^{\pi}\left(S^n\right), W^{n+1}\right)$.

For the state-dependent final reward, we employ an objective $\max_{\pi^{lrn}} \mathbb{E}_{S^0} \mathbb{E}^{lrn}_{W^1,...,W^N|S^0} \mathbb{E}^{imp}_{S|S^0} \mathbb{E}_{W|S^0} C\left(S, X^{\pi^{imp}}\left(S \mid \theta^{\pi^{imp}}\right), W\right)$.

Finally, for the state-dependent cumulative reward, the objective is given by $\max_{\pi} \mathbb{E}_{S_0} \mathbb{E}_{W_1,...,W_\tau|S_0} \left\{ \sum_{t=0}^{T} C\left(X^{\pi}\left(S_t\right), S_t, W_t\right) \mid S_0 \right\}$

3.2 Active Single-Asset Trading Problem

The main objective of this section is to provide a comprehensive description of the Active Single-Asset Trading Problem (ASATP), a complex and challenging problem that traders face in the financial market. In order to generate profit, traders actively trade a single asset over time to capitalize on market anomalies. To determine the optimal moment to trade an asset, traders observe various market data such as financial reports, news, asset price time series, and financial indexes. Using market data, the trader estimates the future asset price and determines whether to purchase or borrow an asset to sell. These two decisions may increase or reduce the trader's cumulative profit depending on the actual price movement. When traders purchase an asset, they become the owner of that item, assuming a long position. On the other hand, traders anticipating a decline in the asset's price assume a short position by borrowing the asset and selling it, incurring debt. Assuming a short position requires traders to borrow the asset from a financial institution, such as a brokerage business, and sell it at the current market price. The trader must eventually repay the borrowed asset; the profit or loss is the difference between the sale and buy price. Therefore, traders must observe all available market information at each time step, determine which position (long or short) to take, and observe price movements affecting their wealth.

3.2.1 Mathematical description of the problem

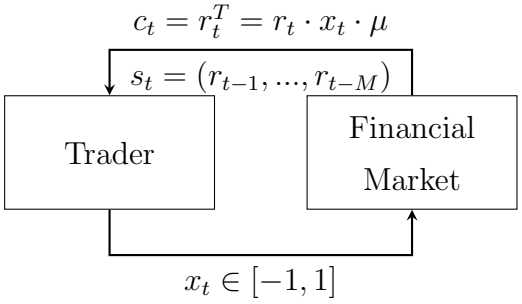
In our studies, we presume that traders can make optimal trading decisions based solely on the asset price history. Technical analysts commonly make this assumption, particularly for high-frequency trading in which traders must make decisions every hour or minute (KIRKPATRICK II; DAHLQUIST, 2010). Instead of using the past asset price time series directly, we construct and utilize a series of previous asset price change values $r_t, ..., r_1$ at each time step t. We define the variable $R \in \mathcal{R}$ as the difference between the asset price at time t and the asset price at time $t+1$, where $P \in \mathcal{P}$ is the price at time step t of a past time series of prices values $p_t, ..., p_1, p_0$. Using the asset price changes rather than the asset price itself has several advantages. The correlation between P_t and P_{-1} can result in a multicollinearity issue that diminishes the statistical significance of the independent variable (ALLEN, 1997). Using asset price changes overcomes this issue as it is less correlated with past asset prices. Additionally, using the asset price changes results in a time series with favorable statistical features, such as stationarity (CAMPBELL; LO; MACKINLAY, 1997). The stationarity of the time series makes it less susceptible to the

multicollinearity issue, which can lead to more accurate trading decisions by the trader.

In an ideal scenario, the agent would have access to the entire asset price return history to understand market conditions comprehensively. However, a fixed-size window of past price time series can provide sufficient information about the current market while maintaining relevant details. The state space is defined by the asset price return time series in a predetermined window of M hours. To make the subsequent trading decision, the trader observes only the M past asset price changes $r_{t-1}, ..., r_{t-M}$. For high-frequency trading, using a fixed window size presupposes that recent prices can provide a reliable indication of future asset prices. Furthermore, setting a constant window size M is common in other research works (ABOUSSALAH; LEE, 2020; DENG et al., 2017; ALMAHDI; YANG, 2019; WU et al., 2020; YU et al., 2021).

Having established the market information, we focus on characterizing the trader's action options and how they affect the trader's wealth. We assume that at each time step, the trader can only take a long or short position with a fixed position size of μ, simplifying the problem and avoiding concerns related to the magnitude of asset price changes (MOODY et al., 1998). By fixing the value of μ, we also achieve the stationary asset price variations with the aforementioned statistical advantages (CAMPBELL; LO; MACKINLAY, 1997). While this assumption is only required during model training, it can be relaxed during actual operations. Depending on their position, traders will receive a positive or negative return. If the asset's price rises, traders who own it will receive a positive return, while those who borrowed and sold it will receive a negative return. Thus, long and short positions can be modeled as variations in the eventual asset price return signal. We use the superscript T to differentiate the trader's return from the asset price return. If the trader takes a long position, their asset price return will have the same sign as the asset price return ($R_t^T = R_t$), but if the trader takes a short position, the sign will be negative ($R_t^T = -R_t$). As the trader can convert a portion of the μ available assets to a long or short position, we can interpret the trader's decisions at each time step as a continuous spectrum between -1 and 1. In Figure 1, we present our model of the trader's relationship with the market.

Figure 1: Model of the trader interaction with the financial market.



Here, we unfold the mathematical formulation of the problem at hand:

$$\max . \sum_{t=1}^{T} r_t^T \tag{3.3}$$

$$r_t = p_t - p_{t+1}, \qquad \forall t \in [1, T] \tag{3.4}$$

$$s_t = r_{t-1}, ..., r_{t-M}, \qquad \forall t \in [M+1, T] \tag{3.5}$$

$$\text{s.t. } s_t = r_{t-1} \parallel s_{t-1}, \qquad \forall t \in [M+1, T] \tag{3.6}$$

$$-1 \leq x_t \leq 1, \qquad \forall t \in [1, T] \tag{3.7}$$

$$r_t^T = x_t \cdot r_t \cdot \mu, \qquad \forall t \in [1, T-1] \tag{3.8}$$

The expression (3.3) outlines the objective function to maximize the total accumulated profit up to time step T. The objective is calculated based on the asset price change at each time step, as defined in equation (3.4). The state space, denoted in equation (3.5), is then defined for each time step after the initial window. The state transition is then given by the equation (3.6). To ensure the trader's action falls within the desired range, equation (3.7) constrains this action between -1 and 1 at each time step. The trader's return at each time step is then defined by equation (3.8).

The market reconfiguration due to the trader's position is an essential element that must be considered. It is commonly assumed that individual trades do not affect market supply and demand, as we denote in equation (3.6). This assumption is valid for small trades where market trade quantities are considered, such as the number and size of trades. Most literature typically adopts this assumption (ABOUSSALAH; LEE, 2020; DENG et al., 2017; MOODY et al., 1998). However, the trader's position may affect the market for large trades. This phenomenon happens because a significant trade by a trader may lead to a significant shift in supply and demand. For large trades, it is

necessary to consider the impact of the trader's position on the market in order to make more accurate predictions. In this work, we assume that the trader's position does not significantly impact the market and adopt the common assumption that market supply and demand are unaffected by individual trades.

In summary, this work formulates the problem as a decision-making process in which the trader aims to maximize their total return based on market observations. The trader observes a window of past asset price changes, which serves as the state space, and decides whether to take a long or short position at each time step, represented by the decision variable X_t. The market's supply and demand will affect asset prices, and subsequently, the trader's asset price return R_t for the chosen decision can be calculated. This iterative process repeats at each time step. The goal is to automate this process effectively to maximize the total accumulated profit of the trader.

In the following section, we will dissect the elements of the ASATP problem within the mathematical framework proposed by Powell (2021). This framework will serve as the foundation for our discussion and analysis of the ASATP problem. The solution to the ASATP problem requires a robust mathematical framework to model its decision process accurately. Thus, we will define the ASATP problem within this mathematical framework. This definition will delineate the states, actions, transition probabilities, and reward structures. By doing this, we establish a clear theoretical basis that can be used to formulate solution approaches to the problem.

3.2.2 Decision problem mathematical framework for the Active Single-Asset Trading Problem

Recent works employing RL (DENG et al., 2017; ABOUSSALAH; LEE, 2020; FELIZARDO et al., 2022a), use Markov Decision Process (MDP) to model the decision problem of the Active Single-Asset Trading Problem (ASATP). Adapting from our previous work (FELIZARDO et al., 2022a), we employ the mathematical framework presented by Powell (2021) instead of the MDP. By employing this different mathematical framework, we hope to give a more detailed description of the problem that helps the understanding of our proposed solution.

State variables: We divide the state variables in the three categories suggested in Powell (2021), physical, information, and belief states:

- *Physical state:* In the ASATP problem, the physical state of the agent refers to the

number of assets the agent owns and whether the agent is long or short in that asset. The number of assets is denoted by $\mu = 1$ in this work (as also assumed in Moody and Wu (1997)). The physical state variable $S_t^R \in [-1, 1]$ represents whether the agent is long ($S_t^R = 1$) or short ($S_t^R = -1$) in the asset. The physical state can be defined as the product of the number of assets and the asset's current price, as expressed by the equation $S_t^R = \mu_t P_t$. However, since we assume $\mu = 1$ for all trades, the physical state will equal the previous decision variable.

- *Information state:* The price is usually assumed to be provided exogenously, which is reasonable since it is impossible to capture all the information about the market state. Here we assume that the information state variable can be established by the history of price changes sufficiently large, $S_t^I = r_{r-1}, ..., r_{t-M}$. This assumption is important since the state variables must contain all the information needed to model the process. However, the reality is that it is impossible to generate state variables that can precisely model the complex relation of market agents and numerous sources of information (news, balances, order books, different price time series, and others). Therefore, we give as much information as possible using only the asset price history, a common technical analysis strategy. Here, we assume that the information captured in the price history of the asset traded is sufficient to establish a profitable policy.

- *Belief state:* In this work, we assume that the historical price data, which serves as the information state variable, can provide useful information about the exogenous factors that affect the market process. The data can give us the parameters of the function that models the exogenous information, with the price change at time t given by $R_t = W_t(\theta, r_{t-1}, ..., r_{t-M})$, where θ represents the parameters that describe the exogenous information distribution for the data. By using the historical price data, we can estimate the parameters of the function that models the exogenous information, which can be used to predict future price changes. This information is crucial for the agent to make informed decisions and develop profitable trading policies.

The state is then composed of the three state variables,
$S_t = (S_t^R, S_t^I, S_t^B) = (\mu_t p_t, r_{t-1}, ..., r_{t-M}, \theta)$. Here, since we assumed a fixed size of μ, the position is deterministically affected by the agent's decision, and the data employed in the model construction is fixed, the only state variable that needs to be modeled $r_{t-1}, ..., r_{t-M}$ and the decision. Therefore, we suppress the notation to $S_t = (r_{t-1}, ..., r_{t-M})$.

Decision variables: The only decision variable we assume is the next agent position, defined by the scalar, $X_t \in [-1, 1]$.

Exogenous information: Here, we assume that the only exogenous information necessary to model this decision process sufficiently is the price change, R_t.

Transition function: One common assumption adopted by other works is that the agent decision does not change the prices of the assets in the market (ABOUSSALAH; LEE, 2020; ALMAHDI; YANG, 2019). In this work, we also make this assumption, which then directly affects the way we model the transition function. The transition function will depend only on the new exogenous information and the previous decision, being $S_{t+1} = S^M(S_t, R_t)$. This state transition occurs by the inclusion of the most recent price change R_t and the exclusion of the oldest price change R_{t-M}. This notation helps us understand why this is considered a contextual bandit problem (at least given the assumptions made) and not a full RL (using Sutton and Barto (2018) nomenclature) case. If this problem were a full RL case, the transition function, S^M, would be a function of the agent decision.

Objective function: In this work, we adopt a state-dependent case with cumulative contribution, the cumulative agent return. The expression gives the objective function:

$$\max_{\pi} \mathbb{E}_{S_0} \mathbb{E}_{r_1,\ldots,r_T|S_0} \left\{ \sum_{t=0}^{T} C\left(X^{\pi}\left(S_t\right), S_t, R_t\right) \mid S_0 \right\} \tag{3.9}$$

where S_0 denotes the initial state, $X^{\pi}(S_t)$ is the decision policy, and R_t represents the trader return received at each time step. The function $C\left(S_t, X^{\pi}\left(S_t\right), R_t\right)$ is the cost function[4], which depends on the current state, decision, and reward. Assuming a state-dependent objective function means the transition function depends on the current state. This assumption is consistent with the efficient market hypothesis debate, which suggests that the current state of the market affects the probability distribution of future returns. To justify the use of past time series of prices, we assume that the past returns have at least some contribution to estimating the next state. This assumption is based on the idea that past returns may provide information that can be used to predict future returns.

[4] In the context of this thesis, the cost function $C\left(S_t, X^{\pi}\left(S_t\right), R_t\right)$ operates without a specified unit of measurement. This approach allows for a degree of abstraction, focusing on the mathematical or algorithmic properties of the system rather than its implementation in a specific real-world scenario. While real-world applications of such cost functions typically involve units of measurement or a combination of such units – financial, temporal, energetic, or otherwise – they have been omitted here to emphasize the generality of the proposed methods and theories. It is essential to note that in real-world applications, the units of cost must be appropriately defined and understood, as they can significantly impact decision-making and the interpretation of results

3.3 Stochastic Discrete Lot-Sizing Problem

The optimization of production involves deciding how many units to produce during each period to meet the demand for those units. When production capacity is limited and uniform, this is known as the capacitated lot-sizing problem. In such a situation, the challenge is to minimize the costs associated with inventory keeping and setup. With deterministic elements such as demand, mathematical optimization methods can be used to tackle production scaling difficulties. When uncertainty is present, the goal includes the minimization of lost sales as the demand may not be met.

Stochastic programming has been utilized as a solution method to investigate the capacitated lot-sizing problem under uncertainty. This method has been applied to study the problem with large and small time buckets. In this work, we focus on a discrete setting with small time buckets, where the planning horizon is divided into short time intervals (in contrast with large time buckets on which the planning horizon is divided into larger, more coarse intervals). When this division of the planning horizon is assumed, the production plan must account for machine configuration adjustments if there is insufficient capacity to manufacture multiple products. Considering the setup cost as a cost source is one important variable for the objective function of the problem. The policy or planning must balance the expenses of holding inventory and starting up the machines. When the problem's stochastic processes are considered, and we assume that the objective is to minimize long-term costs by planning on a horizon comprised of small time intervals and limited capacity, we have a Stochastic Discrete Lot-Sizing Problem (SDLSP).

The problem that we try to solve in this work is detailed following the structure suggested by Haase (1994), Xiao et al. (2015). In this work, we make subsequent assumptions about the resource constraints. We test different numbers of items to be produced by a different number of machines, and we work on scarcity scenarios. In these scenarios, we assume that resources are limited and production capacity is insufficient to meet demand, leading to stockouts and lost sales.

In the SDLSP, we consider each machine's "all-or-nothing" production. The "all-or-nothing" assumption means that the machine will only produce a given item amount each time bucket or nothing. It is also assumed that the production capacity remains constant over time, a simplification adopted in this work. However, the problem becomes more intricate when different setup costs are introduced between items and machines and when setup times are different and may last for more than one period. The setup cost depends on the current item being produced and the item that will be produced next. We

may encounter different setup times that extend beyond a single period when modifying the setup. When changing the setup time, we assume that the last setup is carried over during *idle* (setup transition state). This assumption is also known as the conservation of setup rule (FLEISCHMANN; MEYR, 1997). Furthermore, we assume the machine can be deactivated or activated, with associated inactivation, deactivation, and action times and costs.

We consider a single-level or stage production, which is the simplest case of production structure in which the product in a single period does not depend on the presence of other items in inventory. This production structure means the product can be treated independently and planned for in a single period. For example, if a company is producing two different products, each with its production line, and the production of one product does not depend on the production of the other, then the company is operating in a single-level or stage production structure. This structure is considered the simplest production structure case since the production planning and control can be carried out for each item independently. A more complex scenario would be the multi-level lot-sizing problem (HAASE, 1994), which is beyond the scope of this work.

In addition to production and setup costs, external demand for the items is also a crucial factor in this problem. This work focuses on the stochastic demand case, where we limit the analysis to stationary demand and do not experiment with generated demands with autocorrelation. With the assumption of the demand for our problem, we can define the relevant costs that comprise our objective function:

- **Setup costs**: The cost of setting up a machine to produce a different item, including startup time, which we also consider in this work.
- **Inventory costs**: The holding costs, which can be a combination of the opportunity cost of capital and the direct costs associated with storing items (space, management, and other expenses).
- **Lost-sales costs**: The direct loss of profit from not making a sale, as well as indirect costs related to customer behavior and product consumption.

Our objective is to minimize the sum of these three costs over the planning horizon. There are two possible configurations: finite or non-finite time horizons. We can approximate the solution for a non-finite time horizon by considering a sufficiently large number of decision steps. We simulate a finite number of decision steps (or "buckets") to allocate the production of items exploring different time horizon sizes.

3.3.1 Mathematical description of the problem

We define I as the max number of items, M as the number of machines, and T as the number of time steps:

- $\mathcal{I} = \{1, \ldots, I\}$, the set of items (product types).
- $\mathcal{T} = \{0, \ldots, T\}$, the set of time periods.
- $\mathcal{M} = \{1, \ldots, M\}$, the set of (not necessarily identical) machines. We will also use the notation $\mathcal{M}(i)$ to denote the set of machines that are able to produce item $i \in \mathcal{I}$.
- $\mathcal{I}(m)$, the set of states that machine m can assume. It comprises the subset of items this machine can produce plus the idle state.
- $\mathcal{M}(i)$, the set of machines that can produce $i \in \mathcal{I}$.
- $\mathcal{C} \subseteq \mathcal{M} \times \mathcal{I}$ the possible set of current machine states when we schedule production. The pair (m, i) belongs to $\mathcal{C}$ if machine m has been producing item i immediately before $t = 0$.

The models need the following parameters:

- $p_{i,m}$: number of items i produced by the machine m.
- $f_{i,m}$: setup cost if machine m starts to produce item i.
- $c_{i,m}$: setup loss if machine m starts to produce item i.
- h_i: holding cost for storing item i.
- l_i: lost sales cost of item i.

Finally, we consider the decision variables:

- $I_{i,t}$: inventory of item i at time t.
- $z_{i,t}$: the lost sales of item i at time t.
- $x_{i,m,t}$: binary variable equal to 1 if machine m is producing item i at time t.
- $\delta_{i,m,t}$: binary variable equal to 1 if machine m has done a setup between time $t-1$ and time t.

Each successive discrete time interval, delineated as t, entails a transformation of the machine configurations, an operation that subsequently imposes an associated setup cost. After this initialization, each non-idle machine is responsible for the fabrication of $p_{i,m}$ units, granted the absence of any configuration changes, or contrarily, yields $p_{i,m} - c_{i,m}$ units in the event of a setup. Upon culmination of the production phase, the demand, symbolized as $d_{i,t}$, materializes. The products available at this part are thus leveraged to satisfy demand. As we approach the terminal phase of this sequence, the calculations for the cost of lost sales and inventory carrying costs are performed. For each product, the per unit inventory holding cost is denoted as h_i, while the unit cost associated with sales forgone is denoted as l_i. This progression of events is graphically elucidated in Figure 2, providing a comprehensive panorama of the process trajectory. The pre-decision state is represented by the state variables at time t, which include information about the current inventory levels, the backlog of orders, and the expected demand for the next period. The decision at time t is represented by the control action x_t, which determines the production levels for the period.

Figure 2: Flow of operations

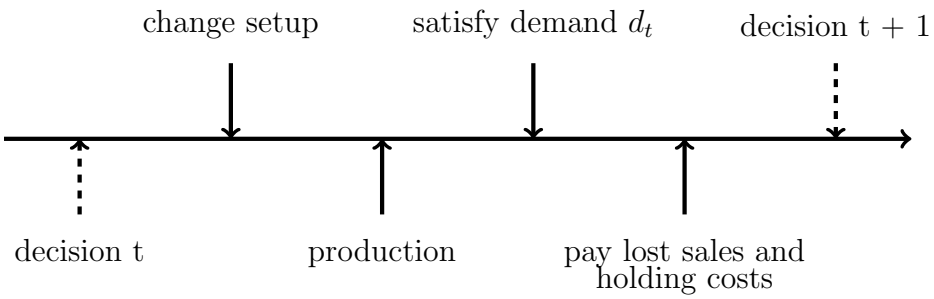
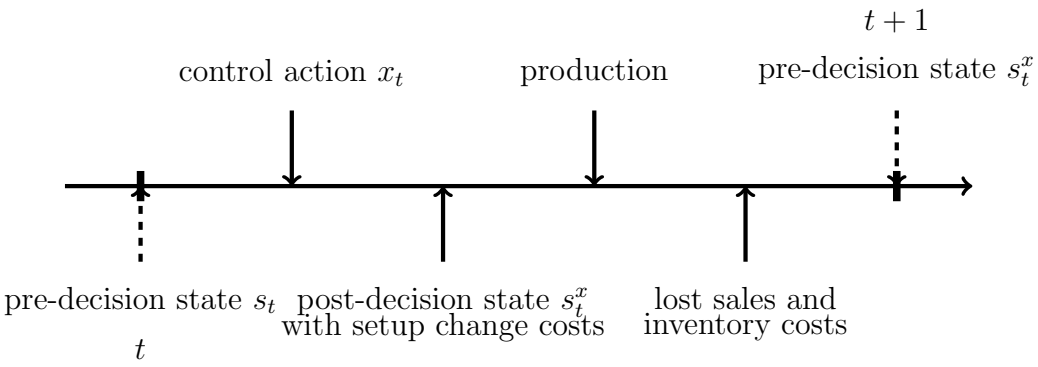


Figure 3: Flow of operations with pre-decision and post-decision notation



In this thesis, we present the employed mathematical formulation of the DLSP (deterministic case). The mathematical formulation begins with the objective function (3.10). Following this, present the constraints, encompassing aspects such as inventory and pro-

duction (3.11), machine setting encoding (3.12), (3.13), (3.14), and (3.15), inventory (3.18), and variable types (3.19) and (3.20):

$$\min \sum_{i=1}^{I} \left[\sum_{m \in \mathcal{M}(i)} \sum_{t=0}^{T} f_{i,m} \delta_{i,m,t} + \sum_{t=1}^{T} (h_i I_{i,t} + l_i z_{i,t}) \right] \tag{3.10}$$

$$\text{s.t. } I_{i,t} - z_{i,t} = I_{i,t-1} + \sum_{m \in \mathcal{M}(i)} (p_{i,m} x_{i,m,t-1} - c_{i,m} \delta_{i,m,t-1}) - d_{i,t} \qquad \forall i \in \mathcal{I},\ t \in \mathcal{T}^+ \tag{3.11}$$

$$\sum_{i \in \mathcal{I}_0(m)} x_{i,m,t} \leq 1 \qquad \forall m \in \mathcal{M}, t \in \mathcal{T} \tag{3.12}$$

$$x_{i,m,t} = 0 \qquad \forall i \notin \mathcal{I}_0(m), t \in \mathcal{T} \tag{3.13}$$

$$\delta_{i,m,t} \geq x_{i,m,t} - x_{i,m,t-1} \qquad \forall\ m \in \mathcal{M},\ i \in \mathcal{I}_0(m)\ t \in \mathcal{T}^+ \tag{3.14}$$

$$x_{i,m,t-1} - x_{i,m,t} + \varepsilon \leq (1+\varepsilon)(1 - \delta_{i,m,t}) \qquad \forall\ m \in \mathcal{M},\ i \in \mathcal{I}_0(m),\ t \in \mathcal{T}^+ \tag{3.15}$$

$$\delta_{i,m,0} \geq x_{i,m,0} - \mathbb{1}_{(m,i) \in \mathcal{C}} \qquad \forall\ m \in \mathcal{M},\ i \in \mathcal{I}_0(m) \tag{3.16}$$

$$\mathbb{1}_{(m,i) \in \mathcal{C}} - x_{i,m,0} + \varepsilon \leq (1+\varepsilon)(1 - \delta_{i,m,0}) \qquad \forall\ m \in \mathcal{M},\ i \in \mathcal{I}_0(m) \tag{3.17}$$

$$I_{i,0} = \bar{I}_{i0} \qquad \forall i \in \mathcal{I} \tag{3.18}$$

$$I_{i,t} \in [0, I_{\max}], z_{i,t} \in \mathbb{R}^+ \qquad \forall i \in \mathcal{I},\ t \in \mathcal{T} \tag{3.19}$$

$$\delta_{i,m,t}, x_{i,m,t} \in \{0,1\} \qquad \forall\ i \in \mathcal{I}, m \in \mathcal{M}, t \in \mathcal{T}, \tag{3.20}$$

The objective function, as denoted by (3.10), comprises an aggregate of setup, holding, and lost sales costs. Notably, the last element is incorporated even within a deterministic model, as there exist instances where it may prove beneficial to not fully meet the demand to evade larger inventory expenditures. An exemplification of this scenario surfaces when the production volumes $p_{i,m}$ are substantial while the demand remains critically low. Under such circumstances, it is strategically prudent to forfeit some sales rather than initiate a new production cycle and store it.

The constraints depicted by (3.11) establish the balance of inventory, while those

denoted by (3.12) fortify the "all-or-nothing" supposition. Further, constraints (3.13) prohibit the production of items that fall beyond the manufacturing capabilities of a machine. Constraints (3.14), and (3.15) enforce that $\delta_{i,m,t}$ must adopt a value of one if and only if there is a change in machine settings. Specifically, constraints (3.15) are integral to prevent the model from setting $\delta_{i,m,t} = 1$ to curtail production and better align with demand even if a setup is not necessitated. The parameter ε symbolizes a minimal number requisite for logical constraints.

Lastly, constraints (3.16), and (3.17) mandate the initial setup, while constraints (3.19), (3.20) prescribe the variable types. In the interest of simplicity, we assume that all items possess an equivalent maximum inventory, designated as $I_{\max}$. It is worth attention that Model (3.10) - (3.20) possesses complete knowledge of the demand throughout the entire time horizon, an assumption not typically based in practical situations. Regardless of this limitation, we will utilize this model as a benchmark. After this, we will refer to Model (3.10) - (3.20) as the Perfect Information (PI) model.

In this work, we consider the role of demand as a risk factor in the context of stochastic programming. A common method for modeling uncertainty in this field is using a scenario tree. This tool provides a structured way to represent a variety of possible outcomes. The model expressed in Equations (3.10) - (3.20) can be adjusted to account for stochastic demand, enhancing its ability to capture real-world unpredictability. As per the notation established in Brandimarte (2006), we define the following terms:

- The set of nodes in the scenario tree is $\mathcal{N}$, and $\mathcal{N}^+ = \mathcal{N} \setminus \{0\}$.
- $p(n)$ the parent of node $n \in \mathcal{N}^+$.
- $\pi^{[n]}$ the unconditional probability of node n ($\pi^{[0]} = 1$).
- $d_i^{[n]}$ the demand for item i at node $n \in \mathcal{N}$.

We use the term branching factor to denote the number of child nodes originating from each parent node at a specific level within the tree. For example, a $[2, 2, 2]$ branching factor represents a binary tree spanning four-time instances. These instances include the present time, corresponding to the root node, and extend to eight scenarios. Each scenario is defined as a path of nodes extending from the root node to a leaf node, symbolizing the progression of the stochastic process over time. All nodes are at the same level and share the same time instant. Within the SDLSP context, the information flow can be effectively visualized as shown in Figure 3.

Figure 3 details the information flow by including pre-decision and post-decision variables in the representation. In this Figure 3, the pre-decision state is represented by S_t, which includes the state variables observed before the decision, and the post-decision state is represented by S_t^x, which includes the state variables observed after the decision. After the decision, the system moves to a post-decision state represented by the state variables at time $t+1$, which includes information about the actual demand for the next period and the inventory levels at the end of the period.

The decision variable of the stochastic version of the problem (3.10)-(3.20) have the superscript $\cdot^{[n]}$ instead of the subscript $\cdot_t$ to identify the node $n \in \mathcal{N}$ to which they refer. For example, the inventory of item i in node n will be $I_i^{[n]}$, etc.

We introduce the Multi-Stage variant of the deterministic model as represented by equations (3.10) through (3.20). This variant is defined through the following mathematical formulations:

$$\min \quad \sum_{i=1}^{I}\left[\sum_{n\in\mathcal{N}}\pi^{[n]}(\sum_{m\in\mathcal{M}(i)}f_{i,m}\delta_{im}^{[n]})+\sum_{n\in\mathcal{N}^+}\pi^{[n]}(h_iI_i^{[n]}+l_iz_i^{[n]})\right] \tag{3.21}$$

$$\text{s.t.} \quad I_i^{[n]}-z_i^{[n]}=I_i^{[p(n)]}+\sum_{m\in\mathcal{M}(i)}(p_{i,m}x_{i,m}^{[p(n)]}-c_{i,m}\delta_{i,m}^{[p(n)]})-d_i^{[n]} \quad \forall i\in\mathcal{I},\ n\in\mathcal{N}^+ \tag{3.22}$$

$$\sum_{i\in\mathcal{I}_0(m)}x_{i,m}^{[n]}\leq 1 \quad \forall m\in\mathcal{M},\ n\in\mathcal{N} \tag{3.23}$$

$$x_{i,m}^{[n]}=0 \quad \forall i\notin\mathcal{I}_0(m), n\in\mathcal{N} \tag{3.24}$$

$$\delta_{im}^{[n]}\geq x_{i,m}^{[(n)]}-x_{i,m}^{[p(n)]} \quad \forall\ m\in\mathcal{M},\ i\in\mathcal{I}_0(m),\ n\in\mathcal{N}^+ \tag{3.25}$$

$$x_{i,m}^{[p(n)]}-x_{i,m}^{[n]}+\varepsilon\leq(1+\varepsilon)(1-\delta_{i,m}^{[n]}) \quad \forall\ m\in\mathcal{M},\ i\in\mathcal{I}_0(m),\ n\in\mathcal{N}^+ \tag{3.26}$$

$$\delta_{i,m}^{[0]}\geq x_{i,m}^{[0]}-\mathbb{1}_{(m,i)\in\mathcal{C}} \quad \forall\ m\in\mathcal{M},\ i\in\mathcal{I}_0(m) \tag{3.27}$$

$$\mathbb{1}_{(m,i)\in\mathcal{C}}-x_{i,m}^{[0]}+\varepsilon\leq(1+\varepsilon)(1-\delta_{i,m}^{[0]}) \quad \forall\ m\in\mathcal{M},\ i\in\mathcal{I}_0(m), \tag{3.28}$$

$$I_i^{[0]}=\bar{I}_{i0} \quad \forall i\in\mathcal{I} \tag{3.29}$$

$$I_i^{[n]}\in[0,I_{\max}], z_i^{[n]}\in\mathbb{R}^+ \quad \forall i\in\mathcal{I},\ n\in\mathcal{N} \tag{3.30}$$

$$\delta_m^{[n]}, x_{i,m}^{[n]}\in\{0,1\} \quad \forall m\in\mathcal{M}, i\in\mathcal{I}, n\in\mathcal{N}. \tag{3.31}$$

The objective function, denoted as (3.21), aims to calculate the expected total cost, which includes setup costs, lost sales, and holding costs. The constraints (3.22) to (3.28) serve as the stochastic equivalents to the deterministic constraints (3.11) to (3.17), respectively. It is important to note that, given a solution to the model, the action executed by the agent at time t is represented as $\mathbf{X}_t=[x_{i,m}^{[0]}]_{m\in\mathcal{M},i\in\mathcal{I}}$. In the chapters, the model denoted by (3.21) to (3.31) will be referred to as the Multi-Stage (MS) model.

3.3.2 Decision problem mathematical framework for the Stochastic Discrete Lot-Sizing Problem

Considering that the mathematical structure of our problem is founded on the principles of stochastic optimal control, we are adapting this formulation to align with the decision problem framework as presented by Powell (2021):

- **State variable**: we divided the state variable in:

  Physical state: $\mathbf{I_t}$, the inventory after replenishing inventory, before serving the demand that we define in (3.11) for the deterministic case and (3.22) for the non-deterministic.

  $\mathbf{I}_t \in \mathcal{I}$ where $\mathcal{I} = \{1, \ldots, I\}$ the set of items represents the physical state, S^I of the items that are being produced and managed, similar to the resources like inventory level.

  $\mathbf{M}_t \in \mathcal{M}$ where $\mathcal{M} = \{1, \ldots, M\}$ the set of machines, is the collection of setup states which we encode as a vector of m entries, each entry in the $[I]$ interval.

  Information state: $\mathcal{I}(m)$, and $\mathcal{C} \subseteq [M] \times [I]$ defined before. $\mathcal{I}(m)$ the set of states that machine m can assume. It is composed of the set of items that the machine can produce plus the idle state. $\mathcal{C} \subseteq \mathcal{M} \times \mathcal{I}$ the set of initial condition. The couple (m, i) belongs to $\mathcal{C}$ if machine m is producing item i immediately before $t = 0$. All these states are deterministic information variables that can evolve exogenously or be controlled by decisions, such as the set of machines, the items each machine can produce, and the initial production conditions.

  Belief state: The belief state includes the unconditional probabilities of each node, $\pi^{[n]}$, and the demand for each end item at each node, $d_m^{[n]}$.

  Therefore, the state variables are summarized as $S_t = (\mathbf{I}_t, (X_{t-1}, \delta_{t-1}), \pi^{[n]})$ In the subsequent solutions presented in this subsection, we suppress the belief state notation as it is part of the initial assumptions of the problem.

- **Exogenous information**: The demand is one of the exogenous information, $W_t = d_{i,t}$, and is a crucial component of our problem, as it represents the number of items that need to be sold at each time step. To account for this uncertainty, we model the demand as a risk factor and employ a scenario tree to represent it. The demand is also incorporated into the belief state, $\pi^{[n]}$ and $d_m^{[n]}$, as it is used to generate the scenarios in the scenario tree.

- **Transition function**:$S_{t+1} = S^M(S_t, X_t, W_t)$, can be divided into two main components, the inventory level and the machine setup. First, we present the equation that describes the transition of the inventory level given the action X_t:

$$I_{i,t+1} = \left[I_{i,t} + \sum_{m=1}^{M} (p_{i,m} x_{i,m,t} - c_{i,m} \delta_{i,m,t}) - d_{i,t} \right]^+ \tag{3.32}$$

  The machine setup change is directly changed, but the action:

$$\delta_{i,m,t} \geq x_t - x_{t-1} \tag{3.33}$$

- **Decision variables**: $I_{i,t}$, $z_{i,t}$, $x_{i,m,t}$, $\delta_{i,m,t}$, are the decision variables, as they are the variables that are deterministic information that can evolve exogenously or be controlled by decisions. The policy $X^\pi(S_t)$ gives the decision x_t. The decision variables are collected in vector $\mathbf{X_t}$, constrained to set $\mathcal{X}$. We include the surperscript x (e.g. $\mathbf{M^x}_t$,$\mathbf{I^x}_t$) for a postdecision statevariable.

- **Objective function**: $d_{i,t}$, $f_{i,m}$, $l_{i,m}$, $c_{i,m}$, h_i, are part of the objective function, $C(S_t, X_t)$, as they represent the costs associated with producing and managing the items. The immediate contribution depends on the setup cost, the inventory at the end of the time period, and the possible lost sales:

$$\begin{aligned} C\left(S_t, X_t\right) = & \sum_{m=1}^{M} \sum_{i=1}^{I} f_i \delta_{i,m,t} + \sum_{i=1}^{I} h_i \left[I_{i,t} + \sum_{m=1}^{M} (p_{i,m} x_{i,m,t} - c_{i,m} \delta_{i,m,t}) - d_{i,t} \right]^+ \\ & + \sum_{i=1}^{I} l_i \left[d_{i,t} - I_{i,t} + \sum_{m=1}^{M} (p_{i,m} x_{i,m,t} - c_{i,m} \delta_{i,m,t}) \right]^+, \end{aligned} \tag{3.34}$$

  where $[y]^+ \doteq \max\{y, 0\}$. Note that the first term of the immediate contribution is deterministic, while the second and third ones are stochastic (since at time t, $d_{i,t}$ is unknown).

4 REINFORCEMENT LEARNING METHODS

This chapter will delve into the mainstream state-of-the-art RL methods utilized in one of the two distinct classes of problems addressed in this thesis. Readers can refer to the works of Baird (1993), Mnih et al. (2013), Mnih et al. (2015), Schulman et al. (2017) for a more in-depth understanding of the methods employed. All the RL methods discussed can be utilized to tackle most decision-making processes, but they might be better for them. As we will observe in the Chapter 5, the trial-and-error, model-free RL strategy may not be appropriate for that type of problem. For this thesis, we have adapted the notation for the mathematical framework of Powell (2021), deviating from the commonly used RL notation[1].

Before we delve into the specific RL methods used in this thesis, it is beneficial to revisit the fundamental elements that constitute the RL framework briefly. As we know, RL is a complex field with a rich set of concepts and techniques. While we will not delve into the intricate details of each element, a high-level overview can help set the stage for the forthcoming discussions. RL is characterized by an agent learning to make decisions by interacting with its environment, receiving feedback in the form of rewards or penalties, and adjusting its actions accordingly to maximize the cumulative reward.

1. **States:** In RL, a state represents the current situation or condition of the environment. The agent's understanding of the state significantly influences its decisions or actions.

2. **Actions:** Actions refer to the choices or moves an agent can make in a given state. The action taken by the agent influences the subsequent state of the environment.

[1] Throughout this thesis, the subscript t or the parameter notation θ is occasionally omitted when referencing the value function V, the action-value function Q, or the policy function X^{π}. This simplification is made for brevity and readability, with the understanding that these functions are inherently time-dependent and parameter-dependent in the context of Reinforcement Learning. Specifically, the value and action-value functions depend on the state and action at each time step t, while the policy function depends on the current policy parameters θ. While these dependencies are always implicitly assumed, they are sometimes left out of the notation for ease of exposition. Please note that this does not affect the theoretical soundness or accuracy of the formulations presented.

3. **Rewards:** Rewards are the feedback that an agent receives after taking action. The agent's ultimate goal is to learn a policy that maximizes the cumulative reward over time.

4. **Policies:** A policy defines the agent's behavior at a given time. It is a mapping from states to actions that determines how the agent responds to the current state of the environment.

5. **Value Functions:** Value functions estimate the expected cumulative reward for each state or state-action pair under a particular policy. They play a crucial role in determining the optimal policy.

These elements form the foundation of RL and are present in various forms in all RL methods. As we navigate through the following sections, we will explore how these fundamental elements manifest in four state-of-the-art RL methods, here explained in this chapter, using the mathematical framework described in Chapter 3. Each method brings a unique perspective, offering diverse solutions to the complex problems we aim to tackle in this thesis.

4.1 Bootstrapped Thompson Sampling

Bootstrapped Thompson Sampling (BTS) (ECKLES; KAPTEIN, 2014) is a variant of the classical Thompson sampling algorithm. The idea behind Thompson sampling is to maintain a probability distribution over the expected contribution for each action and to sample from these distributions to determine which action to take at each time step. In BTS, multiple estimates of the expected contribution for each action are maintained instead of a single estimate. Each estimate is obtained by bootstrapping (i.e., sampling with replacement) from the available contribution observations for that action. According to the bootstrapped estimates, the action with the highest expected contribution is selected as the action to take at the next time step. BTS can account for uncertainty in the estimated contribution distributions by maintaining multiple estimates, making it well-suited for problems with noisy contribution signals.

Unlike full RL scenarios where state transitions play a vital role, contextual bandits focus on making optimal decisions based only on the current context. BTS is also computationally efficient compared to certain RL algorithms, especially when applied to contextual bandit problems. While BTS offers several advantages in contextual bandit

settings, it is essential to understand its limitations in full RL scenarios. In RL, actions taken affect immediate rewards and influence future states, thereby having long-term consequences. BTS, designed to focus on the current context, needs the state transition modeling crucial in RL. In the ASATP, the way it is usually modeled, we assume a contextual bandit framework. Therefore, one reasonable method to be chosen is the BTS.

Algorithm 1 Bootstrapped Thompson Sampling training

procedure BTS_TRAINING(N)
  ▷ Number of training episodes: N
  ▷ Output: Trained parameters: θ_N
  Initialize the environment and get a random initial state: $s_0 \in \mathcal{S}_0$
  Initialize the model parameters with random values: θ_0
  Initialize the episode memory: $\mathbf{M}^h$
  for $i \in (0, N]$ **do**
    $s_0 \leftarrow \text{Random}(\mathcal{S}_0)$
    ▷ Generate bootstrapped sample
    $Z \leftarrow \text{Boostrap sample}(\mathbf{M}^h)$
    ▷ Approximate a model parameter θ_i
    $\theta_i \leftarrow \text{Multiple regression models}(Z)$
    for $t \in (0, N]$ **do**
      ▷ Select the decision using each bootstrapped model
      $x_{t,b} \leftarrow X^\pi(s_t, \theta_i)$
      ▷ Estimate the reward of each action
      $C_{t,b} \leftarrow \text{Estimate contribution}(s_t, x_{t,b})$
      ▷ Sample an action based on the estimated reward
      $x_t \leftarrow \text{Sample action}(x_{t,b}, C_{t,b})$
      ▷ Take the action and observe the next state
      $s_{t+1} \leftarrow S^M(s_t, x_t)$
      ▷ Take the action and observe the contribution
      $c_t \leftarrow C(s_t, x_t)$
      $\mathbf{M}^h \leftarrow \mathbf{M}^h \cup (s_t, x_t, c_t)$
    ▷ Update the boostrapped model
    $\theta_{i+1} \leftarrow \text{Update model}(\theta_i, Z)$
  return θ_N

Algorithm 1[2] describes the BTS training procedure, a Reinforcement Learning method for solving decision-making problems. The goal is to learn a policy, which is a mapping from states to actions, that maximizes the expected cumulative reward. The algorithm runs for a fixed number of training episodes[3] N. In each episode, the algorithm starts by initializing the environment and getting a random initial state. The model parameters are initialized with random values. The episode memory $\mathbf{M}^h$ starts with an empty set. The first step of the algorithm is to sample a batch of data Z from the historical memory $\mathbf{M}^h$ to generate a bootstrapped sample Z with replacement. The batch of data Z contains a set of state-action pairs and their corresponding rewards, which is used to estimate the model's parameters. Next, the algorithm approximates a model parameter[4] θ_i using multiple regression models applied to the bootstrapped sample Z. The model parameter is used to select actions based on the current state. Then, the algorithm loops over a fixed number of time steps T. In each time step, the algorithm selects the decision using each bootstrapped model to estimate the reward of each action.

Note that up until this point, this algorithm is similar to a Supervised Learning approach. This similarity inspired us to develop the RSLSTM-A that we further explain.

Then, an action is sampled based on the estimated expected reward for each action. The selected action is taken, and the contribution and next state are observed. The contribution is added to the episode memory $\mathbf{M}^h$. Finally, after the end of the episode loop, the algorithm updates the bootstrapped model by applying the model update function

[2] Throughout this thesis, a specific notation has been adopted to distinguish between the concept of a state variable, the value of a state variable, and the state space. The upper-case notation (e.g., S) represents a state variable in the abstract. In contrast, the lower-case notation (e.g., s) denotes a specific value that this state variable can take. To denote the entire state space, we utilize the notation $\mathcal{S}$, which represents all possible states that the variable can take.

A similar notation is followed for the text's decision and other variables. The use of upper-case (e.g., X) refers to the decision variable in general, whereas lower-case (e.g., x) represents a specific decision made at a particular point in time. This consistent notation aids in distinguishing between different variable types and their uses.

[3] **Episode**: An episode typically refers to one sequence of states, actions, and rewards in a Reinforcement Learning environment. It starts from an initial state and ends at a terminal state. The agent learns from these episodes by using its rewards to update its policy, intending to maximize its reward in future episodes.

Epoch: An epoch, on the other hand, is a term used in the context of training a machine learning model, such as a neural network. One epoch is completed when the model has seen all samples in the training set once.

[4] It is worth noting that, in many instances throughout this thesis, the notation $X^\pi(s_t, \theta)$ is used, where θ represents the model parameters. However, we occasionally simplify this notation to X^π or $X^\pi(S_t)$ for practical readability. Such simplification is a common practice in machine learning literature to ensure clarity and readability, especially in contexts where the parameter dependence is understood. This simplification improves readability and reduces clutter, but it is crucial to remember that the underlying action-value function (or policy) is parameterized by θ. Despite this omission, the dependence on the parameters is always implied. Furthermore, the context and discussion around the notation should clarify when the model parameters are explicitly considered or updated.

to the current and bootstrapped data Z. The output of the algorithm is the trained parameters θ_N.

4.2 Deep Q-Network

Deep Q-Network (DQN) (MNIH et al., 2013) is a type of RL algorithm that uses a neural network to approximate the Q-function, which maps states to the expected cumulative contribution of taking a specific decision in that state. The goal of DQN is to learn the optimal policy, which is the decision that maximizes the expected cumulative contribution at each state.

The Q-Network parameters, represented by θ, are updated during training to minimize the difference between predicted and true Q-values. It uses an experience replay memory, represented by $\mathbf{M}^h$, to store transitions of the form (state, action, contribution, next state). The transitions are randomly sampled from the replay memory to update the Q-Network parameters. This process is known as experience replay, and it is used to decorrelate the data samples and break the temporal correlation in the data.

The DQN algorithm also uses an ϵ-greedy policy for action selection. With a probability of ϵ, a random action is chosen, and with the probability of $1 - \epsilon$, the action with the highest Q-value, as estimated by the Q-Network, is chosen. This ϵ-greedy heuristic allows for exploration of the state-action space during the early stages of training while exploiting the learned policy during later stages. The DQN also includes a target network, which is used to stabilize the learning process. The target network is a separate copy of the Q-Network used to compute the target Q-values during training. The target Q-values are then used to update the Q-Network parameters.

The Q-Network is trained using the Bellman equation, which states that the Q-value of a state-action pair can be defined as the immediate contribution plus the expected future contribution, discounted by a factor of γ. The Q-Network is trained to minimize the mean squared error between the predicted and target Q-values, as defined by the Bellman equation. Mathematically, this is represented as:

$$L_Q = \frac{1}{2}(Q(S_t, X_t, \theta) - (C_t + \gamma \max_{X_{t+1}} Q(S_{t+1}, X_{t+1}, \theta))^2, \tag{4.1}$$

where S_t is the current state, X_t is the current action variable, θ are the Q-Network parameters, C_t is the immediate contribution, S_{t+1} is the next state, X_{t+1} is the next action, and γ is the discount factor. The training algorithm is described in Algorithm 2.

Algorithm 2 Deep Q-Network (DQN) training

procedure DQN_TRAINING(N, γ, θ)
    ▷ Number of training episodes: N
    ▷ Output: Trained parameters: θ_N
    Initialize the environment and get a random initial state: $s_0 \in \mathcal{S}_0$
    Initialize the Q-Network with random weights: θ_0
    Initialize the historical memory: $\mathbf{M}^h$
    for $i \in (0, N]$ **do**
        $s_0 \leftarrow \text{Random}(\mathcal{S}_0)$
        for $t \in (0, N]$ **do**
            ▷ Select an action based on the epsilon-greedy policy
            $x_t = \begin{cases} \text{random action}, & \text{if } U \sim \text{Uniform}(a, b) < \epsilon \\ \arg\max_x Q\left(s_t, x_t, \theta_i\right), & \text{otherwise} \end{cases}$
            ▷ Take the action and observe the next state
            $s_{t+1} \leftarrow S^M(s_t, x_t)$
            ▷ Take the action and observe the contribution
            $c_t \leftarrow C(s_t, x_t)$
            ▷ Store the transition in replay memory
            $\mathbf{M}^h \leftarrow \mathbf{M}^h \cup (s_t, x_t, c_t, s_{t+1})$
            ▷ Sample the batch Z from the historical memory
            $Z \leftarrow \text{Sample}(\mathbf{M}^h)$
            ▷ Loss calculation using the elements of Z using:
            $L_Q = \frac{1}{2}(Q(s_t, x_t, \theta_i) - (c_t + \gamma \max_{x_{t+1}} Q(s_{t+1}, x_{t+1}, \theta_i))^2$
            ▷ Update the Q-function network:
            $\theta_{i+1} \leftarrow \text{backpropagation}(\theta_i, L_Q)$
    return θ_N

Note that different from BTS, this method considers the impact of the decisions in the subsequent states as we evaluate the state-action pair using the subsequent state-action pairs discounted by the γ factor. Therefore, the set of problems that can be solved using the DQN is much larger, but so is the problem's dimensionality. Now, we are modeling how each action affects a future chain of states and actions. Since the complexity of the problem we are trying to solve increased, we might also face other issues related to convergence of the algorithm and processing time performance. Many works have tried to solve those problems using prioritized experience replay (SCHAUL et al., 2016). Also,

as we will further see, other methods using the actor-critic approach try to overcome this convergence problem.

4.3 Actor-critic

In DRL, actor-critic architectures have two main components: the actor and the critic. The actor is the policy function, which is approximated by an ANN and is denoted $X^\pi(S)$. The critic is the value function approximated by a second ANN. It can either estimate the state-value function or the state-action value function. Note that the state-action value function can be expressed in terms of the state-value function: $V(S_t) = \mathbb{E}\left[C(S_t, X_t) + \gamma V(S_{t+1})\right]$, where $X_t = X^\pi(S_t)$. Actor and critic work to improve both the policy and value estimation.

The update of the actor network is done by using batch stochastic gradient:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \mathbb{E}[(\log X^\pi(S, \theta))V(S, X)] \tag{4.2}$$

We do not employ the t subscript notation as we work with batches of X and S. Therefore, as in Sutton and Barto (2018), we adopt a simplification in the notation, where X and S also represent batches of X and S. The gradient is calculated concerning the loss to minimize it: $\nabla_\theta L = \nabla_\theta \mathbb{E}[(\log X^\pi(S, \theta))V(S, X)]$, which is defined by the expected value of the cumulative reward (or in our problem application, the minimization of the costs employing the policy X^π.

The update of the critic is done using the stochastic gradient descent as in the DQN Equation (4.1). Notice that in Eq. (4.1), a temporal difference is used to calculate the loss (SAMUEL, 1959; SUTTON, 1988). This equation enables the method to estimate the state-action value without knowing the transition functions. The lack of dependence on the transition function exemplifies the key characteristic of model-free methods. It is important to mention that employing ANN for both the actor and the critic may give a good capacity to handle large action and large state spaces. Take, for example, the neural network used for image recognition, which can deal with images characterized by a huge amount of pixels.

While the discrete state space of problems such as SDLSP does not generate a problem (it can be used as input for both action and critic networks), the discrete action space must be considered with care since actor-critic agents handle large continuous action spaces describing the probability of taking action in a given state. Thus, we transform the

probabilities into action through a function called embedding. Throughout the training phase, the conversion from continuous to discrete space is accomplished by sampling based on the probability vector. This method enables the policy to explore the action space. Instead, the action selected corresponds to the maximum probability during the testing phase.

As we also explain in the subsequent section, the actor-critic has some advantages over the Deep Q-Network. The Policy Gradients can learn better policies, especially on continuous actions spaces (LILLICRAP et al., 2016). The problem is that the Policy Gradient relies on the sampled trajectories to estimate the gradient of the expected reward. Since each trajectory is a sequence of states, actions, and rewards sampled from the environment, it inherently carries noise. Due to that problem, we use a critical network to evaluate the decisions and guide the policy updates (see Algorithm 3).

The following sections explain two actor-critic methods, the A2C and the PPO.

4.4 Advantage Actor-Critic

In the Advantage Actor-Critic (A2C), the standard actor-critic architecture is modified by considering the difference $A(S, X) = Q(S, X) - V(S)$, called advantage. It measures the discrepancy between the state-action value of the given policy, $Q(S, X)$, and the state-value, $V(S)$ (MNIH et al., 2016). In other words, the advantage represents the degree to which the expected total discounted reinforcement is increased by acting X in state S relative to the action currently considered best. Instead of using an objective function that relies only on the policy and the Q-values as in Eq. (4.2), A2C uses the advantage in the formula:

$$\theta_{i+1} \leftarrow \theta_i + \alpha \nabla_\theta \mathbb{E}[\log X^\pi(S, \theta_i) A(S, X)] \tag{4.3}$$

This choice helps to reduce variance in the computation of the gradient (SCHULMAN et al., 2018). The algorithm uses a learned estimate of the value function as the baseline, which approximates the expected total discounted return starting from state S and following the policy, denoted as $X^\pi(S)$. This estimate of the value function leads to a lower variance in the Policy Gradient, which is used to update the policy.

Using the advantage function in the A2C algorithm is a way to overcome the limitations of the value-based methods in stochastic environments. The advantage function provides a more accurate estimate of the value of an action, allowing the algorithm to

converge to a better policy more quickly than if it were only using the value function. Additionally, after convergence, the policy can be extracted from the advantage function alone. The optimal policy for state variable S is any decision X that maximizes $A(S, X)$ (MNIH et al., 2016).

The pseudocode in Algorithm 3 describes the A2C training process. The algorithm starts by initializing the model parameters, θ_0. The algorithm then runs for a maximum number of episodes, N, and in each episode, the agent interacts with the environment and collects data. The agent starts in a given initial state, s_0, and at each time step, t, the agent calculates the policy probability, $P(x_t)$, for the current state using the policy function, X^π. The agent then selects an action, x_t, based on the policy probability and takes action in the environment. The agent observes the contribution and next state, s_{t+1}, and calculates the advantage function, A_t, using the current contribution, c_t, the expected future contribution, $V(s_{t+1})$, and the value function, $V(s_t)$.

To compute the gradient used in the policy update, we derive it from the loss function, which is composed of the sum of three distinct components:

1. **Actor Loss** (L_{Actor}): This is the part of the loss function responsible for updating the policy. It is computed using the Policy Gradient loss function, but instead of using the total discounted return, the advantage function is used. This loss calculation is written mathematically as:

$$L_{Actor} = -\mathbb{E}_t \left[\log(X^\pi(X_t|S_t, \theta))A(S_t, X_t)\right] \tag{4.4}$$

2. **Critic Loss** (L_{Critic}): This part of the loss function is designed to minimize the error in the value function estimate. It is typically calculated as the squared error between the predicted state-value function $V(S)$ and the actual state-value G. This calculation is represented as:

$$L_{Critic} = \mathbb{E}\left[(V(S) - G)^2\right] \tag{4.5}$$

   In the context of Reinforcement Learning, G often represents the "return" from a certain state, which is the sum of rewards received after that state, each discounted by a factor that depends on how far in the future the reward is received.

   The return G_t from a time step t is defined as:

$$G_t = C_{t+1} + \gamma C_{t+2} + \gamma^2 C_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k C_{t+k+1} \tag{4.6}$$

Here, $C_{t+1}, C_{t+2}, C_{t+3}, \ldots$ are contributions (rewards) received at time steps $t+1, t+2, t+3, \ldots$ respectively. The γ is a discount factor between 0 and 1, which determines the present value of future rewards: a reward received k time steps in the future is worth only γ^k times what it would be worth if received immediately. The return G captures how good it is to be in a certain state: the larger the return, the better the state. Usually, the agent's goal is to maximize the expected return from each state, and the return is used in the update rules of many Reinforcement Learning algorithms.

3. **Entropy Bonus** (B): Like PPO, A2C also includes an entropy bonus in the loss function to encourage exploration. This loss is typically computed as the entropy of the policy distribution:

$$B(\theta) = \mathbb{E}_t[H(\mu_\theta(S_t))] \tag{4.7}$$

The combined A2C loss function would then be:

$$L_{A2C} = L_{Actor} + c_1 L_{Critic} - c_2 B(\theta) \tag{4.8}$$

Here, $c_1 \in \mathbb{R}$ and $c_2 \in \mathbb{R}$ are coefficients that control the relative importance of the critic loss and the entropy bonus, respectively.

Algorithm 3 Advantage Actor-Critic (A2C) training

procedure A2C_TRAINING(N, γ)
  ▷ Number of training episodes: N
  ▷ Discount factor γ
  ▷ Regularization of the advantage λ
  ▷ Output: Trained parameters: θ_N
  Initialize the environment and get a random initial state: $s_0 \in \mathcal{S}_0$
  Initialize the policy and value networks with random weights: θ_0
  Initialize storage of contributions: $\mathbf{C^h}$
  for $i \in (0, N]$ **do**
    $s_0 \leftarrow \text{Random}(\mathcal{S}_0)$
    for t in (0, N] **do**
      ▷ Get the action probabilities for current state
      $P(x_t) \leftarrow X^\pi(s_t, \theta_i)$
      ▷ Select an action based on the policy
      $x_t \leftarrow$ Sample the action$(P(x_t))$
      ▷ Calculate the value function for the current state
      $V(s_t) \leftarrow V(s_t)$
      ▷ Take the action and observe the next state
      $s_{t+1} \leftarrow S^M(s_t, x_t)$
      ▷ Take the action and observe the contribution
      $c_t \leftarrow C(s_t, x_t)$
      ▷ Calculate the advantage
      $A_t \leftarrow c_t + \gamma V(s_{t+1}) - V(s_t)$
      ▷ Store the contribution
      $\mathbf{C^h} \leftarrow \mathbf{C^h} \cup c_t$
      ▷ Calculate the return using $\mathbf{C^h}$
      $G_t = \sum_{k=t}^{T} \gamma^{k-t} C_{k+1}$
      ▷ Calculate the loss of the actor
      $L_{Actor} = -\log(X^\pi(x_t|s_t, \theta_i)) \cdot A(s_t, x_t)$
      ▷ Calculate the loss of the critic
      $L_{Critic} = (V(s_t) - G_t)^2$
      ▷ Calculate the Advantage Actor-Critic loss:
      $L_{A2C} = L_{Actor} + c_1 \cdot L_{Critic} - c_2 \cdot H(X^\pi(s_t, \theta_i)$
      ▷ Update the policy and value function parameters as in:
      $\theta_{i+1} \leftarrow \theta_i - \alpha \nabla_\theta L_i$
  return θ_N

4.5 Proximal Policy Optimization

Proximal Policy Optimization (PPO) (SCHULMAN et al., 2017), as A2C, uses an actor-critic approach where the loss function is composed of three terms:

1. **Clipped Surrogate Objective** (L_{CLIP}): The surrogate objective serves as an approximate representation of the true objective function, and it's used for updating the policy in the PPO algorithm. It primarily utilizes the concept of the importance sampling ratio.

   This ratio compares the probabilities of executing an action under the current and previous policy. Denoted as $\rho_t(\theta)$ in Equation (4.9), this ratio calculates the probability of choosing a particular action X_t given a state S_t, based on the new policy parameterized by θ. This probability is then divided by the equivalent probability under the old policy, parameterized by θ_{old} as shown in:

   $$\rho_t(\theta) = \frac{X^{\pi}\left(X_t \mid S_t, \theta_i\right)}{X^{\pi}\left(X_t \mid S_t, \theta_{i-1}\right)}, \tag{4.9}$$

   The surrogate objective is used to construct a clipped version of the policy update, where the ratio $\rho_t(\theta)$ is clipped within a certain range to prevent overly large updates. The "clipped" term in the clipped surrogate objective refers to a clipping function applied to the importance sampling ratio. This function limits the update step to a smaller range in the policy space, as shown in the equation:

   $$\text{clip}(x, \epsilon) = \begin{cases} x + \epsilon, & \text{if } x < -\epsilon \\ x, & \text{if } -\epsilon \leq x \leq \epsilon \\ x - \epsilon, & \text{if } x > \epsilon \end{cases} \tag{4.10}$$

   The clipping function limits the ratio within a predefined range, defined by the hyperparameter ϵ representing the clip range (e.g., 0.1 or 0.2). By clipping the ratio, PPO ensures that the policy updates stay within a trust region, avoiding overly large updates that might lead to instability in the learning process. The L_{CLIP} function is then defined as follows:

   $$L_{CLIP}(S, X, \theta) = \mathbb{E}_t\left[\min\left(\rho(\theta)A(S, X), \text{clip}(\rho(\theta), 1 - \epsilon, 1 + \epsilon)A(S, X)\right)\right], \tag{4.11}$$

   where $\mathbb{E}_t$ denotes the expectation operator with respect to the distribution at time step t, which represents the expectation taken over different possible values of the

action.

In summary, the clipped surrogate objective $L_{CLIP}(S, X, \theta)$ is a function that combines the importance sampling ratio, the advantage function, and the clipping function and is used as the objective function for updating the policy parameters in the PPO method.

2. **Value Function Loss** (L_V): This component aims to minimize the error in the value function estimate (critic). It is typically calculated as the squared error between the predicted state-value function $V(S)$ and the actual state-value $L_{VF} = \mathbb{E}_t[(V(S) - G)^2]$. Here, $V(S)$ is the predicted value function for the state S.

3. **Entropy Bonus** (B): An entropy bonus is incorporated into the loss function to promote exploration and deter premature convergence towards suboptimal policies. The entropy of the policy X^π distribution, which represents the probability of each possible action i according to the policy X^π, is computed as $B(\theta) = \mathbb{E}t[H(X^\pi(\theta))]$. Here, $H(X^\pi(\theta)) = -\sum iq_i \log(q_i)$ represents the entropy calculation. Using logarithm in the probabilities, a common practice in RL algorithms such as entropy bonuses, transforms probabilities into log probabilities. This transformation not only simplifies mathematical computations but also helps avoid numerical instability, enhancing the learning process's robustness.

The combined PPO loss function is:

$$L_{PPO} = \mathbb{E}_t[L_{CLIP}(\theta) - c_1 L_V + c_2 B(\theta)] \tag{4.12}$$

Here, as in the A2C, c_1 and c_2 are coefficients that control the relative importance of the value function loss and the entropy bonus, respectively.

The algorithm runs for a specified number of episodes, N, during which data is collected for each episode. The data collected in each episode is stored in a buffer, **D**. After collecting data, the policy parameters are updated using the data and the old policy parameters. The new policy parameters are then clipped to prevent them from deviating too much from the old policy parameters. Finally, the old policy parameters are updated with the new policy parameters, and the algorithm continues running for the next episode. After all episodes have been completed, the final policy parameters are returned.

In PPO, the objective function is defined as the ratio of the new policy's probability to the old policy's probability for the action taken. The algorithm then maximizes

this objective function using gradient ascent[5], with the constraint that the new policy's probability must be within a certain range of the old policy's probability. This procedure is adopted to prevent the policy from changing too much from one iteration to the next, which may cause instability in the learning process. PPO also uses a trust region optimization method, where a hyperparameter bounds the step size of the update, called the clipping parameter, to ensure the new policy does not deviate too far from the old one. PPO is a powerful algorithm that can converge to good policies quickly, even in complex environments. It is particularly useful when dealing with high-dimensional and continuous action spaces (HÄMÄLÄINEN et al., 2018).

[5] In the case of PPO, the objective is to maximize the expected cumulative reward. This objective is often formulated as maximizing an objective function. Hence, the term "gradient ascent" is used. However, in the implementation, this is typically achieved by minimizing the negative of the objective function, which is equivalent to maximizing the original function. This procedure is where "gradient descent" comes into play. The backpropagation algorithm works by computing the gradient of the loss function concerning the network's weights and then adjusting the weights in the direction that minimizes the loss.

Algorithm 4 Proximal Policy Optimization (PPO)

procedure PPO(N, K, ϵ)
    ▷ Number of training episodes: N
    ▷ Number of policy updates: K
    ▷ Clipping range ϵ
    ▷ Output: Trained policy parameters θ_N
    Initialize the policy and value networks with random weights: θ_0
    Initialize the environment and get a random initial state: $s_0 \in \mathcal{S}_0$
    for $i \in (0, N]$ **do**
        Collect a set of trajectories $\mathbf{D}$ using the current policy
        ▷ Compute the rewards-to-go for each trajectory in $\mathbf{D}$:
        $C_t = \sum_{i=t}^{T} \gamma^{i-t} C_i$
        ▷ Compute the advantage estimates for each state in each trajectory in $\mathbf{D}$:
        $A_t = c_t - V(s_t)$
        ▷ Update the policy by maximizing the PPO clip objective:
        for $j \in (0, K]$ **do**
            ▷ Compute the loss with respect to clipped surrogate objective:
            $L_{CLIP} = \mathbb{E}_t \left[\min \left(\rho_t(\theta_i) A(s_t, x_t), \text{clip}(\rho(\theta_i), 1-\epsilon, 1+\epsilon) A(s_t, x_t) \right) \right]$
            ▷ Compute the loss with respect to the value function:
            $L_{VF} = \mathbb{E}_t[(V(s_t) - G)^2]$
            ▷ Compute the entropy bonus:
            $B(\theta_i) = \mathbb{E}_t[H(X^\pi(\theta_i)]$, where $H(X^\pi(\theta_i)) = -\sum_i q_i \log(q_i)$
            ▷ Sum all the three losses to calculate the PPO loss:
            $L_{PPO} = \mathbb{E}_t[L_{CLIP}(\theta_i) - c_1 L_{VF} + c_2 B(\theta_i)]$
            ▷ Update the policy network parameters θ_i using a gradient-based optimizer:
            $\theta_{i+1} \leftarrow \theta_i + \alpha \nabla_{\theta_i} L_{PPO}$
    return θ_N

5 ACTIVE SINGLE-ASSET TRADING PROBLEM

The world of finance is filled with challenges that require calculated decision-making. One key problem is the Active Single-Asset Trading Problem (ASATP). The ASATP revolves around making optimal decisions on when to buy, sell, or hold a single financial asset over a given time horizon to maximize returns. These decisions hinge on understanding the asset's price dynamics and discerning patterns, which are affected by various external factors, such as market trends, geopolitical events, and economic indicators.

Such trading decisions mirror more extensive decision-making problems seen across different domains. For instance, in any other type of market, such as the energy market, one must decide the optimal time to buy energy assets. While the contexts differ, the core principle remains: making optimal decisions based on available historical information, where actions may not influence the market price. Recall from Section 3.2 the problem constraints established for the ASATP. The constraints make this decision problem more specific, requiring specific solutions. Those solutions have limitations, and the algorithms described here will not perfectly capture the intricacies of the market, nor can they predict unforeseeable events. Furthermore, while our approach leverages Supervised Learning and has shown promising results against RL methods, it is rooted in historical data. This characteristic results in the possibility that the model's effectiveness decreases when faced with unprecedented market scenarios.

This chapter is organized into two main sections. The first section, Section 5.1, presents the methods specifically developed to address the ASATP. The second section, Section 5.2, discusses the results of applying these methods. The methods explained in this chapter are compared against the Reinforcement Learning methods detailed in Chapter 4.

In the forthcoming Section 5.1, we introduce two distinct methods explicitly designed to address the ASATP. Initially, we examine the Recurrent Reinforcement Learning (RRL) methodology, which was first introduced by Moody and Wu (1997) and employed as an

RL method purposed to solve trading problems within the scope of portfolio management. This method has since been adapted for single asset trading. Section 5.1 also presents our innovative method, the Residual Network Long Short-Term Memory Actor (RSLSTM-A) (FELIZARDO et al., 2022a), which primarily utilizes a Supervised Learning approach to address the decision problem. Within this, we employ a modified version of the ResNet architecture, functioning as a time series classifier for decision problems.

Following this, we pivot our attention to the experimental results of our research in Section 5.2. We briefly overview our experimental setups and analyze the pertinent market data and performance metrics. After that, we present the performance of our proposed RSLSTM-A method. Our experiments investigate two fundamental problem setups: those inclusive and exclusive of transaction costs. The critical influence of transaction costs on policy formulation highlights its significant role within this context.

Our investigation extends beyond the exploration of our proposed RSLSTM-A algorithm. It encompasses a comparative analysis of the current set of solutions available for the ASATP, including both full RL algorithms (DQN, A2C, and PPO) and the contextual bandit solution Bootstrapped Thompson Sampling (BTS). We intend to comprehensively review existing techniques, emphasizing those already implemented within trading scenarios. As we explore these various methodologies, our focus remains on RL algorithms that have been comprehensively tested within the trading context, thus ensuring an accurate analysis.

Our contributions in this problem instance are two-fold: First, we show pieces of evidence that a supervised approach can produce comparable or even better outcomes than RL methods for the active trading problem. Second, we introduce and modify the residual neural network (a.k.a. Residual Network, ResNet) architecture as a time series classifier for decision problems. We also illustrate the different features created by ResNet in a graphical representation of the convolutional layer outputs, exploring how this may help our approach outperform the RL counterparts in our experiments. Given the nature of our supervised approach, ResNet architecture is an ideal choice, as it is considered one of the most effective time series classifiers (FAWAZ et al., 2019).

5.1 Active single asset trading methods

The ASATP can be tackled using a variety of methods. We begin this section by explaining the RRL method proposed by Moody and Wu (1997) that is, in some sense, a

Policy Gradient method.

We also employ the RL methods scribbled before to solve the problem of ASATP as we show in 5.2. RL methods, such as the ones employed in previous works (DENG et al., 2017; ABOUSSALAH; LEE, 2020; ALMAHDI; YANG, 2019), and in this work as well, were originally built to solve problems on which the actions affect future states and rewards. While they may still be applicable to the ASATP, they may need to be more optimal, as they may introduce noise in the final policy by attempting to account for non-existent relations in the transition functions. We present evidence to suggest that a contextual bandit (SUTTON; BARTO, 2018) or even the Supervised Learning method proposed by us, explained in Section 5.1.2, can be more effective than RL for active trading problems.

5.1.1 Recurrent Reinforcement Learning

The Recurrent Reinforcement Learning (RRL) training algorithm, introduced by Moody and Wu (1997), is a learning approach to train a linear model to make trading decisions. RRL is considered one of the pioneering methods that employ RL to address the ASATP and the portfolio management problem. The RRL method computes the partial derivatives of the utility function to update the policy function parameters that maximize the utility. Algorithm 5 is specifically designed to maximize the utility function, which can be the contributions of a portfolio of assets or a single asset. The utility function in Recurrent Reinforcement Learning is the Sharpe Ratio (SR) at time t, computed in terms of the distribution moments of the returns, A and B. Specifically, the SR is given by the following formula:

$$SR_t = \frac{A_t}{K_t \left(B_t - A_t^2\right)^{1/2}} \tag{5.1}$$

Here, A_t and B_t are calculated as:

$$A_t = \frac{1}{t}\sum_{i=1}^{t} R_i \ , \quad B_t = \frac{1}{t}\sum_{i=1}^{t} R_i^2 \ , \quad K_t = \left(\frac{t}{t-1}\right)^{1/2} \tag{5.2}$$

A_t and B_t are defined as exponential moving estimates of the first and second moments of the variable R_t, the price return. The first moment of a random variable is its expected value or mean. It provides a measure of the central tendency of the distribution. The second moment of a random variable is its variance. It provides a measure of the dispersion or spread of the distribution. In the context of returns R_t, the first moment would be the expected return, which measures the returns' central tendency. The second moment

would be the returns' variance, which measures the volatility or risk associated with the returns. K_t is a scaling factor used in calculating the Sharpe Ratio. The purpose of this scaling factor is to normalize the Sharpe Ratio over different time periods.

Given the Sharpe Ratio's formulation in terms of return moments, we calculate the gradient of the policy function as follows:

$$\theta_{i+1} = \theta_i + \rho \frac{\mathrm{d}SR_t(\theta_i)}{\mathrm{d}\theta_i} \tag{5.3}$$

Here, we compute the gradient of the Sharpe Ratio by:

$$\frac{\mathrm{d}SR_t(\theta_i)}{\mathrm{d}\theta_i} = \frac{1}{t}\sum_{i=1}^{t}\left\{\frac{B_t - A_t R_i}{K_t\left(B_t - A_n^2\right)^{3/2}}\right\}\left\{\frac{\mathrm{d}R_i}{\mathrm{d}X^{\pi}(S_i)}\frac{\mathrm{d}X^{\pi}(S_i)}{\mathrm{d}\theta_i} + \frac{\mathrm{d}R_i}{\mathrm{d}X^{\pi}(S_{i-1})}\frac{\mathrm{d}X^{\pi}(S_{i-1})}{\mathrm{d}\theta_i}\right\} \tag{5.4}$$

The recurrent term in the Recurrent Reinforcement Learning algorithm stems from using previous and current actions to compute the Sharpe Ratio's gradient.

The Algorithm 5 outputs the trained parameters of the linear model: θ_N. It initializes the linear model parameters: θ_0, the replay memory: $\mathbf{M}^h$, the contribution in validation memory: $\mathbf{C}^{h,V}$ employed to have a validation metric (out-of-sample), the history of policy decisions: $\mathbf{X}^h$ and the initial state: s_0. The algorithm then executes N training episodes. The algorithm makes decisions within each episode over a sequence of T steps. In each episode, the environment is reset to the initial state, and the algorithm composes the state and selects the decision. The decision is taken, and the agent receives the contribution and next state. The gradient of the contribution for the model parameters is then calculated, and the model parameters are updated using gradient descent. Finally, it validates the model, checks if the model is the best so far, and saves the best model parameters. The algorithm returns the best model parameters.

Algorithm 5 Recurrent Reinforcement Learning (RRL) training

procedure RRL($S^{M,T}, M, , \alpha\delta, \mu, T, N$)
    ▷ Transition function for training: S^M
    ▷ Initial state variables: s_0
    ▷ Initial contribution in training: c_0
    ▷ Initial contribution in validation: c_0^V
    ▷ Number of recurrent steps: M
    ▷ Commission rate: δ
    ▷ Scaling factor for decision: μ
    ▷ Learning rate: α
    ▷ Number of episodes: T
    ▷ Number of training episodes: N
    ▷ Output: Trained parameters: θ_N
    Initialize the model parameters: θ_0
    Initialize the environment and get a random initial state: $s_0 \in \mathcal{S}_0$
    for $i \in (0, N]$ **do**
        for $t \in (0, T]$ **do**
            ▷ Reset the environment
            $s_0 \leftarrow \text{Random}(\mathcal{S}_0)$
            for $j \in (1, T - M - 1]$ **do**
                ▷ Compose the state and select decision
                $s_j \leftarrow 1|s_j|X^\pi(s_{j-1}, \theta_{i-1})$
                $j_j \leftarrow X^\pi(j_t, \theta_i)$
                ▷ Take the decision and observe the contribution and next state
                $s_{j+1} \leftarrow S^M(s_j, x_j, r_j)$
            ▷ The transition function return the Sharpe Ratio (contribution)
            $c_t \leftarrow SR(r_t, ..., r_{t-M}) = C(s_t, x_t)$
            ▷ Calculate the gradient to update the weights
            $dCd\theta_i \leftarrow f_{dSRd\theta}(R, X^\pi(\theta_i), \mu, \delta)$
            ▷ Update the model parameters using gradient descent
            $\theta_{i+1} \leftarrow \theta_i - \alpha \times dC^T d\theta_i$
        ▷ Validate the model in the validation environment with validation data
        $\theta_{i+1} = \begin{cases} \theta_{i+1}, C_t^V \leftarrow C_{max}^V \text{ if } \sum_{t=0}^T C_t^V > C_{max}^V \\ \theta_i \text{ if } \sum_{t=0}^T C_t^V < C_{max}^V \end{cases}$
    return θ_N

5.1.2 Residual Network Long Short-Term Memory Actor

We propose the Residual Network Long Short-Term Memory Actor (RSLSTM-A) (FELIZARDO et al., 2022a) to solve the ASATP, a method that primarily employs the ResNet architecture introduced by He et al. (2016). Our objective is to ascertain, at every time step, whether a short (-1) or long (1) position is most suitable based solely on the previous price returns. To achieve this, we use a custom variant of the ResNet architecture to classify the time series of past price returns into a short or long position. The ResNet architecture is especially adept for our task due to its capacity to accurately capture nonlinear relationships without making any presumptions about the nature of the underlying time series. Additionally, studies like Fawaz et al. (2019) and Urbinate, Felizardo and Del-Moral-Hernandez (2022) have highlighted that the ResNet or CNN-based architectures can surpass other deep learning techniques in univariate as well as multivariate time series classification tasks. Also, in Felizardo et al. (2019), we present some alternative ANNs architecture experiments to deal with time-series analysis that helped us choose the employed architecture in this thesis and Felizardo et al. (2022a). These results reinforce our choice to implement this specific architecture in our proposed method.

The ResNet architecture was initially developed to address the degradation in training accuracy caused by the vanishing or exploding gradient, a common problem in deep learning architectures. It achieves this by fitting a residual mapping, $\mathcal{F}(S)$, utilizing a block of stacked convolutional layers, as opposed to directly learning the underlying mapping, $\mathcal{H}(S)$. The underlying mapping, $\mathcal{H}(S)$, represents the unknown function that, in our case, gives the optimal decisions for each state encoded by the input S, which represents the market state. The residual mapping is defined as $\mathcal{F}(S) = \mathcal{H}(S) - S$ and is achieved through the use of shortcut connections, which allow for the direct use of the input S with the output of the stacked layers. This approach has been shown to reduce the variation of learned features, resulting in improved accuracy and performance.

Additionally, we utilize the Rectified Linear Unit (ReLU) activation function within each convolutional block. The ReLU activation function, defined as $ReLU(x) = max(0, x)$, is a popular choice because it is computationally more efficient than sigmoid functions. This activation function ensures that the ResNet architecture can quickly and effectively adapt to the input data, further improving its performance. In this work, we utilize the ResNet architecture to approximate the policy function $X^{\pi}(S_t)$ by indirectly approximating the underlying function $\mathcal{H}(S)$. A policy function (X^{π}) provides the decision

$X_t \in [-1, 1]$ given a state $S_t = (r_{t-1}, ..., r_{t-M})$.

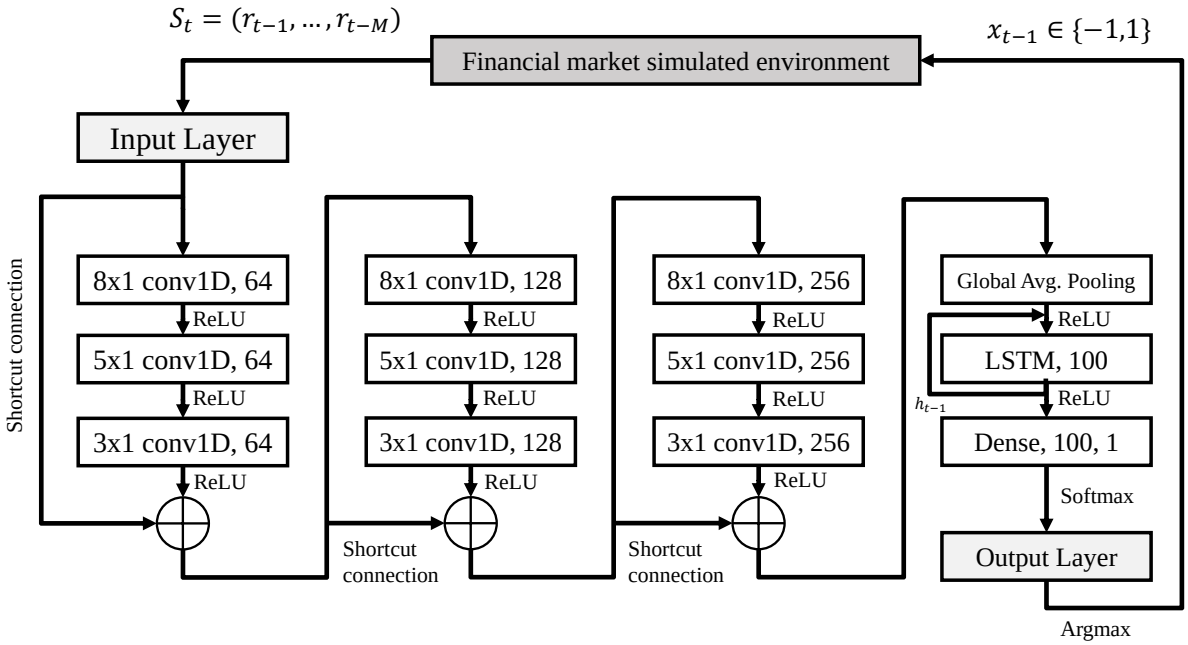
The convolutional layers of the ResNet model approximate the underlying function $\mathcal{H}(S)$ by the residual function $\mathcal{F}(S)$. We aim to maximize the next agent's contribution (or the agent's return, expressed by $C(S_t, X_t)$) by reducing the loss between decision x_t provided by the ResNet approximate policy X^π and the best decision X^* (the one that maximizes the immediate contribution) given by the signal of the exogenous information W_t. A Supervised Learning algorithm uses labeled data to update the model parameters (i.e., ANN internal weights). The label is that the state's best decision can be defined by the action that maximizes the cumulative contribution in a certain future window. We update the artificial neural network weights using a loss function calculated as the difference between the model's output and the actual labels (i.e., the best decision) using a backpropagation algorithm. We apply the Adam optimizer with a learning rate of 0.01, which adapts over time, and use a batch of size 64 to correct our model parameters through training for 150 epochs. In the output layer, we have a softmax layer that generates a vector of probabilities of success (success, meaning the positive agent asset price return), each probability associated with a decision.

Upon learning the function $\mathcal{H}(S_t)$, the model advances to the second phase of online execution (evaluation or test). This phase evaluates the model's generalization capabilities to unseen states and its efficacy in real-world testing scenarios. During this phase, the model is presented with a new state S_t at each time step t, from which it must decide. Moreover, in the context of transaction costs, we assess different trading frequencies by training the agent based on the cumulative contributions over a future window of size M^f rather than solely considering the immediate next decision trader return R^T_{t+1}. Adopting a reduced trading frequency and considering the cumulative future contribution over a specified window can lower transaction costs and enhance overall performance. In our implementation, we set an equal value for the time gap between decisions and the future window size M^f. While the future window and gap between each trade can be independent, we align them for simplicity.

Figure 4 illustrates the RSLSTM-A architecture employed to solve the ASATP. This structure is adapted from the original design by Fawaz et al. (2019), but several amendments have been incorporated to make it fit for the ASATP. We have introduced a batch normalization layer, ReLU activation function, and a max-pooling layer post each convolution, enhancing the model's feature extraction capabilities. Each convolutional block consists of three convolution, batch normalization, and max-pooling sequences. The number of channels gradually expands across these sequences (64, 128, and 256), allowing for

the extraction of an increasing number of features from the time series. Our model employs a long short-term memory artificial neural network layer instead of the average global pooling layer in the original architecture. This adjustment facilitates the model's ability to consider past actions and observations. The utility of long short-term memory ANNs in preserving long-term temporal sequence data and bolstering performance in time series classification and forecasting tasks has been established in earlier studies such as Choi, Ryu and Kim (2018), He et al. (2019). The long short-term memory layer is designed to remember prior data, going beyond the last M features, even when the size of the past asset price changes under observation is fixed at M. The long short-term memory layer's hidden feature h_{t-1} is supplied as an input for the succeeding Long Short-Term Memory (LSTM) feature generation. This process equips the model with the knowledge of decisions and observations from the previous time step.

Figure 4: RSLSTM-A architecture and financial market online execution (evaluation) using actions, X_{t-1}, and receiving informational state, S_t, of the environment



Source: Felizardo et al. (2022a).

The RSLSTM-A training and testing procedures are described in detail in Algorithms 6 and 7, respectively. During training, RSLSTM-A calculates the probability vector $P(X)$ to determine the method decision $X = \arg\max_X(P(X))$ at each time step. The neural network's weights are updated based on the model's outputs, with the loss calculated using binary cross-entropy (HE et al., 2016) between the best action probability vector and the

RSLSTM-A probability vector. The backpropagation procedure then updates the internal weights of the model, effectively defining the action policy for each state. The data is divided into training, validation, and test sets to evaluate the model's performance. In training, we update the model parameters when achieving the best performance evaluated on the validation set after each episode. The parameters with the best performance are stored. The function $f_{eval.}$ calculates the total loss on the validation set. Finally, the best parameters θ_i obtained from the RSLSTM-A interactions with the environment during training are used to evaluate the test data set. At each time step during testing, the transition function $S^M(S_t, X_t)$ outputs the subsequent state S_{t+1}. This response is used to update the model's internal state and determine the subsequent decision to be made by RSLSTM-A. The testing algorithm also records the cumulative contribution of the decisions made by RSLSTM-A, which is used to evaluate the model's performance on the test set.

Algorithm 6 Resnet LSTM actor (RSLSTM-A) training

procedure RSLSTM-A TRAINING($\mathbf{S}^h, N$)
  ▷ Storage of all observable states in training $\mathbf{S}^h$
  ▷ Number of training episodes: N
  ▷ Output: Trained parameters: θ_N
  Initialize the model parameters: θ_0
  Initialize storage of decisions: $\mathbf{X}^h$
  Initialize storage of best decisions: $\mathbf{X}^{*,h}$
  ▷ Define the function to get the best decision: f
  $L_0^{val} \leftarrow f_{eval.}(\theta_0)$ ▷ Initialize Loss
  for $i \in (0, N]$ **do**
    for s_t in $\mathbf{S}^h$ **do**
      ▷ Calculate the probability of each decision for current state
      $P(x_t) \leftarrow X^{\pi}(s_t, \theta_i)$
      ▷ Select the decision with highest probability
      $x_t \leftarrow$ Select action$(P(x_t))$
      ▷ Store the selected decision
      $\mathbf{X}^h \leftarrow \mathbf{X}^h \cup x_t$
      ▷ Get the next state from the environment
      $s_{t+1} \leftarrow S^M(s_t, x_t)$
      ▷ Get the best decision based on the actual state and the price return
      $x_t^* \leftarrow S^M(s_t, r_t)$
      ▷ Store the best decision
      $\mathbf{X}^{*,h} \leftarrow \mathbf{X}^{*,h} \cup x_t^*$
    ▷ Calculate the binary cross-entropy loss
    $L_i = -\left(\mathbf{X}^{*,h} \log(P(\mathbf{X}^h)) + ([1, ..., 1] - \mathbf{X}^{*,h}) \log(1 - P(\mathbf{X}^h))\right)$
    ▷ Update the model parameters using backpropagation
    $\theta_{i+1} \leftarrow$ backpropagate(θ_i, L_i)
    ▷ Validate the model in the validation environment with validation data
    $\theta_{i+1} = \begin{cases} \theta_{i+1}, C_t^V \leftarrow C_{max}^V \text{ if } \sum_{t=0}^{T} C_t^V > C_{max}^V \\ \theta_i \text{ if } \sum_{t=0}^{T} C_t^V < C_{max}^V \end{cases}$
  return θ_N

In order to assess the performance of the RSLSTM-A model and its ability to generalize to new data, we use an out-of-sample dataset. This approach is a standard practice

in Supervised Learning methodologies and is essential for determining the model's generalization capabilities and ensuring that overfitting to the in-sample (training) data has not occurred. The pseudocode in Algorithm 7 illustrates the interaction of our model with a simulated market environment, where we measure the accumulated profit obtained by our model. By utilizing the model's decisions and considering the changes in asset prices, we calculate various financial metrics, such as profits, that are used to evaluate the performance of our model. These metrics are further described in the experimental section of this text. It is worth noting that our out-of-sample evaluation serves as a crucial validation step for our model and ensures that it can make sound decisions based on unseen data.

Algorithm 7 Resnet LSTM actor (RSLSTM-A) evaluation

procedure RSLSTM-A TESTING(θ_N,N)
  Initialize storage of contributions: $\mathbf{C^h}$
  Initialize the environment and get a random initial state: $s_0 \in \mathcal{S}_0$
  for t in (0, N] **do**
    ▷ Get the next decision from the model every i steps
    if $i = 1$ **then**
      $x_t \leftarrow X^\pi(s_t, \theta_N)$
    if $x_t! = x_{t-1}$ **then**
      $i = M^f$ ▷ M^f is future window model hyperparameter
    ▷ i here acts like a counter to block the agent trading trading
    $i = i - 1$
    ▷ Get the next state from the environment
    $s_{t+1} \leftarrow S^M(s_t, x_t)$
    ▷ Add the current contribution to the storage
    $\mathbf{C^h} \leftarrow \mathbf{C^h} \cup c_t$
  ▷ Calculate evaluation metrics
  $\langle SR, AR, ACR \rangle \leftarrow f_{val.}(\mathbf{C^h})$ ▷ SR is the Sharpe Ratio, AR is the average return, ACR is the area under the AR curve.
  ▷ Return the evaluation metrics
  return $\langle SR, AR, ACR \rangle$

5.2 Experimental results for the Active Single-Asset Trading Problem

This section begins with a concise outline of the experiment setups, followed by an exploration of market data and performance metrics. We then showcase the results of our proposed RSLSTM-A method, comparing it with state-of-the-art RL techniques. This experimental research investigates two primary problem setups: those with and without transaction costs. It is important to note that including transaction costs can significantly impact the policies generated since they may limit position changes.

This thesis undertakes a comparative analysis of established RL algorithms, the contextual bandit solution Bootstrapped Thompson Sampling (BTS), and our proposed RSLSTM-A algorithm, specifically in active trading. Despite the continual emergence of novel RL methodologies within the machine learning arena, our research scope remains anchored to algorithms that have already undergone trial in trading scenarios. Past implementations of RL in trading contexts (such as portfolio management and single-asset trading) (ALMAHDI; YANG, 2019; ABOUSSALAH; LEE, 2020; PARK; SIM; CHOI, 2020; ZARKIAS et al., 2019; PONOMAREV; OSELEDETS; CICHOCKI, 2019; LI; ZHENG; ZHENG, 2019), have predominantly leveraged Recurrent Reinforcement Learning (RRL) (MOODY; WU, 1997), Deep Q-Network (DQN) (MNIH et al., 2013), and Asynchronous Advantage Actor-Critic (BAIRD, 1993; MNIH et al., 2016)[1]. For our investigation, we employed the non-asynchronous variant of the Asynchronous Advantage Actor-Critic, the Advantage Actor-Critic (A2C). The intention behind this comprehensive examination was to encompass an inclusive range of RL techniques that are presently mainstream, thereby providing a robust comparative analysis.

In this work, we conduct our experiments using two distinct hardware configurations. The primary setup consists of an i7-7700HQ CPU clocked at 2.80GHz, accompanied by a GTX 1060 (6GB) GPU and 16GB of RAM. For more extensive and demanding experiments, we utilize a system equipped with an AMD Ryzen 5 5600X 6-Core Processor operating at 3.70 GHz, an RTX 3060 (12GB) GPU, and 32GB of RAM. The software aspect of our experimentation relies on Python language (ROSSUM; DRAKE, 2009), supplemented by a suite of libraries including Keras (CHOLLET et al., 2015), Tensorflow (ABADI et al., 2015), Pytorch (PASZKE et al., 2019), Stable baselines3 (RAFFIN et al., 2021), and Scikit-learn (PEDREGOSA et al., 2011).

To ensure transparency and reproducibility of our experiments, we have made all

[1] For an extensive discussion on this, kindly refer to chapter 2.2

the codes available in our GitHub repository, accessible via the following link: ⟨https://github.com/leokan92/Contextual-bandit-ResNet-trading⟩.

5.2.1 Market data

Our experimental models were evaluated employing cryptocurrency data, a decision driven by two primary considerations:

- **Availability of rich, high-frequency time series data**: Unlike traditional financial markets, cryptocurrency trading operates round-the-clock, seven days a week, offering an abundance of free, high-frequency time series data. This uninterrupted operation overcomes the data limitations associated with weekends and end-of-market events.
- **Influence of speculative nature and human behavior**: Cryptocurrency's highly speculative nature makes its asset prices particularly susceptible to fluctuations based on human behavior and market sentiment. This characteristic contrasts with assets like bonds or commodities, whose prices are more dictated by economic factors or supply-demand dynamics. As such, cryptocurrencies offer a more pronounced illustration of the impact of human behavior on asset prices. Patterns in the asset time series may capture this human behavior, potentially offering richer information for decision-making and improving the chances of outperforming simple Buy and Hold strategies.

The market simulations for our experiments were conducted on an hourly basis. We initiated the experimental tests with six selected assets: Bitcoin (BTC), Ethereum (ETH), Litecoin (LTC), Dash (DASH), Next (NXT), and Monero (XMR). For more comprehensive experimentation, we included an additional four assets - Ripple (XRP), New Economy Movement (XEM), Lisk (LSK), and Ethereum Classic (ETC) - along with the initial six. In trading research, acquiring standardized, high-frequency public datasets for cryptocurrencies and other assets remains a substantial hurdle (NASSIRTOUSSI et al., 2014; PINEAU et al., 2021). Nonetheless, we managed to source the necessary data directly from various web portals or APIs, with the search for a reliable source posing a certain challenge. All of our data was procured from the Poloniex broker API. Each asset's dataset contains approximately 29,600 data points, encompassing the period from September 1, 2017, to November 20, 2020.

Our dataset was partitioned into training, validation, and test sets, with proportions allocated as 0.8, 0.1, and 0.1, respectively. Adhering to conventional procedures in other Reinforcement Learning (RL) trading studies, we employed the validation set to fine-tune the hyperparameters for our RL and Supervised Learning models. The configurations that delivered the highest performance were then selected. To create the training, validation, and test datasets for our RSLSTM-A model, we used a window of past price changes, denoted as $(r_{t-1}, ..., r_{t-M})$, as the input. We assigned the target as -1 if the next return, represented as R_t, is negative and $+1$ if one of the succeeding returns is positive. In transaction costs scenarios, we marked negative sums of M^f future price changes with a label of -1 and positive sums with a label of $+1$. As for the RL methods, an environment simulator leverages the price returns time series to generate the state $S_t = r_{t-1}, ..., r_{t-M}$ and the contribution C_t at each time step within the decision process. The RL agents learn in the try-and-error process as described in Section 4.

5.2.2 Performance metrics

First, we define two of the most commonly used performance metrics: Annualized Returns (AR) and Sharpe Ratio (SR). The annualized asset price return is calculated as follows:

$$AR = (\mathbb{E}(r_t, ..., r_0) + 1)^{T_D} - 1, \tag{5.5}$$

where $\mathbb{E}(r_t, ..., r_0)$ represents the expected daily asset price return rate of the tested method, and T_D is the number of trading days in a period.

The Sharpe Ratio can also be calculated by considering the risk-free ratio to first find the expected annualized mean excess return, $\mathbb{E}((r_t, ..., r_0) - r^f)$, and then dividing it by the standard deviation, $std(r_t, ..., r_0)$, of the daily return:

$$SR = \frac{\mathbb{E}((r_t, ..., r_0) - r^f)}{std(r_t, ..., r_0)}, \tag{5.6}$$

where r^f is the risk-free daily rate, which we define as 0.01%.

To further compare the performance of algorithms, we also consider the total area under the Accumulated Agent Asset Price Return (ACR). This metric is calculated by summing the daily returns as:

$$ACR = \sum_{t=0}^{T} [r_1, r_1 + r_2, r_1 + r_2 + r_3, ..., r_1 + r_2 + r_3 + ... + r_T] \tag{5.7}$$

Applying the cumulative asset price return curve as a metric has several critical implications for the trader. Primarily, it allows for a more accurate estimation of the potential profit that a trader could realize upon liquidating their assets at any given time point. Essentially, it reflects the accumulated profit or loss over a period, presenting a comprehensive view of an investment's performance over time. This characteristic makes it particularly useful in scenarios where trading is not continuous or when liquidation is considered before the end of the trading period. By providing a cumulative view of returns, the cumulative asset price return curve metric offers valuable insights into the profitability of a strategy across different points in time, thus supporting more informed decision-making.

To make comparative assessments of the algorithms in our study, we utilized a relative metric encompassing Accumulated Agent Asset Price Return (ACR), Sharpe Ratio (SR), and Annualized Return (AR) values. These could assume both positive and negative values. In order to standardize these into positive values, we established a baseline value, facilitating an accurate reflection of the performance of the different algorithms. The baseline value was determined by taking 1.5 times the minimum value found among the models in our study, including B&H, Recurrent Reinforcement Learning (RRL), Deep Q-Network (DQN), Advantage Actor-Critic (A2C), Bootstrapped Thompson Sampling (BTS), RSLSTM-A, and Proximal Policy Optimization (PPO) when applicable. This choice prevented any division by zero scenarios and helped minimize distortion in the relative metric. Selecting an excessively high baseline value would cause all relative values to appear unduly small, whereas an overly low baseline would inflate the relative values. Therefore, to establish a balanced and representative baseline, we multiplied the minimum value by 1.5, and then added this result to all other values. This procedure served to maintain the relative relationships among the values while also converting them all into positive numbers. In doing so, we ensure that our relative metric offers an accurate depiction of the performance of the algorithms, mitigating any skewness caused by negative values.

To provide a comprehensive evaluation of the performance of each algorithm, we employ a multi-metric ranking system was devised. For every performance metric (ACR, SR, AR), each algorithm was ranked. The average of these ranks was then computed to serve as a composite comparative metric. This method of ranking has several advantageous

implications. Firstly, it allows for the holistic evaluation of algorithms based on multiple performance measures, reducing the bias that may occur if only a single metric is considered. Each metric provides different insights into the algorithm's performance, and by considering them all, we ensure that the assessment captures a wide range of performance aspects. Secondly, using an average rank smoothens potential anomalies or outliers in individual experimental results. Thus, it presents a more balanced and robust measure of the overall performance of the algorithms. Lastly, to statistically differentiate the average ranks, we incorporated the Wilcoxon signed-rank test, a non-parametric statistical hypothesis test. Applying the Wilcoxon signed-rank test offers a pairwise comparison of ranks, thus evaluating whether the differences in ranks are statistically significant. By applying this combined approach, we offer a detailed, balanced, and statistically validated comparison of the algorithms, ensuring that our assessments are robust and representative. This comparison method aids in discerning subtle differences in performance that could be pivotal for algorithm selection in real-world applications.

5.2.3 Experimental results discussion

Table 3 presents the consolidated results for ACR, SR, and AR for all models and assets without transaction costs. The best results are highlighted in bold. The results indicate that RSLSTM-A outperformed other RL algorithms, achieving the highest overall performance across all metrics (ACR, SR, and AR) for four assets (BTC, DASH, LTC, and NXT). Additionally, RSLSTM-A attained the second-best ACR performance for the remaining assets (ETH and XMR). These findings suggest that RSLSTM-A is a suitable technique for investors who wish to disinvest at any point, as it provides higher agent asset price returns regardless of the stopping point. Furthermore, the results indicate that RSLSTM-A is the best algorithm for risk management and profit maximization, as evidenced by its high SR, indicating the trading system's stability under high volatility situations. Additionally, RSLSTM-A had the best performance regarding the mainstream metric for comparison, AR.

It is important to note that these results are based on the assumption of zero transaction costs, which is a conservative starting point and not very realistic. However, the results can be further improved (as we do in the extensions in the following subsections) by incorporating transaction costs and reducing the algorithm trading frequency while accounting for the cumulative contribution (agent asset return) in a future window. Figure 5 presents a visualization of the results of different algorithms for trading assets without transaction costs. The results indicate that RSLSTM-A (dotted green line) is a promising

approach for selecting the best action in the trading environment, outperforming other RL algorithms, and equaling or exceeding the B&H benchmark in most cases. Interestingly, the BTS baseline displayed competitive performance, achieving better results than mainstream RL methods for several evaluated assets. This result suggests that BTS is a possible technique for addressing the exploration-exploitation dilemma in online learning and that its performance could be further improved by using better approximation functions. Given the good results that BTS and our proposed method, RSLSTM-A, obtained, we gather some pieces of evidence that the Active Single-Asset Trading Problem (ASATP) is a contextual bandit problem that RL methods may not specifically develop to solve.

Not only RSLSTM-A achieved higher Annualized Returns and overall cumulative returns, but the stability of the returns was also maintained. The SR metric, which considers the trading system's stability under high volatility, shows that RSLSTM-A was better in four of six assets, making it the best model to balance risk and agent asset price return. It is worth noting that incorporating transaction costs can significantly affect the policies generated by these algorithms, particularly since they may lead to less frequent trading. However, by accounting for transaction costs and reducing the trading frequency, the performance of these algorithms can be improved.

Table 3: Consolidated results for the Accumulated Agent Asset Price Returns (ACR), Sharpe Ratio (SR), and annualized agent asset price return (AR) for all the models employed and assets without transaction costs. The results are a relative (a percentage) metric of the B&H strategy, as explained in the Subsection 5.2.2. The number in parenthesis ranks the method, being the first in ranking the best method for the respective metric. The best results of the average ranking are the ones in bold.

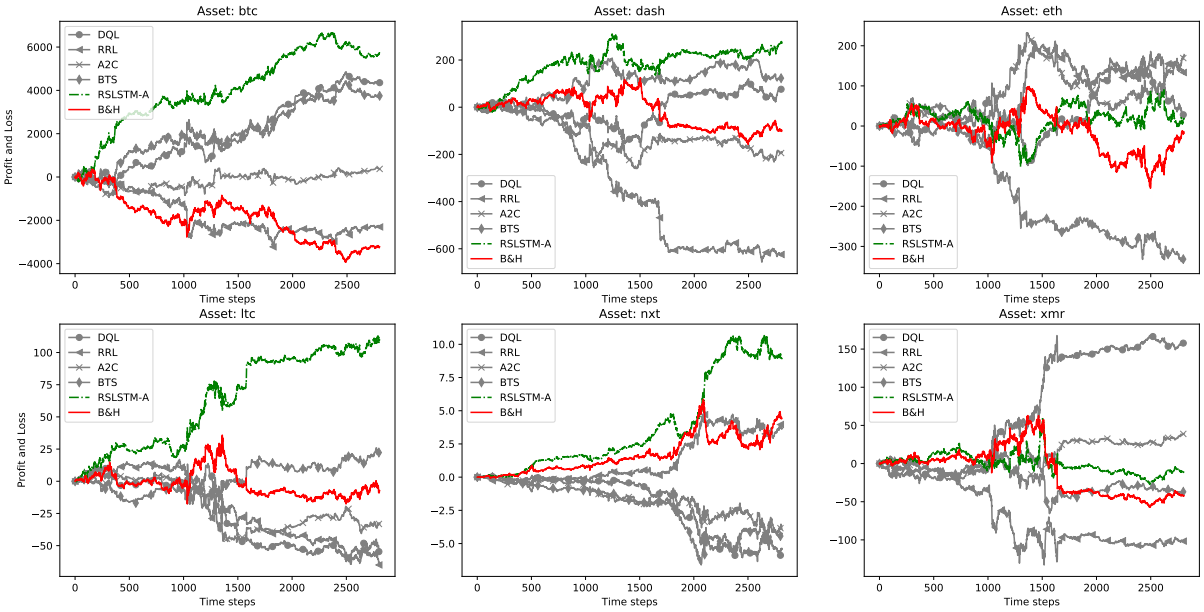
	Asset	**B&H**	**RRL%**	**DQN%**	**A2C%**	**BTS%**	**RSLSTM-A%**
	BTC	-5.31E+06(6)	15.44(5)	416.2(3)	184.67(4)	423.35(2)	629.38(1)
	DASH	-4.09E+04(4)	-65.69(6)	2.56(3)	-19.95(5)	24.01(2)	39.38(1)
ACR	ETH	-4.44E+04(5)	44.08(1)	22.35(3)	43.92(2)	-64.24(6)	14.84(4)
	LTC	-5.04E+03(3)	-52.38(5)	-65.02(6)	-47.09(4)	13.03(2)	178.37(1)
	NXT	4.70E+03(2)	-8.46(3)	-72.09(5)	-65.64(4)	-77.91(6)	43.04(1)
	XMR	-2.57E+04(4)	-63.34(6)	90.26(1)	18.23(2)	-10.16(5)	9.74(3)
	Avg. Rank	3.67	4.33	3.50	3.83	3.83	**1.83**
	BTC	-2.49E-06(6)	2063.45(5)	12569.48(1)	8955.02(4)	10079.52(3)	11926.91(2)
	DASH	4.55E-04(4)	-69.69(6)	8.55(3)	-4.91(5)	15.25(2)	16.05(1)
SR	ETH	7.15E-04(5)	27.7(2)	10.78(3)	45.42(1)	-84.27(6)	5.54(4)
	LTC	2.66E-03(3)	-80.73(6)	-58.52(5)	-25.38(4)	44.57(2)	82.47(1)
	NXT	2.35E-01(2)	-0.81(3)	-79.22(5)	-74.68(4)	-79.38(6)	10.55(1)
	XMR	1.06E-03(5)	-88.59(6)	181.76(1)	138.34(2)	24.81(4)	36.6(3)
	Avg. Rank	4.17	4.67	3.00	3.33	3.83	**2.00**
	BTC	3.51E-02(6)	55.84(5)	330.2(2)	184.33(4)	310.26(3)	370.09(1)
	DASH	7.35E-02(4)	-74.33(6)	10.8(3)	-7.1(5)	13.62(2)	21.13(1)
AR	ETH	9.15E-02(5)	27.87(2)	7.98(3)	36.61(1)	-84.48(6)	6.34(4)
	LTC	8.60E-02(3)	-98.84(6)	-74.53(5)	-38.28(4)	34.8(2)	114.49(1)
	NXT	3.91E-01(2)	-0.7(3)	-69.91(6)	-68.66(4)	-68.91(5)	2.88(1)
	XMR	6.97E-02(5)	-61.55(6)	133.86(1)	63.56(2)	4.73(4)	26.4(3)
	Avg. Rank	4.17	4.67	3.33	3.33	3.67	**1.83**

When considering the SR as the performance metric, RSLSTM-A's advantage over other techniques appears less pronounced, although it still offers the highest SR for most assets. Interestingly, RSLSTM-A consistently outperformed the B&H benchmark across all assets, underlining its effectiveness against this traditional, more conservative approach. As a strategy, B&H is unencumbered by transaction costs and is predicated on the efficient market hypothesis. However, for certain assets like Bitcoin and Ethereum, as illustrated in Figure 5, most of the tested algorithms surpassed the B&H benchmark. Fascinatingly, a few algorithmic behaviors mirrored the B&H strategy, suggesting a conservative tactic given that their performance closely tracked the benchmark. Regarding average rank analysis, RSLSTM-A emerged as the top performer across all metrics. The Wilcoxon signed-rank test, conducted to compare average ranks, revealed evidence favoring RSLSTM-A over DQN for all metrics (AR, SR, ACR). This result indicates RSLSTM-A's potential as an effective strategy for the ASATP, as it outperformed standard RL methods and demonstrated superior time series classification performance with

its ResNet and LSTM combination architecture. The superior performance of RSLSTM-A can largely be attributed to ResNet's proven efficacy in time series classification tasks, as outlined by Fawaz et al. (2019), who showed ResNet outperforming Multilayer Perceptron in this task, which further validates our method. This performance makes us question the absolute necessity of RL methods for ASATP. In scenarios where the agent's actions do not influence the state, optimizing time series classification and heuristics should be emphasized to reduce transaction costs.

Another remarkable aspect of RSLSTM-A is the smoothness in the loss reduction observed during training and validation, facilitating overfitting control and favoring its generalization capacity and better performance. RL methods working with very noisy data are unstable during the training phase (HENDERSON et al., 2018; PAIVA et al., 2022), making it harder to control overfitting. At the same time, Supervised Learning uses many techniques to have a smooth convergence with overfitting control using validation sets, giving our method a generalization advantage. Finally, regarding the LSTM, as previously pointed out, it plays an essential role in the proposed architecture by improving the time series classification performance and dealing with the inconsistency between different observed periods of the time series. The LSTM architecture has the properties to learn data dynamics better and is an excellent combination for ResNet, as observed by Choi, Ryu and Kim (2018). This combination likely contributed to RSLSTM-A's consistent performance across different assets and better generalization capacity.

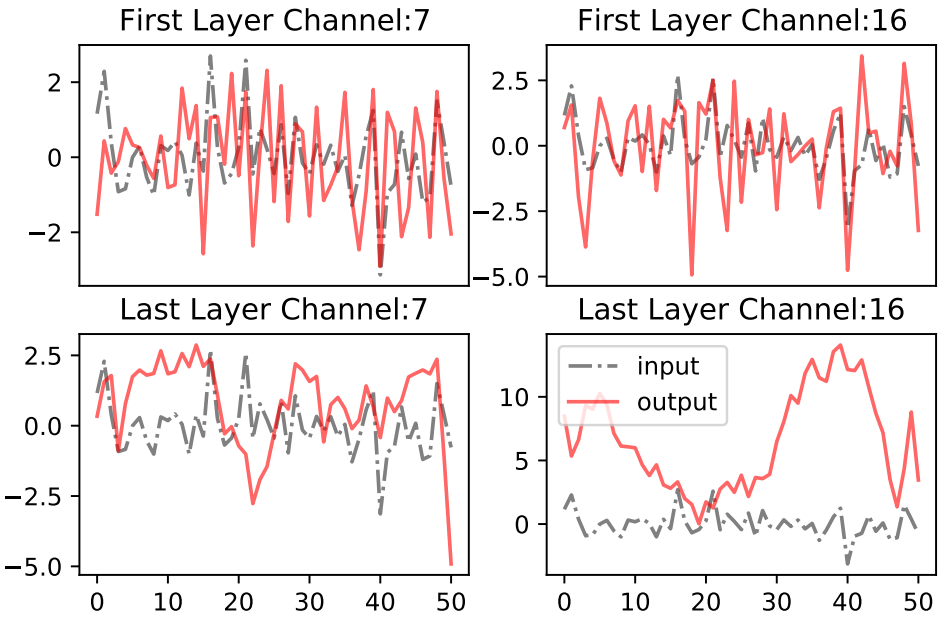
Figure 5: Cumulative agent asset price return for the test set of all assets (BTC, DASH, ETH, LTC, NXT, XMR) for transaction cost equals to zero.



Source: Felizardo et al. (2022a).

Convolutional neural networks serve as a pre-processing step to extract features that assist the agent in selecting the optimal action. The LSTM network utilizes these extracted features, retaining the memory of previous actions and states, to extract new features. A linear layer generates the probability of success for each action given a state. In our analysis, we also examine the outputs of some convolutional channels in the RSLSTM-A architecture, specifically the first and last convolutional layers. To examine the extracted features from the original time series, we present two samples of the features generated by the convolutional layers in Figure 6.

Figure 6: Two samples of the first and the last convolutional layer outputs. The x-axis, ranging from zero to 50, is the input size of the network (state dimension or the look-back window time series), and the y-axis displays the agent asset price return values for each time step.



Source: Felizardo et al. (2022a).

As shown in Figure 6 (bottom part of the figure), the last convolutional layer exhibits a more pronounced behavior that has less overlapping with the original input. The features extracted by the final layers capture a trend approximation or a volatility estimation. As Figure 6 shows, the convolutional neural networks generate forecasts in some channels. The extracted sample of the 128 channels also shows that channels 7 and 16 appear to represent an intensified representation of the price change trend (the original signal), with multiple channels displaying similar behavior. We can observe this intensified representation as the channel output is a similar signal but with a higher amplitude.

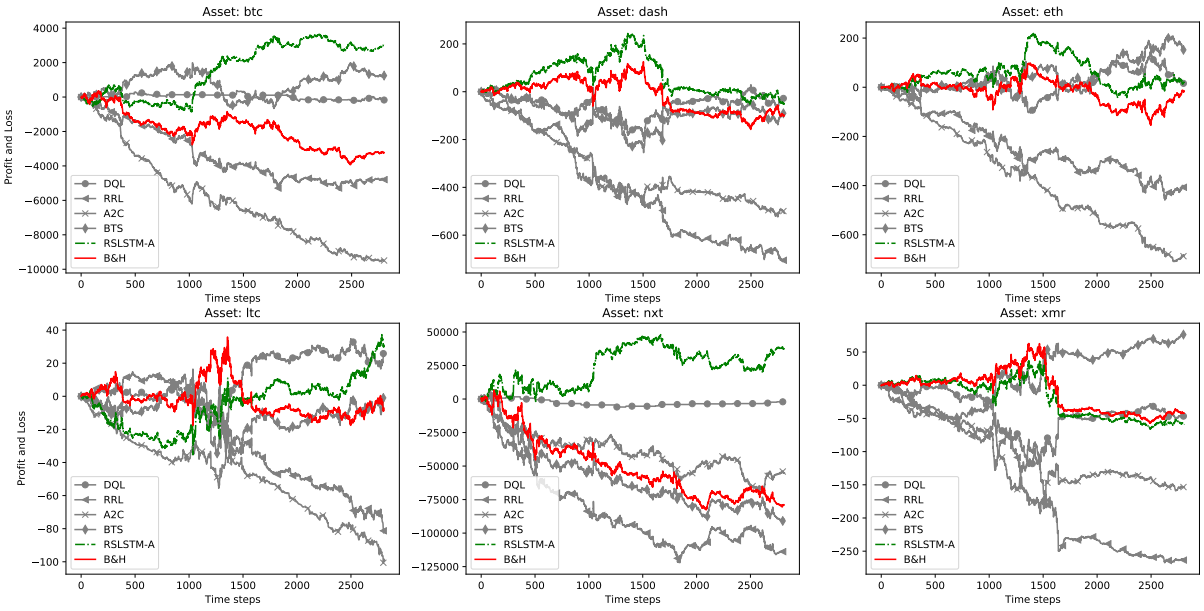
Given the improved performance observed in the experiments, we may assume that these convolutional mechanics enhance the most relevant signals as features for decision-making. This assumption is one possible interpretation of the phenomenon through the graphical analysis of channel outputs. However, such an analysis of outputs is a promising method for understanding the network's learning process and providing insights into interpreting the features extracted from convolutional neural networks.

5.2.4 The effect of transaction costs

One possible argument for using RL techniques is the ability to account for future contributions in scenarios with transaction costs rather than only considering immediate contributions. In light of this, we adapted the labeling heuristic to incorporate the cumulative future value of price returns. The optimal action is to assume a short position (-1) if the cumulative sum of price returns is negative and a long position (1) if it is positive. In addition, when employing the RSLSTM-A under the presence of transaction costs, we choose a lower frequency of trades to mitigate the impact of transaction costs. In this case, the trading frequency is a hyperparameter of our proposed model, different from the RL method on which this best trading frequency is learned.

In a subsequent experiment, we adopted a future window of size $M^f = 80$ to calculate future contributions for the training of our RSLSTM-A model. Using this straightforward approach, RSLSTM-A outperformed other models across most assets, consistently exceeding the B&H benchmark. As shown in Figure 7, the RSLSTM-A often has a similar pattern of movements compared to the asset price trajectory yet remains above it. For certain assets, such as NXT, RSLSTM-A outpaced other techniques and surpassed the B&H benchmark considerably.

Figure 7: Cumulative agent asset price return for the test set of all assets (BTC, DASH, ETH, LTC, NXT, XMR) for transaction cost equals to 0.001.



Source: Felizardo et al. (2022a).

Surprisingly, the DQN algorithm performed competitively against the RSLSTM-A algorithm when transaction costs were present, as shown in Table 4. For example, concerning the ACR metric, DQN was better than the RSLSTM-A for the LTC cryptocurrency. It is important to note that DQN performed worse when transaction costs were not present. Although DQN outperformed RSLSTM-A regarding the SR metric, this metric may be overestimated as DQN tends to adopt a neutral position in most cases. When considering the SR metric, DQN was the best technique, being better in three assets: BTC, DASH, and NXT. Using the rank metric, we can infer that RSLSTM-A outperforms the other techniques. We compare the mean ranks using the Wilcoxon signed rank test to evaluate if the difference is significant. Comparing RSLSTM-A results to the second-best method, the DQN, and the third-best, B&H and BTS, we cannot affirm that we have significant differences in ACR. The significant difference appears from the RSLSTM-A to the A2C. Therefore, we have weak evidence that RSLSTM-A is better than DQN and BTS.

Table 4: Consolidated results for the Accumulated Agent Asset Price Returns (ACR), Sharpe Ratio (SR), and annualized agent asset price return (AR) for all the models employed and assets with transaction costs equal to 0.001. The results are a relative (a percentage) metric of the B&H strategy, as explained in the Subsection 5.2.2. The number in parenthesis ranks the method, being the first in ranking the best method for the respective metric. The best results of the average ranking are the ones in bold.

	Asset	**B&H**	**RRL%**	**DQN%**	**A2C%**	**BTS%**	**RSLSTM-A%**
	BTC	-5.31E+06(4)	-20.88(5)	27.46(3)	-57.63(6)	38.23(2)	49.77(1)
	DASH	-4.09E+04(2)	-65.85(6)	-7.08(3)	-52.61(5)	-13.61(4)	12.7(1)
ACR	ETH	-4.44E+04(4)	-41.15(5)	5(3)	-65.69(6)	12.63(2)	13.75(1)
	LTC	-5.04E+03(2)	-31.73(5)	16.83(1)	-65.82(6)	-12.18(4)	-4.49(3)
	NXT	-1.41E+08(4)	-44.44(6)	62.74(2)	14.18(3)	-11.35(5)	98.68(1)
	XMR	-2.57E+04(2)	-65.21(6)	-15.15(4)	-45.87(5)	11.18(1)	-4.82(3)
	Avg. Rank	3.00	5.50	2.67	5.17	3.00	**1.67**
	BTC	-2.49E-06(4)	-17.59(5)	71.86(1)	-66.4(6)	29.48(3)	37.46(2)
	DASH	4.55E-04(4)	-68.01(6)	1.78(1)	-19.96(5)	0.36(3)	1.19(2)
SR	ETH	7.15E-04(4)	-16.62(5)	0.72(3)	-69.44(6)	4.37(1)	0.87(2)
	LTC	2.66E-03(4)	-36.71(5)	8.27(2)	-70.81(6)	1.96(3)	9.95(1)
	NXT	-2.16E-05(6)	92.59(5)	541.67(1)	142.96(3)	111.11(4)	258.43(2)
	XMR	1.06E-03(2)	-70.66(6)	-1.12(3)	-38.21(5)	25.21(1)	-3.81(4)
	Avg. Rank	4.00	5.33	1.83	5.17	2.50	**2.17**
	BTC	3.51E-02(4)	-1.19(5)	1.69(3)	-67.02(6)	2.31(2)	3.03(1)
	DASH	7.35E-02(4)	-70.54(6)	2.42(1)	-22.34(5)	0.21(3)	1.72(2)
AR	ETH	9.15E-02(4)	-25.42(5)	1.2(3)	-74.28(6)	7.12(1)	1.45(2)
	LTC	8.60E-02(4)	-55.27(5)	13.81(2)	-78.66(6)	2.18(3)	16.74(1)
	NXT	-2.67E-01(4)	-63.76(6)	11.67(2)	7.05(3)	-63.11(5)	13.55(1)
	XMR	6.97E-02(2)	-67.36(6)	-0.1(3)	-2.77(5)	1.82(1)	-0.31(4)
	Avg. Rank	3.67	5.50	2.33	5.17	2.50	**1.83**

5.2.5 Extended results with higher costs

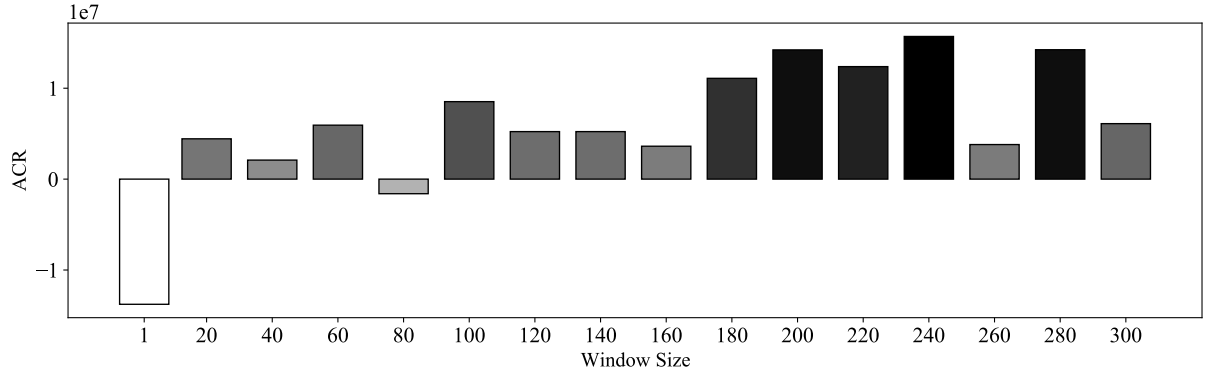
In an effort to enhance the performance of the RSLSTM-A and gather more compelling evidence of its effectiveness, we analyzed to determine the optimal future window, M^f, that would maximize profit. The results of this analysis are depicted in Figure 8, which illustrates how the performance of RSLSTM-A varies with adjustments to the future window (M^f) and the frequency of trades.

For this analysis, we selected BTC as our representative asset to help establish a suitable value for the future window hyperparameter. While a more sophisticated method could be employed to determine this hyperparameter, we opted for a straightforward

approach to visually demonstrate one possible strategy for deciding the size of the future window.

Our findings indicate that the most favorable results were achieved within a future window range of 100 to 200. These results suggest implementing a simple heuristic to regulate trade frequency and consider cumulative future price changes when making trading decisions.

Figure 8: Testing the ResNet performance for the BTC asset trading considering different future windows sizes from 1 to 300. The gray scale and the size of the bars are related to the value of the ACR, being a darker bar, also a higher ACR.



One of the contributions of this thesis is to expand upon our previous work in Felizardo et al. (2022a) by evaluating a more comprehensive range of assets and incorporating the PPO method while also considering higher transaction costs. We increased the transaction cost to 0.002, double the cost used in our previous work, and decided to use $M^f = 240$ based on the analysis shown in Figure 8. This future trading window is three times larger than the previous window for the transaction cost 0.001. The results shown in Table 5 are consistent with those reported in Table 4. In this increased transaction cost configuration, we observed a statistically significant difference in the average rank of RSLSTM-A and all the other ACR and AR metrics methods as determined through the Wilcoxon signed-rank test. Similarly, for the SR metric, the Wilcoxon signed rank test results indicate a significant difference in the average rank of DQN and RSLSTM-A, but we do not find a significant difference between DQN and the B&H. In summary, the RSLSTM-A model is the top-performing model overall. The other models exhibit poor performance in comparison to the DQN and B&H for the SR metric. The B&H strategy has the poorest performance for the AR and ACR metrics, but it was the best strategy, along with DQN, when considering the SR metric.

Table 5: Consolidated results for the Accumulated Agent Asset Price Returns (ACR), Sharpe Ratio (SR), and annualized agent asset price return (AR) for all the models employed and assets using transaction cost equals 0.002. The results are a relative (a percentage) metric of the B&H strategy, as explained in the Subsection 5.2.2. The number in parenthesis ranks the method, being the first in ranking the best method for the respective metric. The best results of the average ranking are the ones in bold.

	Asset	**B&H**	**RRL%**	**DQN%**	**A2C%**	**PPO%**	**BTS%**	**RSLSTM-A%**
	BTC	-5.31E+06(6)	18.7(5)	61.4(4)	119.31(3)	-48.68(7)	157.62(2)	192.17(1)
	DASH	-4.09E+04(3)	-59.68(6)	-6.2(4)	-66.15(7)	-39.14(5)	1.6(2)	5.24(1)
	ETH	-4.44E+04(3)	-45.34(5)	-9.23(4)	-64.94(6)	-65.93(7)	8.48(2)	10.76(1)
	LTC	-5.04E+03(2)	-47.64(5)	-4.31(3)	-64.11(6)	-66.17(7)	-29.7(4)	4.98(1)
ACR	NXT	-1.41E+08(6)	87.73(5)	137.38(4)	268.09(1)	-33.33(7)	229.08(2)	200.71(3)
	XMR	-2.57E+04(3)	-33.97(6)	3.71(2)	-65.92(7)	-22.48(5)	-3.51(4)	7.66(1)
	ETC	-7.50E+05(7)	305.6(5)	353.6(3)	63.2(6)	474.67(1)	338.67(4)	400.27(2)
	XRP	-2.66E+07(6)	38.75(5)	99.94(4)	-40.24(7)	155.59(2)	143.67(3)	270.64(1)
	XEM	-3.03E+07(6)	-24.58(7)	319.58(2)	259.17(3)	420.83(1)	173.75(5)	235.83(4)
	LSK	-3.09E+05(3)	-20.76(6)	-10.57(4)	-14.52(5)	7.4(2)	-64.93(7)	1084.29(1)
	Avg. rank	4.50	5.50	3.40	5.10	4.40	3.50	**1.60**
	BTC	5.88E-05(3)	-45.58(6)	10.71(1)	-35.03(5)	-57.82(7)	-29.25(4)	9.52(2)
	DASH	7.06E-04(1)	-54.67(6)	-7.93(3)	-53.68(5)	-47.31(4)	-0.85(2)	-64.59(7)
	ETH	7.51E-04(1)	-55.53(5)	-7.06(2)	-51.26(4)	-57.92(6)	-66.58(7)	-49.67(3)
	LTC	3.11E-03(1)	-51.77(3)	-49.84(2)	-56.91(6)	-54.98(4)	-55.31(5)	-65.59(7)
SR	NXT	-4.85E-07(6)	159.93(5)	298.34(2)	224.17(4)	-55.96(7)	247.35(3)	422.85(1)
	XMR	1.85E-03(1)	-55.24(5)	-50.05(2)	-51.14(3)	-55.14(4)	-62.43(6)	-65.78(7)
	ETC	7.96E-04(1)	-50.63(7)	-2.64(2)	-47.86(5)	-48.99(6)	-12.69(3)	-31.41(4)
	XRP	7.34E-06(4)	-40.6(6)	30.93(2)	-44.82(7)	-20.84(5)	35.01(1)	0.41(3)
	XEM	8.15E-06(2)	-64.91(7)	-30.43(4)	-59.02(6)	-47.85(5)	-13.01(3)	16.81(1)
	LSK	1.06E-04(1)	-51.04(4)	-5.66(2)	-63.49(6)	-66.23(7)	-34.34(3)	-63.4(5)
	Avg. rank	**2.00**	5.10	2.30	4.80	5.20	3.70	4.90
	BTC	5.21E-02(6)	57.39(5)	99.62(4)	161.04(2)	-72.55(7)	197.5(1)	132.25(3)
	DASH	6.76E-02(4)	-66.74(6)	0.28(3)	-67.32(7)	-65.58(5)	1.28(2)	2.73(1)
	ETH	9.08E-02(2)	-66.67(5)	-0.79(3)	-67.55(6)	-67.55(6)	0.38(1)	-1.5(4)
	LTC	8.49E-02(3)	-66.91(5)	0.04(2)	-67.5(6)	-67.5(6)	-2.34(4)	0.94(1)
AR	NXT	2.91E-02(6)	161.39(5)	195.06(4)	314.03(1)	-88.44(7)	275.87(2)	248.93(3)
	XMR	5.64E-02(4)	-8.86(6)	2.28(2)	-67.21(7)	-4.3(5)	2.08(3)	3.23(1)
	ETC	3.19E-02(7)	279.31(5)	304.39(3)	42.95(6)	373.35(2)	294.98(4)	398.43(1)
	XRP	7.49E-02(6)	26.03(5)	34.85(4)	-9.35(7)	49.53(2)	42.86(3)	52.2(1)
	XEM	5.95E-02(6)	-90.72(7)	81.87(3)	21.1(5)	109.76(1)	29.96(4)	103.7(2)
	LSK	9.36E-02(3)	-4.27(6)	-1.6(4)	-2.14(5)	0.53(2)	-6.94(7)	0.96(1)
	Avg. rank	4.80	5.50	3.30	5.30	4.20	3.30	**1.40**

The results of our experiments suggest that RSLSTM-A is a highly effective method for trading single assets. In most cases, our proposed approach outperformed DQN, BTS, A2C, PPO, RRL, and B&H, demonstrating its potential for maximizing profits in real-world scenarios. While DQN performed well in some scenarios, RSLSTM-A consistently outperformed all other methods when considering the AR and ACR metrics. When considering the SR metrics, the B&H outperformed all algorithms, and the DQN was the second best. BTS and DQN were the next-best-performing methods, performing similarly

to RSLSTM-A for some assets. Interestingly, recent DRL methods such as PPO and A2C could not perform better than RSLSTM-A or BTS, and RRL was not competitive. In fact, in some cases, A2C, PPO, and RRL performed worse than the B&H strategy, while DQN was the only method to achieve better results than B&H in some cases. Moreover, our experiments indicate that the future trading window (M^f) significantly influences the results when transaction costs are present.

6 STOCHASTIC DISCRETE LOT-SIZING PROBLEM

In the rich realm of operations research and decision-making, many are the challenges. One such intricate challenge is the Stochastic Discrete Lot-Sizing Problem (SDLSP). The SDLSP concerns determining the optimal quantities and timings for ordering or producing items to meet uncertain future demands while minimizing associated costs. Like the intricacies of deciding when to buy, sell, or hold an asset in the world of finance, as seen in the Active Single-Asset Trading Problem (ASATP) discussed in the previous chapter, the SDLSP hinges on discerning patterns and making informed decisions in the face of uncertainty. Factors like supply chain disruptions, demand variability, and production constraints are critical in shaping these decisions.

The decision-making principles applied in SDLSP echo across various other domains. For instance, as previously discussed in ASATP, determining the optimal time to make a trade requires a deep understanding of asset dynamics guided by historical data and market trends. Similarly, the SDLSP leans heavily on historical demand data and supply chain patterns. Here, in this problem, additional complexities may arise. As alluded to in Chapter 3, the problem constraints of SDLSP necessitate tailored solutions that consider more complex dynamics between states and actions. In the ASATP, we modeled the problem as a contextual bandit problem, but in SDLSP, we have to deal with a full Reinforcement Learning problem (see (SUTTON; BARTO, 2018) for the explanation of full Reinforcement Learning and contextual bandit).

Assuming an environment model of the problem, we can propose many solutions, but like all models and algorithms, those solutions have their boundaries. Here, one special limitation is connected to the dimensionality of the problem. As we are dealing with a nondeterministic polynomial-time hard (NP-hard) problem (see Chapter 2), in this lot-sizing problem instance, we might require a better heuristic as a solution. To find this heuristic, RL fits as one possible candidate for the solution policy search method. RL also has potential pitfalls, as we will further observe when testing in simulated environments.

This chapter delves into three methods to address the SDLSP. We commence with the Branch and Bound Approximate Dynamic Programming (BBADP) method, an approach rooted in Dynamic Programming to calculate the optimal solution, assuming that the demand distribution is known. Then, we explore the Decision Rule (DR) method, which uses eligibility and priority rules to guide the decision-making process. Lastly, we proceed to an innovative method, the Lot-Sizing Cooperative Multi-Agent Adjustment (LSCMA), that relies on a cooperative multi-agent approach to improve the recommendations from a baseline agent using multiple RL sub-agents.

This chapter tests these methods in a simulated environment for comparative analysis. We present the experimental results of employing these methods and the other RL solutions to supplement our examination. A comprehensive analysis of these results sheds light on their performance under various conditions, offering insights into their potential real-world applications.

Much like our previous investigation into the ASATP, this exploration of SDLSP methods forms part of a broader goal. We aim to understand the landscape of solutions available for the SDLSP, identifying the most effective solutions within the context of this domain. Our focus remains on presenting an enlightening and informative analysis as we navigate this exploration.

6.1 Stochastic Discrete Lot-Sizing Methods

The Stochastic Discrete Lot-Sizing Problem (SDLSP) can be addressed using various methods, each with unique characteristics and potential applications. This section focuses on the methods developed explicitly for the SDLSP: the Branch and Bound Approximate Dynamic Programming (BBADP), the Decision Rule (DR) heuristic, and the Lot-Sizing Cooperative Multi-Agent Adjustment (LSCMA) method.

Following the next subsections, first, we present our proposed method BBADP, which is a mathematical approach grounded in Dynamic Programming, offering precise solutions for medium and large-size instances of the SDLSP. Our following proposed method, the DR method, serves as a benchmark heuristic to compare the performance of our proposed solutions, offering an eligibility-based and priority-based solution. Finally, we also propose the LSCMA, which joins multiple RL agents in a cooperative system, each working towards enhancing the initial recommendations of a baseline agent (we further explain this concept in Subsection 6.1.3).

The following sections plunge into the specifics of the methods of the BBADP, DR, and LSCMA. We highlight their underlying principles, functionalities, and potential applications, providing a thorough understanding of their strengths and weaknesses that could be instrumental in guiding future research and application in this field.

6.1.1 Branch and Bound Approximate Dynamic Programming

For smaller instances of the SDLSP problem, Dynamic Programming can be employed to compute the optimal solution, given the assumption of known demand distribution. Leveraging the mathematical framework set out by Powell (2021), which is detailed in Section 3, we designate the state *value function* as $V_t(S_t)$:

$$\begin{aligned} V_t\left(S_t\right) &= \min_{X_t \in \mathcal{X}} \mathbb{E}\left[C\left(S_t, X_t\right) + \gamma V_{t+1}\left(S_{t+1}\right)\right] = \\ &= \min_{X_t \in \mathcal{X}} \mathbb{E}\left[C\left(S_t, X_t\right)\right] + \gamma \mathbb{E}\left[V_{t+1}\left(S_{t+1}\right)\right]. \end{aligned} \tag{6.1}$$

Regrettably, Equation (6.1) presents a complex stochastic optimization problem and is subject to the curse of dimensionality, rendering it solvable only in small-sized instances. One possible approach to simplify this problem, thereby enabling Approximate Dynamic Programming (ADP) grounded in statistical learning, involves the introduction of post-decision states, as suggested by Powell (2011). In this context, post-decision state variables represent the changes in the state following the implementation of a decision but prior to the realization of risk factors. Within the framework of the SDLSP, we define post-decision states as the inventory level after replenishment but before demand fulfillment ($\mathbf{I_t}^x$), as well as the setup state at the end of the time bucket ($\mathbf{M_t^x}$). These post-decision states are linked to the subsequent pre-decision states through the following relationship, as represented in Equation (6.2):

$$\begin{aligned} \mathbf{I}_{t+1} &= \left[\mathbf{I_t}^x - \mathbf{d}_t\right]^+ \\ \mathbf{M}_{t+1} &= \mathbf{M}_t^x. \end{aligned} \tag{6.2}$$

By employing these variables, we can construct a value function centered around post-decision states, denoted as $V_t^x\left(\mathbf{I}t^x, \mathbf{M}t^x\right)$. The conventional Dynamic Programming recursion for pre-decision states is then as follows:

$$V_t\left(S_t\right) = \min_{X_t \in \mathcal{X}} \left\{\mathbb{E}\left[C\left(S_t, X_t\right) \mid X_t, S_t\right] + \gamma \mathbb{E}\left[V_{t+1}\left(S_{t+1}\right) \mid S_t, X_t\right]\right\}. \tag{6.3}$$

The value function surrounding the post-decision state is defined as:

$$V_t^x\left(S_t^x\right) = \mathbb{E}\left[V_{t+1}\left(S_{t+1}\right) \mid S_t^x\right]. \tag{6.4}$$

By substituting Equation (6.4) into Equation (6.3), we obtain:

$$V_t\left(S_t\right) = \min_{X_t \in \mathcal{X}} \left\{\mathbb{E}[C\left(S_t, X_t\right) | X_t, S_t] + \gamma V_t^x\left(S_t^x\right)\right\} \tag{6.5}$$

Taking expectations at $t-1$ yields:

$$V_{t-1}^x\left(S_{t-1}^x\right) = \mathbb{E}\left[V_t\left(S_t\right) \mid S_{t-1}^x\right] = \mathbb{E}\left[\min_{\mathbf{X}_t \in \mathcal{X}} \left\{\mathbb{E}[C\left(S_t, X_t\right) | X_t, S_t] + \gamma V_t^x\left(S_t^x\right)\right\}\right] \tag{6.6}$$

Equation (6.6) enables the interchangeability of optimization and expectations with respect to Equation (6.3). We decompose the expected value of the immediate contribution into:

$$\mathbb{E}[C\left(S_t, X_t\right)] = D\left(S_t, X_t\right) + G\left(S_t, X_t\right), \tag{6.7}$$

where $D(S_t, X_t) = \sum_{m=1}^{M} \sum_{i=1}^{I} f_i \delta_{i,m,t}$ is a deterministic component linked to the setup costs. The stochastic component associated with the holding and lost sales costs, which must be learned, is defined as:

$$\begin{aligned} G\left(S_t, X_t\right) = \mathbb{E}\Bigg[& \sum_{i=1}^{I} h_i \left[I_{i,t} + \sum_{m=1}^{M} \left(p_{i,m} x_{i,m,t} - c_{i,m} \delta_{i,m,t}\right) - d_{i,t}\right]^+ \\ & + \sum_{i=1}^{I} l_i \left[d_{i,t} - I_{i,t} + \sum_{m=1}^{M} \left(p_{i,m} x_{i,m,t} - c_{i,m} \delta_{i,m,t}\right)\right]^+ \Bigg] \end{aligned} \tag{6.8}$$

Expressing $G\left(S_t, X_t\right)$ as $G^x\left(\mathbf{I}_t^x\right)$ using post-decision state variables, it's clear that $G^x\left(\mathbf{I}_t^x\right)$ is additively separable with respect to the items:

$$G(\mathbf{I}_t^x) = \sum_{i=1}^{I} G(I_i^x), \text{ where } G_i(I_i^x) = \mathbb{E}\left[h_i[I_i^x - d]^+ + l_i[d - I_i^x]^+\right]. \tag{6.9}$$

Each $G_i(I_i^x)$ is the expected value of piecewise linear convex functions, thus preserving convexity (refer to Section 3.2.1 of Boyd and Vandenberghe (2004)). Consequently, $G(\mathbf{I}_t^x)$ also remains convex, being a summation of convex functions. We approximate each $G_i(I_i^x)$ with a piecewise linear approximation based on a regression tree, denoted as $G(I_i^x)$.

Since V_t^x still suffers from the curse of dimensionality, we approximate it with a sum of two components: one related to the inventory and one related to the machine

configuration. In formula:

$$V^x(\mathbf{I}_t^x, \mathbf{M}_t^x) = \sum_i V_i^{(I)}(I_{i,t}^x) + \sum_i V_i^{(M)}(n_{i,t}^x), \tag{6.10}$$

where:

- $V_t^x(\mathbf{I}_t^x, \mathbf{M}_t^x)$ is the post-decision value function approximation,
- $n_{i,t}^x$ is the number of machines that are producing item i at time t computed after that the decision is made,
- $V_i^{(I)}$ is the function accounting for the contribution of the inventory.
- $V_i^{(M)}$ is the function accounting for the machine states.

Since each inventory may have $I_{\max}$ maximum value and the maximum number of machines producing an item is M, we consider all the $V_i^{(I)}$ and $V_i^{(M)}$ in a tabular representation[1].

It is worth noting that the post-decision value function approximation in Eq. (6.10) does not consider the time index. With this choice, we consider the infinity time horizon problem to approximate a finite time horizon. This approximation is usually made, e.g., in Hezewijk et al. (2022).

The algorithm used to learn $G(\mathbf{I})$, $V_i^{(I)}$, and $V_i^{(M)}$ is described in Algorithm 8.

[1] A tabular representation is often used to store the value of each state or state-action pair in a table. This type of representation is typically used when the state and action spaces are discrete and not too large, which allows for an exact solution. When using the tabular representation, we suppress the θ parameter in the function notation as the tabular representation does not adjust the model's parameters, just the values in the table

Algorithm 8 ADP branch and bound

procedure ADP BRANCH AND BOUND TRAINING(N)
    ▷ Number of training episodes: N
    ▷ Output: Trained parameters for $V_i^{(I)}$, $V_i^{(M)}$, $G_i(\mathbf{I})$: θ_N
    Initialize $V_i^{(I)}, V_i^{(M)}, G_i(\mathbf{I})$ to zero, $\forall I = 0, \ldots, I_{\max}, \forall i$.
    ▷ Execution
    for $k \in (0, N]$ **do**
        Set the initial state s_0.
        Generate a sample path for the demand $d, t = [1, \ldots, T]$.
        for $t \in (0, T]$ **do**
            ▷ Compute the objective function

$$X_t^* = \begin{cases} \arg\min_x D(S_t, X_t) + \sum_i [G_i(\mathbf{I}_t^x) + \gamma V_k^x(\mathbf{I}_t^x, \mathbf{M}_t^x)] \text{ with probability} \\ 1 - \epsilon \text{random with probability } \epsilon \end{cases}$$

$$\widetilde{V}_t = \min_x D(S_{t+1}, X_t) + \sum_i [G_i(\mathbf{I}_t^x) + \gamma V_k^x(\mathbf{I}_t^x, \mathbf{M}_t^x)] \tag{6.11}$$

            Updates V_{k+1}^x using $\widetilde{V}t$
            ▷ Generate next post-decision state based on the current state and the optimal decision
            $s_t^x \leftarrow S^M(s_t, x_t)$
            ▷ Generate next pre-decision state
            $s_{t+1} \leftarrow S^M(s_t, x_t, d_t)$
            Updates $G_{k+1}(I^x)$ using the observation of $G(s_t, x_t)$
    return θ_N

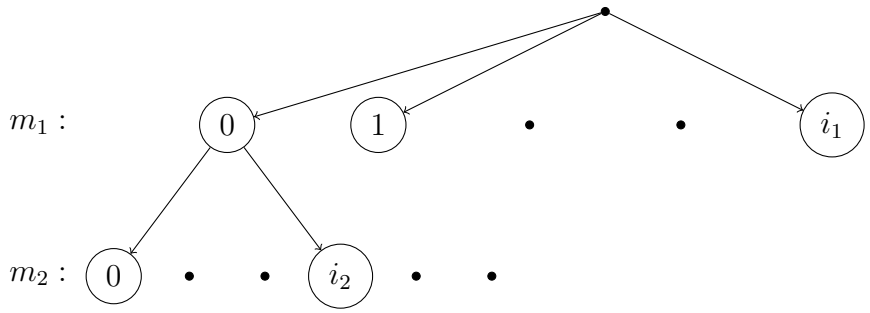
The central role in the Approximate Dynamic Programming algorithm is held by problem (6.11), as it provides data for both updating the value function estimation and determining the optimal action. Given its significance, we have opted for an exact solution. However, due to the nonlinearity of model (6.11) brought about by V^x, an exhaustive search procedure for all possible solutions becomes necessary. This procedure would mean exploring $\prod_{m=1}^{M} |I_m|$ potential solutions, which is unfeasible. As a remedy, we have adapted the branch and bound algorithm to enable a smart exploration strategy.

The branch and bound algorithm creates a tree structure with a root node at the beginning and several levels, each associated with a machine. Each node within a level represents a potential state for the corresponding machine — this could be a particular item to produce or an idle state. A solution path from the root node to a leaf node

presents a potential solution for model (6.11), indicating a state for each machine. This path from the root node to any intermediate node is referred to as a partial solution. We illustrate a portion of a typical tree in Figure 9.

We implement two pruning strategies to constrain the search space: feasibility pruning and optimality pruning.

Figure 9: Part of the general branch and bound tree.



We apply feasibility pruning when the quantity of items produced by a partial solution leads to an inventory greater than the maximum allowed. For example, suppose the maximum inventory level for item i equals 10. In that case, the initial inventory equals 5, and we are considering a partial solution in which the production of item i is 6. There is no point in continuing the exploration of the successor of that node since they will violate the maximum inventory constraint. Instead, we apply optimality pruning if all the successors of one node lead to a sub-optimal solution. We can detect this condition by looking at the sum of the setup cost and the expected inventory cost since it is a lower bound of the cost of the final solution. If this value is greater than the value of an incumbent solution, continuing the exploration is pointless.

6.1.2 Decision Rule

In order to evaluate the performance of our proposed solutions, we use another benchmark heuristic for comparison. This benchmark heuristic is predicated on eligibility and priority, based on expected run-out times and the current state of machine setups, as proposed by Karmarkar (1981). A comparable linear discrete choice model, which proved reasonably effective in addressing the decision problem, was also utilized in one of our prior studies as documented in Gioia, Felizardo and Brandimarte (2023). When demand is independent, we can readily calculate the expected run-out time for each item. The heuristic operates in three steps:

1. Identification of all eligible items (produced by the machines) with an expected run-out time exceeding a certain threshold.

2. Creation of a priority list of eligible items, considering a blend of factors, including the ratio of lost sales costs to expected run-out time, the number of machines producing the item, and the ratio of average demand to maximum production capability.

3. Assign to each machine the item to produce (or if it will move to the idle state).

The decision-making flow of the heuristic is depicted in Algorithm 9. The heuristic begins by calculating the average demand, denoted as $\bar{d}_i$, for each item. This computation is performed by employing Monte Carlo techniques. After this step, the heuristic calculates the anticipated run-out time for each item by dividing the existing inventory by the average demand. Items whose estimated run-out time falls below a specific threshold, denoted as θ_1, are considered eligible for production. Subsequently, the priority of each eligible item is determined based on a careful balance of several factors:

1. The ratio of lost sales costs to the anticipated run-out time: This criterion prioritizes items with high lost sales costs and short expected run-out times.

2. The total number of machines producing the item: A larger number of machines will inversely affect the item's priority.

3. The ratio of average demand to the maximum possible production: A higher ratio implies a greater demand for the item and, thus, an increased need for machines to meet this demand, which enhances the item's priority.

The weights assigned to these factors are θ_2, θ_3, and θ_4, and the sum of their weighted contributions is called the "priority". For simplicity, we assume that θ_2 is equal to 1. Following this, the eligible items are sorted in order of their priorities.

The algorithm starts by setting all machines to idle. It then goes through the items, starting from the highest priority and moving down to the lower ones. For each item, it checks whether there is a machine currently producing that item. If more than one machine is producing the item, the machine with the highest setup costs continues the production. If no machine produces the item, production starts on the idle machine with the smallest production ratio to setup costs.

After all eligible items have been assigned to machines, the algorithm reviews the machines set to idle but still producing items. These machines can continue production if the setup cost exceeds a threshold, denoted as θ_5, multiplied by an estimated inventory cost estimate. This estimate is calculated as follows:

$$\begin{aligned}
&\sum_{t=0}^{\lfloor \frac{I_i^x}{\bar{d}_i} \rfloor} h(I_i + p_{i,m} - \bar{d}_i t) = \\
&= hI_i \left(\left\lfloor \frac{I_i^x}{\bar{d}_i} \right\rfloor + 1 \right) - h\bar{d}_i \sum_{t=0}^{\lfloor \frac{I_i^x}{\bar{d}_i} \rfloor} t = \\
&= hI_i \left(\left\lfloor \frac{I_i^x}{\bar{d}_i} \right\rfloor + 1 \right) - h\bar{d} \frac{\lfloor \frac{I_i^x}{\bar{d}_i} \rfloor \left(\lfloor \frac{I_i^x}{\bar{d}_i} \rfloor + 1 \right)}{2} = \\
&= h \left(\left\lfloor \frac{I_i^x}{\bar{d}_i} \right\rfloor + 1 \right) \left(I_i^x - \frac{\bar{d}_i}{2} \left\lfloor \frac{I_i^x}{\bar{d}_i} \right\rfloor \right),
\end{aligned} \tag{6.12}$$

where, complementing the explanation in Section 3.3, we call $I_i^x = I_i + p_{i,m}$. This final check is meant to avoid stopping the production of machines whose setup costs are much greater than the inventory cost.

This method allows for an optimal distribution of resources, effectively balancing the costs associated with lost sales, setup, production, and inventory holding. It enables strategic decision-making concerning which items to produce and the machines to use, thereby minimizing overall costs while fulfilling the demand to the maximum possible extent. While this heuristic is straightforward in its conceptualization, its application is potent, making it a highly adaptable instrument in the complex landscape of inventory management.

Algorithm 9 Eligibility Decision Rule

procedure ELIGIBILITY DR(θ)
  ▷ Uses a predefined set of parameters θ
  ▷ Output: Decision x_t
  Initialize the eligible items list: eligible_items $\leftarrow$ []
  Compute average demand $\bar{d}$ using standard Monte Carlo method
  Initialize the expected run-out of the inventory: expected_runout $\leftarrow [I_{i,t}/\bar{d}_i]_{i\in\mathcal{I}}$
  ▷ Identify eligible items and calculate their priority
  for $i \in \mathcal{I}$ **do**
    if expected_runout$[i] \leq \theta_1$ **then**
      priority $\leftarrow \theta_2 \frac{l_i}{\text{expected\_runout}[i]} + \theta_3 N_i + \theta_4 \frac{\bar{d}_i}{\max_m p_{i,m}}$
      eligible_items.$add((i, \text{priority}))$
  Sort eligible_items with respect to priority
  ▷ Initialize machine selection array
  $\mathbf{x} \leftarrow [0, \ldots, 0]$
  ▷ Select machine for each eligible item
  for each $(i, \_) \in$ eligible_items **do**
    M $\leftarrow$ set of machines producing item i
    if M != $\emptyset$ **then**
      $\hat{m} \leftarrow$ machine in M with the highest $f_{i,m}$
    else
      $\hat{m} \leftarrow$ free machine with lowest $\frac{f_{i,m}}{p_{i,m}-c_{i,m}}$
    $\mathbf{x}[\hat{m}] \leftarrow i$
  ▷ Assign items to empty machines if profitable
  for $m \in \mathcal{M}$ **do**
    if $\mathbf{x}[m] = 0$ and $\mathbf{M}_t[m] \neq 0$ **then**
      $\hat{i} \leftarrow \mathbf{M}_t[m]$
      $I_i^x \leftarrow I_i + p_{i,m}$
      if $f_{\hat{i},m} \geq \theta_4 h \left(\left\lfloor \frac{I_i^x}{\bar{d}_i} \right\rfloor + 1 \right) \left(I_i^x - \frac{\bar{d}_i}{2} \left\lfloor \frac{I_i^x}{\bar{d}_i} \right\rfloor \right)$ **then**
        $x[m] \leftarrow \hat{i}$
  return x_t

6.1.3 Lot-Sizing Cooperative Multi-Agent Adjustment

In the Lot-Sizing Cooperative Multi-Agent Adjustment (LSCMA) methodology, we engage a baseline agent that can be a pre-trained RL algorithm or another technique. The baseline agent gives an initial set of actions, which we term "recommendations", denoted as X^b. In addition, we enlist M RL agents, termed "sub-agents", where each sub-agent corresponds to an individual machine. These sub-agents endeavor to enhance the initial recommendations offered by the baseline agent.

The objective for each of these sub-agents is to find a policy that minimizes the contribution specific to each agent, as depicted in the following formulation:

$$C_{m,t}(S_t, X_t) = \sum_{i=1}^{I} [f_{i,m}\delta_{i,m,t} + (h_i I_{i,t} + l_i z_{i,t})] \tag{6.13}$$

The state that each sub-agent considers is (S_t, X^b), i.e., the state of the system augmented with the recommendation.

In the LSCMA approach, each sub-agent is tasked with deciding between adhering to the recommended action provided by the baseline agent or opting for the baseline agent's previous action. This decision-making process is formulated as follows:

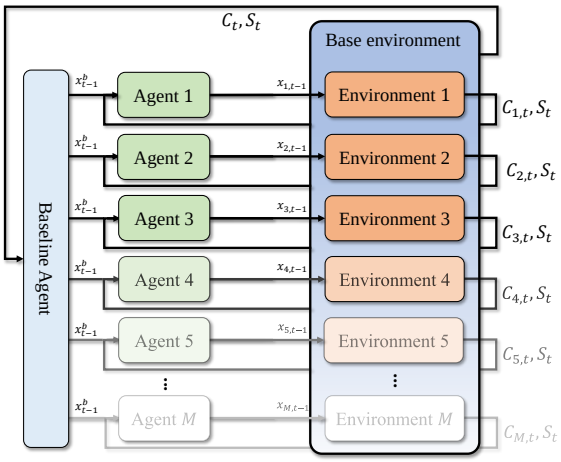
$$x_{m,t} = \begin{cases} x^b_{m,t-1} \text{ if } x^-_{m,t} = 1 \\ x^b_{m,t} \text{ if } x^-_{m,t} = 0 \end{cases} \tag{6.14}$$

In this formula, $x^-_{m,t}$ signifies the action selected by the sub-agent corresponding to machine m at time t. If the chosen action is $x^-_m = 1$, the sub-agent follows the previous setup action, whereas if the selected action is $x^-_m = 0$, it aligns with the current baseline setup. This mechanism confines the decision-making capacity of each sub-agent to a choice between two distinct actions, leading to a reduced action space. This compression of the action space streamlines the convergence process and reduces the necessity for extensive exploration in the quest for optimal behavior.

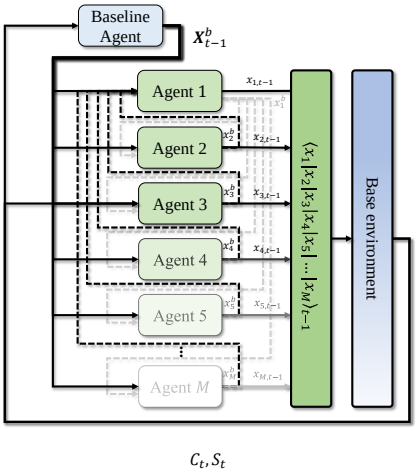
The training algorithm for the LSCMA approach is outlined in Algorithm 10 and visually depicted in Figure 10a. In this framework, each agent operates independently. Hence, we illustrate a collection of sub-environments nested within the overarching base environment. These sub-environments regulate the interactions of each respective agent. This approach emphasizes the independent yet cooperative nature of the multi-agent system in the LSCMA training process.

Figure 10: Graphical representation of the training the and testing procedure of the LSCMA.

(a) LSCMA training



(b) LSCMA test



Algorithm 10 LSCMA training

procedure LSCMA TRAINING(N, ϕ)
  ▷ Total number of iterations N
  ▷ Initial parameters of base agent ϕ
  ▷ Output: Trained parameters: θ_N
  Initialize the parameters $\theta_{m,t}$ of the neural networks $X^{m,\pi}$ of each for the m agents.
  Initialize the parameters ϕ of the baseline RL agent $X^{b,\pi}$
  Initialize the historical memory: $\mathbf{M}^h$
  Set the iteration counter $i = 1$
  for $i \in (0, N]$ **do**
    Initialize the environment and get a random initial state: $s_0 \in \mathcal{S}_0$
    Generate a sample path for the demand $d, t = [1, \ldots, T]$
    for $t \in (0, T]$ **do**
      for $m \in (0, M]$ **do**
        $x_{b,t} \leftarrow X^{b,\pi}(s_t, \phi)$
        $x_{m,t} \leftarrow X^{m,\pi}(s_t, x_{b,t}, \theta_{m,i})$
        ▷ Combine each agent action:
        $x_t \leftarrow (x_{1,t}|x_{2,t}|...|x_{m,t})$
        ▷ Generate the next pre-decision state:
        $s_{t+1} \leftarrow S^M(s_t, x_t)$
      $\mathbf{M}^h \leftarrow \mathbf{M}^h \cup (s_t, x_t, c_t, s_{t+1})$
    ▷ Sample the batch Z from the historical memory
    $Z \leftarrow$ Sample($\mathbf{M}^h$)
    Update the policy and the value networks parameters $\theta_{m,i}$ of each agent the model-free RL method using Z .
  return $\theta_{m,i}$

In the testing phase, the cooperative agents utilize their environment observations alongside the decisions of the baseline agent to formulate their first decision that does not consider the other sub-agents decisions. This procedure is iteratively executed with each sub-agent now considering both the decisions of its counterparts and the shared observations to formulate their subsequent decisions. These decisions are then concatenated into a unified decision array and given to the environment. In response, the environment returns the subsequent state and contribution, which the baseline agent observes to generate the subsequent decision. This iterative cycle of decision-making and observation,

delineated in detail in Algorithm 11, is also visually depicted in Figure 10b.

Algorithm 11 LSCMA testing

procedure LSCMA TESTING($N, \theta_N, \phi, \mathbf{S}$)
▷ Total number of iterations N
▷ Trained parameters of neural networks θ_N
▷ Initial parameters of base agent ϕ
▷ Episode list of observed states $\mathbf{S}^h$
Initialize the parameters $\theta_m \leftarrow \theta_{m,N}$ of the neural networks π for each of the m agents.
Initialize the parameters ϕ of the neural network of the baseline agent π^b
Initialize the environment and get a random initial state: $s_0 \in \mathcal{S}_0$
Generate a sample path for the demand $d, t = [1, \ldots, T]$.
for $t \in 0, 1, 2, ..., T-1$ **do**
    for $m \in 0, 1, 2, ..., M$ **do**
        $x_{b,t} \leftarrow X_b^\pi(s_t, \phi)$
        $x_{m,t} \leftarrow X_m^\pi(s_t, x_{b,t}, \theta_{m,N})$
    for $m \in 0, 1, 2, ..., M$ **do**
        $x_{m,t} \leftarrow X^\pi(s_t, x_t, \theta_{m,N})$
    ▷ Combine each agent action:
    $x_t \leftarrow (x_{1,t}|x_{2,t}|...|x_{m,t})$
    ▷ Generate the next pre-decision state:
    $s_{t+1} \leftarrow S^M(s_t, x_t)$

6.2 Experimental results for the Stochastic Discrete Lot-Sizing Problem

This section presents the results of experiments conducted to test the applicability of Reinforcement Learning (RL) and Approximate Dynamic Programming (ADP) methods in the Stochastic Discrete Lot-Sizing Problem (SDLSP). Unlike the previous experiments in the ASATP (discussed in Chapter 5, Section 5.2), the results suggest that RL methods may be a more promising approach to solving the SDLSP in specific scenarios. Notably, the same RL methods used in the previous problem instances were employed here, indicating the versatility of RL as a decision problem solver. However, the generality of RL also comes with a tradeoff in performance, which was evident in certain experimental settings.

Additionally, some of the best performances were achieved using the ADP framework, which is also based on the Hamilton-Jacobi-Bellman equations.

Our empirical analysis is organized into three distinct experiments, each addressing a unique problem configuration and size. In the initial series of experiments, detailed in Subsection 6.2.2, we confront a scenario involving a single machine and two items. This simple setting permits the computation of an optimal solution using Value Iteration (VI) that we use as a benchmark. This benchmark allows us to compare the Decision Rule (DR), Advantage Actor-Critic (A2C), Proximal Policy Optimization (PPO), Multi-Stage (MS) (Subsection 3.3.1), and Approximate Dynamic Programming (ADP) against each other. As there is only one machine, the multi-agent approach of Lot-Sizing Cooperative Multi-Agent Adjustment (LSCMA) is irrelevant and hence excluded.

The subsequent series of experiments, delineated in Subsection 6.2.3, addresses instances with up to 5 machines and 15 items. The increased complexity of this setting prohibits the use of exact techniques such as Value Iteration. Consequently, we evaluate the performance of DR, A2C, PPO, MS, ADP, and LSCMA by employing the Perfect Information agent as a performance standard.

Finally, in the third series of experiments outlined in Subsection 6.2.4, we tackle large-scale instances encompassing up to 10 machines and 25 items. In this context, using Perfect Information as a baseline for comparison is impractical due to the extensive scale of the mathematical model. Therefore, we compare the absolute costs of DR, A2C, PPO, MS, ADP, and LSCMA.

All experiments were conducted on a system equipped with an AMD Ryzen 5 5600X 6-Core Processor operating at 3.70 GHz, an RTX 3060 (12GB) graphics card, and 32GB of RAM. The experimental code was developed in Python 3.6. The Deep Reinforcement Learning (DLR) algorithms were implemented using the Pytorch (PASZKE et al., 2019) and Stable Baselines3 (RAFFIN et al., 2021) libraries. All models were solved using the Python3 APIs of Gurobi v9.5.0. The complete code is publicly accessible via our online repository[2].

Here are additional details regarding the configurations of the techniques employed in this study:

- For the MS method, a branching factor of $[4, 4, 2, 2]$ is applied. To mitigate the potential inaccuracies caused by rough Monte Carlo scenario generation, we reduce

[2] ⟨https://github.com/EdoF90/discrete_lot_sizing⟩

scenarios per the methodology described by (HEITSCH; RÖMISCH, 2003). Due to the potentially high computational demands for some instances, a time limit of 5 minutes is set for the solution of the model.

- For the DR method, the parameters $\theta_1, \theta_2, \theta_3, \theta_4$, and θ_5 are optimized using Particle Swarm Optimization (KENNEDY; EBERHART, 1995).
- In the LSCMA approach, the DR method serves as the baseline policy, and the PPO method is employed for each sub-agent.
- Both the PPO and A2C techniques underwent training for 50,000 episodes. However, the number of steps iterated varied based on the specific environment configuration[3].
- We aim to give robustness to the results by calculating the average cost from the 100 different episodes sample. Furthermore, we compute the standard deviation to capture the variation across these 100 episodes.

6.2.1 Instance generation

To generate test scenarios, we establish parameters including the number of items I, machines M, time steps T, and maximum inventory $I_{\max}$. We randomly determine the initial inventory for each item, ranging between 0 and $I_{\max}$. Following the methodology in Beraldi et al. (2005), we assign values for h_i, l_i, and $f_{i,m}$ (defined in Subsection 3.3.1) by randomly generating numbers from uniform distributions within the intervals $[0, 1]$, $[5, 10]$, and $[1, 5]$, respectively.

For the production matrix ($p_{i,m}$), we randomly select $2|\mathcal{I}|/|\mathcal{M}|$ items for each machine. The production capacity for these items is randomly drawn from a uniform distribution between 10 and 20. For the remaining items, the production capacity is set to zero, i.e., $p_{i,m} = 0$. This setup ensures that multiple machines can produce the same item, preventing the problem from being decomposable. Subsequently, we verify that each row of the $p_{i,m}$ matrix has at least one non-zero element, guaranteeing that there is at least one machine capable of producing each item. If this condition is not met, two random components are assigned a value randomly drawn from a uniform distribution between 10 and 20.

Lastly, the demand is generated following a specific distribution, which we elaborate

[3] For a detailed list of hyperparameters used for PPO and A2C, refer to Appendix A

on in the respective subsections. This instance generation method results in various test scenarios, enabling a thorough evaluation of the proposed techniques.

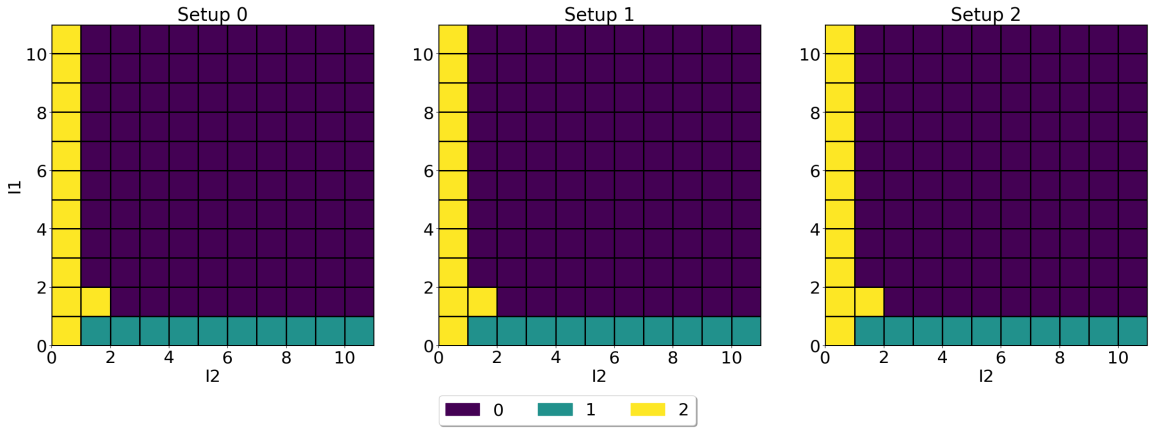
6.2.2 Comparing techniques for small size instance

This subsection shares the computational outcomes for small instances, defined by $I = 2$, $M = 1$, and $T = 10$. We have set the production, setup costs, setup loss, and inventory costs as equal for both items, with the distinction that the lost sales cost for item 2 is twice that of item 1. These settings allow us to compute the exact solution using Value Iteration and provide a clear visualization of policy and value function.

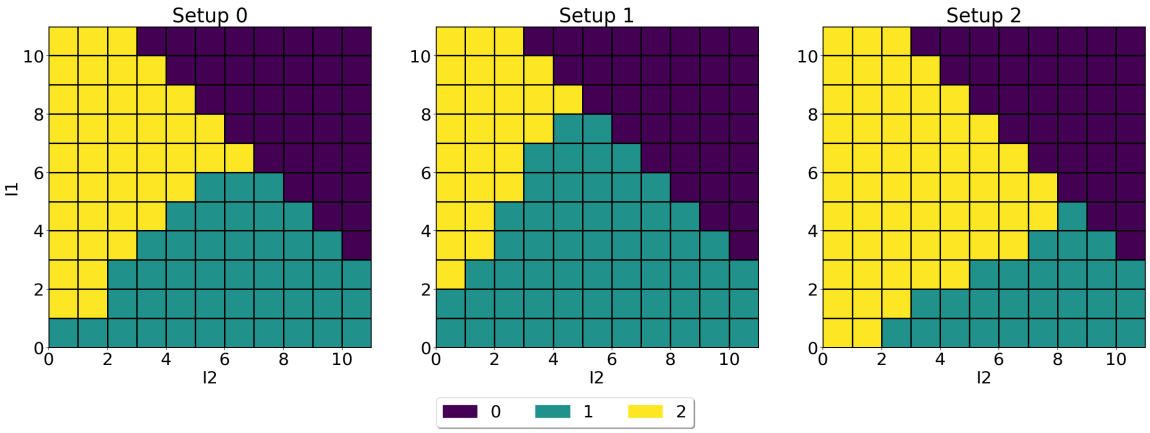
Figure 11 depicts the optimal policy computed via Value Iteration for various demand distributions. Specifically, the demand probability distribution adopted is the binomial distribution with parameters $n = 3$ and $p = \frac{1}{3}$ is employed for Figure 11a, while we utilize $n = 5$, $p = 0.4$ for Figure 11b.

Figure 11: The figure presents the comparison of Value Iteration (VI) policy visualizations with two different demand levels (lower and higher demand levels). The axes represent the inventory levels of item 1 and item 2, respectively. Each coordinate in the plot is colored according to the action taken for a specific combination of inventory levels for item 1 and item 2.

(a) VI policy - lower demands



(b) VI policy - higher demands



The policy is visually represented using three tables, each corresponding to a different setup. Every table comprises $I_{\max} \times I_{\max}$ cells, each denoting a possible inventory level. Each cell is color-coded to indicate the corresponding optimal action. Specifically, the green color (color 1 in the legend) represents the production of item 1, the yellow color (color 2 in the legend) represents the production of item 2, and the blue color represents the idle state (color 0 in the legend). It is observable that the optimal policy shares a consistent pattern: when an item's inventory is low, its production is initiated, and if both inventories are low, production of item 2 is prioritized due to its higher lost sales cost.

Additionally, it can be noted from Figure 11b that the setup significantly influences the optimal policy. For example, when the machine is set to produce item 1 (as depicted in the second table), the number of cells where production continues exceeds that of the other two tables. A similar, nearly symmetrical pattern occurs when the machine produces item 2 (as seen in the third table).

These visual representations provide valuable insights into the structure of the optimal policy. Subsequently, we compare the methodologies using a binomial distribution for item demand with $n = 3$ and $p = \frac{1}{3}$. The outcomes of these experiments are presented in Table 6. As explained before, we compute the average and the standard deviation of all types of costs in the sample of 100 episodes. Upon examination, MS, PPO, and ADP exhibit performance closely aligned with VI, although PPO has a considerably larger standard deviation than the other two methods. The DR method performs quite admirably, with an average gap of 6% (first line in the column "Total Cost %" of the Table 6). A2C, on the other hand, yields the least satisfactory results, notably divergent from the other methods. Its subpar performance is attributed to its propensity for underproduction, which results in substantial lost sales costs.

Table 6: Average total costs, holding, lost sales, and setup costs are a percentage of the Value Iteration for the one machine and two items setting. The Experiment Scenario column indicates the number of items by the number followed by the letter "I", the number of machines with the number followed by the letter "M", the number of time steps in an episode next to "T", and the maximum number of each item in the inventory after the notation "$I_{\max}$". We highlight the lowest total costs in bold.

Experiment Scenario	**Algorithm**	**Total Costs %**	**Holding Costs %**	**Lost Sales Costs %**	**Setup Costs %**
I2 M1 T20 $I_{\max}$10	DR	6±71	-12±25	100±479	0±94
	ADP	2±12	-2±5	3±2	0±4
	A2C	114±172	-31±31	992±1181	-54±120
	PPO	1±46	-3±21	18±301	8±67
	MS	**0±8**	0±4	0±49	0±17

6.2.3 Comparing techniques for medium size instances

This section presents the computational results for the medium instances ($I = 4M = 2, T = 10$, $I = 10M = 5, T = 10$, and $I = 15M = 5, T = 10$). For those instances, VI cannot be applied due to dimensionality. Thus, we use the PI agent (explained in

Subsection 3.3.1) as the benchmark. The distribution of item demand is given by the binomial distribution with $p = 0.4$ and $n = 4$ for the three instances. Table 7 shows the average and standard deviation of the costs.

Table 7: Average total costs, holding, lost sales, and setup costs percentage of the Perfect Information agent for the medium size scenarios. The Experiment Scenario column indicates the number of items by the number followed by the letter "I", the number of machines with the number followed by the letter "M", the number of time steps in an episode next to "T", and the maximum number of each item in the inventory after the notation "$I_{\max}$". We highlight the lowest total costs in bold.

Experiment Scenario	**Algorithm**	**Total Cost %**	**Holding Costs %**	**Lost Sales Costs %**	**Setup Costs %**
I4 M2 T10 $I_{\max}10$	DR	159±159	-59±39	465±493	104±213
	ADP	64±102	-38±41	231±274	16±193
	A2C	208±131	9±41	730±396	-79±162
	PPO	**47±87**	3±44	115±187	29±197
	MS	131±140	-50±39	310±369	161±221
	LSCMA	59±95	-7±43	207±253	-13±182
I10 M5 T10 $I_{\max}10$	DR	94±100	-29±29	425±453	67±126
	ADP	102±97	-23±33	704±458	-56±107
	A2C	138±89	44±35	732±440	-51±140
	PPO	145±108	3±43	840±509	-42±119
	MS	89±84	-29±31	226±253	155±151
	LSCMA	**83±90**	4±36	421±394	3±115
I15 M5 T10 $I_{\max}10$	DR	86±73	-41±26	155±130	67±99
	ADP	**37±48**	-13±27	85±76	-19±104
	A2C	93±57	25±38	193±103	-62±91
	PPO	67±46	23±31	95±70	51±103
	MS	61±53	-33±24	82±77	115±111
	LSCMA	65±60	3±34	132±102	-19±95

In the $I4M2$ scenarios, PPO is the most effective, closely followed by LSCMA and ADP. MS performance is impacted by the time limit, which restricts its ability to reach the optimal solution. While DR results are less satisfactory, LSCMA is the second-best method, demonstrating that the multi-agent strategy can considerably enhance the initial policy. PPO identifies a policy that minimizes the most impactful costs: setup and lost

sales costs. This balance is readily visible in Figures 12a, and 12b, which track the changes in lost sales cost, and holding cost, over 100 test scenarios. While PPO does incur a higher holding cost, it also reduces lost sales costs, suggesting a favorable trade-off. The setup costs across all models are comparable, but PPO's costs remain more stable over time. Considering only the setup costs, PPO once again outperforms the others. As such, this particular scenario best supports PPO's policy.

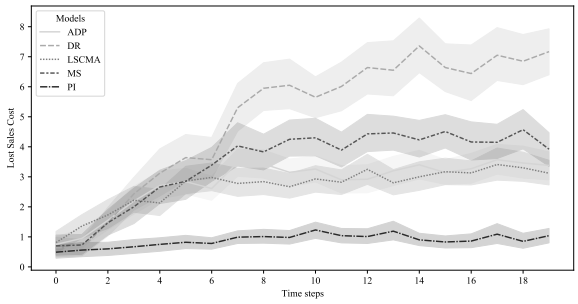
In scenarios involving 10 items and 5 machines, LSCMA achieves the best policy, closely followed by DR and MS. Examining the costs, ADP substantially reduces setup costs at the cost of increased lost sales costs. In contrast, LSCMA keeps setup costs lower than DR and maintains a relatively low lost sales cost, contributing significantly to its superior performance. The holding costs across the three methods are similar, with LSCMA's slightly higher.

In the scenarios with 15 items and 5 machines, the importance of lost sales cost elevates, thus positioning the ADP policy as the most effective, outstripping even the LSCMA policy. The ADP policy incurs a slightly reduced setup cost while securing the lowest lost sales cost. Although DR excels in managing holding costs, it leads to substantial lost sales and setup costs, suggesting a frequent interruption in production. LSCMA rectifies this limitation of DR, delivering enhanced results across all three cost areas. An inspection of Figures 12e and 12f reveals a different narrative for the MS policy. Despite having a marginally elevated setup cost, MS policy maintains the lowest lost sales cost at the beginning of the episode. On the contrary, the ADP policy manifests beneficial results regarding setup costs and lost sales. While DR stands as the optimal model in terms of holding cost, it falls short in effectively managing lost sales. This discrepancy underscores the need for a balanced approach to cost management in inventory scenarios.

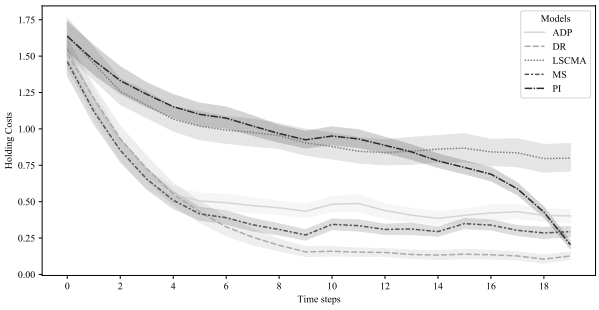
In conclusion, this experiment suggests that while PPO provides a robust solution in small-scale scenarios, techniques such as LSCMA or ADP yield better results in medium-sized ones. This discrepancy could be attributed to convergence issues that arise as the state space dimension expands. Furthermore, ADP and LSCMA might deliver superior outcomes, especially in reducing setup and inventory costs, because they exploit the limited number of decision steps in these experiments.

Figure 12: Cost results for the models ADP, DR, LSCMA, MS, and the PI in medium size instances.

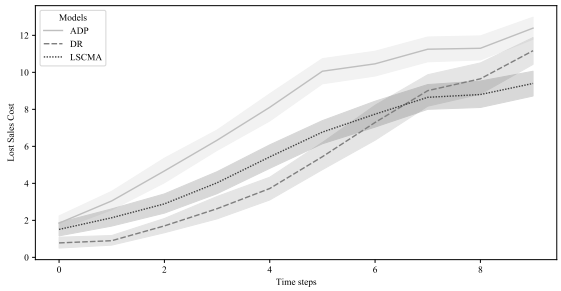
(a) Lost sale cost for 4 items, 2 machines and T = 20 case.



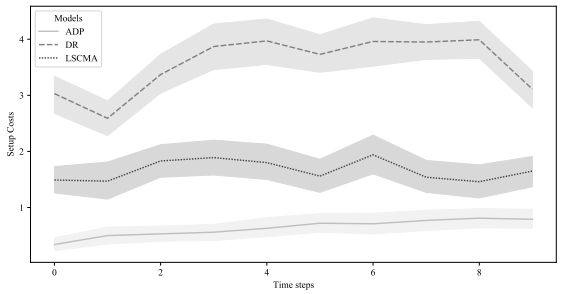
(b) Holding cost for 4 items, 2 machines and T = 20 case.



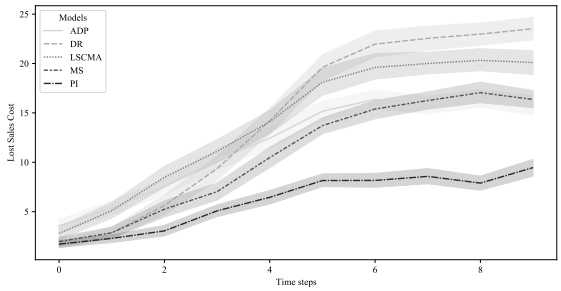
(c) Lost sale cost for 10 items, 5 machines and T = 10 case.



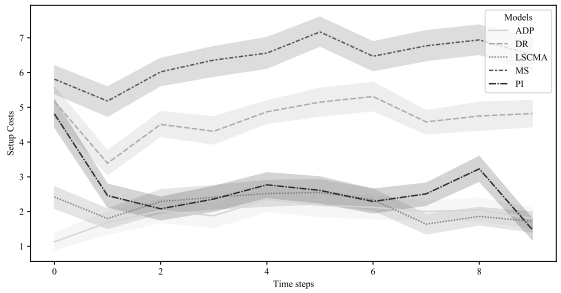
(d) Setup cost for 10 items, 5 machines and T = 10 case.



(e) Lost sale cost for 15 items, 5 machines and T = 10 case.



(f) Setup cost for 15 items, 5 machines and T = 10 case.



In our study, we undertook a series of experiments that utilized a more significant number of time steps per episode. Our objective was to secure more comprehensive and nuanced results. By extending the number of time steps, we could capture more detailed variations and behavior trends over each episode. This extension provided a richer dataset for analysis, allowing us better to understand our models' dynamics and their performance. These experiments, involving extended time steps and higher dimensionality state spaces,

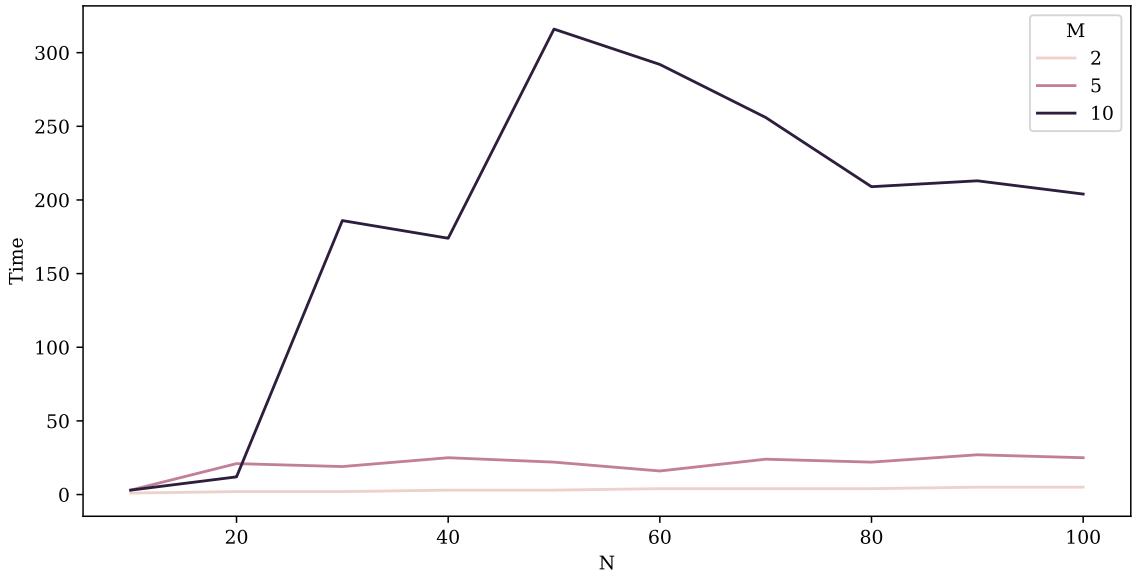
were designed to push the boundaries of our models and offer insights that could aid in further developing and refining our techniques.

6.2.4 Big size instances

The multistage approach, while comprehensive, is constrained by its practicality in numerous scenario configurations due to the extensive staging required for stability. We utilize the methodologies outlined in Heitsch and Römisch (2003) to model the scenario tree. Notably, the demand $d_i^{[0]}$ is not incorporated in constraint (3.22), which instead accounts for $d_i^{[n]}$ for each $n \in \mathcal{N}^+$. As the action space expands, the time necessary to generate each action escalates significantly.

Figure 13 illustrates the surge in processing time correlating with the increased number of machines, as depicted by each of the three curves. We note a spike in processing time as the number of items escalates from 10 to 30. Beyond 20 items, the increase in processing time levels off for scenarios with 2 and 5 machines. For scenarios with 10 machines, processing time continues to grow until reaching 50 machines, after which it diminishes and stabilizes. These curves suggest a connection between processing time, the number of machines, and items. However, as the number of items far exceeds the number of machines, processing time reaches a plateau.

Figure 13: Processing time for executing 10 decisions using the multistage agent across different numbers of machines (M) and items (N)



With the increase in decision steps and the growth of state-space dimensionality, certain techniques became impractical due to processing time constraints. Techniques

such as the PI agent (our comparison baseline) and the multistage agent could not be evaluated due to the extensive decision setups and the large dimensionality of the state space.

This section presents results from instances where we have $I = 15, M = 5$ and $I = 25, M = 10$, with a time horizon extending to 100 time steps. When dealing with these instances, the MS method becomes excessively time-consuming, even when time limits are imposed. For instance, in scenarios with $I15$ $M5$ (the smaller ones), computations for a single test run with 100 steps rarely conclude before the 5-minute mark, thus leading to considerable time requirements when repeated 100 times with 100 time steps. Therefore, we limit our comparison to the DR, ADP, A2C, PPO, and LSCMA methods for the $I15$ $M5$ instances. Moreover, the branch-and-bound procedure implemented in the ADP method is also overly time-consuming for the $I25$ $M10$ scenarios, leaving us with DR, A2C, PPO, and LSCMA as viable methods.

In these instances, demand follows a binomial distribution with parameters $p = 0.4$ and $n = 4$ across all settings, except in scenarios involving $I25$ $M10$, where we increase the parameter n to 20 to simulate a rise in demand. Table 8 provides the average costs for these instances. Of all the methods tested, ADP stands out, producing the best results for all instances that include 15 items.

To illustrate the performance of these methods, we graphed the setup, lost sales, and holding costs throughout the episode in Figures 14a, 14b, and 14c. Interestingly, the PPO method often maintains the current setup, increasing lost sales and holding costs. In contrast, the ADP method effectively reduces holding and lost sales costs.

Table 8: Average total costs, holding, lost sales, and setup costs. Testing in big-size scenarios where the Perfect Information agent cannot provide the optimal solutions. The Experiment Scenario column indicates the number of items by the number followed by the letter "I", the number of machines with the number followed by the letter "M", the number of time steps in an episode next to "T", and the maximum number of each item in the inventory after the notation "$I_{\max}$". We highlight the lowest total costs in bold.

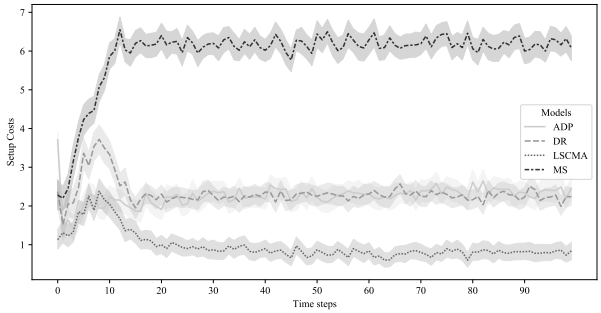
Experiment Scenario	**Algorithm**	**Total Costs**	**Holding Costs**	**Lost Sales Costs**	**Setup Costs**
I15 M5 T100 $I_{\max}$10	DR	2062±443	44±135	1966±647	52±88
	ADP	**1393±163**	112±110	888±217	393±59
	A2C	2000±250	327±72	1555±319	118±76
	PPO	2066±269	322±77	1737±367	7±70
	LSCMA	1988±319	243±97	1715±436	30±28
I15 M5 T100 $I_{\max}$100	DR	1982±384	58±122	1689±515	235±34
	ADP	**1472±169**	109±101	1137±253	226±26
	A2C	2129±270	320±70	1801±364	8±81
	PPO	1918±230	103±104	1705±337	110±79
	LSCMA	1981±343	229±95	1655±459	97±36
I25 M10 T100 $I_{\max}$100	DR	3290±922	129±253	2962±1397	199±272
	A2C	3609±470	436±130	3158±642	15±152
	PPO	3612±468	435±127	3158±641	19±191
	LSCMA	**3249±590**	451±166	2682±825	117±101

When comparing the results with $I = 15$ for $I_{\max} = 10$ and $I_{\max} = 100$, we can observe how the methods respond to changes in the dimensionality of the state space. Interestingly, the results are remarkably similar, indicating that all methods can effectively handle the enlarged state space. It is noteworthy that the increase in dimensionality impacts the computational time of ADP, given the decrease in the number of feasibility cuts.

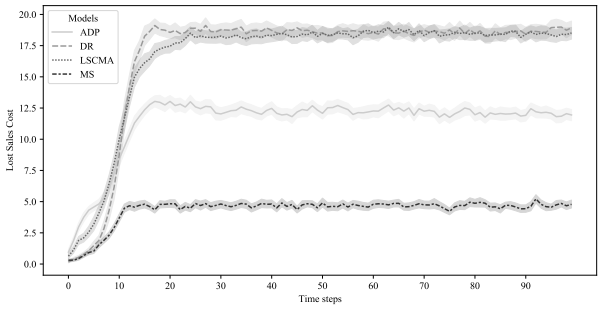
Finally, in scenarios with $I = 25$, the most successful method is LSCMA, which enhances the performance of DR and reduces its variance. Intriguingly, DR outperforms both A2C and PPO, suggesting that the learning algorithms of these two techniques begin to falter in high-dimensional spaces. Specifically, LSCMA mainly reduces lost sales costs by utilizing production capacity more efficiently.

Figure 14: Costs for the models ADP, DR, LSCMA in big size instances.

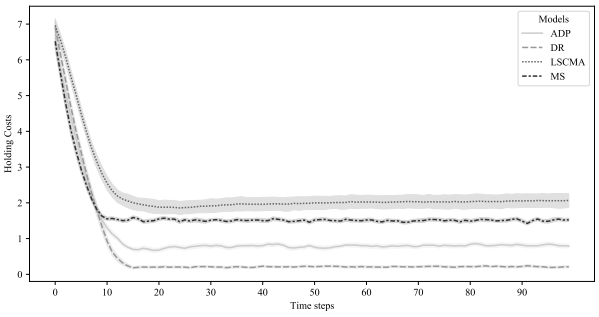
(a) Setup costs for 15 items, 5 machines and T = 100 case.



(b) Lost sales costs for 15 items, 5 machines and T = 100 case.



(c) Holding costs for 15 items, 5 machines and T = 100 case.



7 CONCLUSION

This thesis has explored the utilization of Reinforcement Learning, notably Deep Reinforcement Learning, in contrast with other methods such as ADP and Supervised Learning in addressing two separate practical issues: asset trading with emphasis on the Active Single-Asset Trading Problem and operations research with a focus on the Stochastic Discrete Lot-Sizing Problem. We recall the contributions listed in Chapter 1, reinforcing them here.

Our investigation into active trading systems has shed light on the potential limitations of Deep Reinforcement Learning in financial decision-making contexts. To test the limitations in Deep Reinforcement Learning and provide a contrast with Supervised Learning approaches, we introduced the RSLSTM-A, a state-of-the-art Supervised Learning method for decision-making in active trading. This proposed method is the outcome of previous combined investigations published in (FELIZARDO et al., 2019; PAIVA et al., 2022; URBINATE; FELIZARDO; DEL-MORAL-HERNANDEZ, 2022) that led to the architecture of the proposed method and the experiment setup. Our proposed method has consistently outperformed nearly state-of-the-art Deep Reinforcement Learning methods, as shown in this thesis and the results of the published work of Felizardo et al. (2022a). These findings emphasize the pivotal role of context-specific adjustments when applying DRL techniques. As shown in Chapter 5, our proposed method outperformed the DRL contenders in most assessed metrics, further validating its effectiveness. Noticing that Reinforcement Learning may not fit this specific problem instance, we tested other decision problems in quantitative finance that resulted in another publication (FELIZARDO; MATSUMOTO; DEL-MORAL-HERNANDEZ, 2022b) that was not detailed in this thesis. In that publication, the problem approached has a transition function that depends on the action and the state, making model-free Reinforcement Learning an adequate approach.

Another problem instance that also fitted Reinforcement Learning methods was in operations research. In the second part of this thesis, we extended the use of Deep Re-

inforcement Learning methods in Stochastic Discrete Lot-Sizing Problem. We previously tested the usage of Reinforcement Learning in other operations research problems (GIOIA; FELIZARDO; BRANDIMARTE, 2022; GIOIA; FELIZARDO; BRANDIMARTE, 2023), and we moved to the Stochastic Discrete Lot-Sizing Problem that presented as an adequate challenge for the method. We have offered a new environment model for Reinforcement Learning applications and a priority-based decision-making technique. Further, our branch and bound tree heuristics for Approximate Dynamic Programming and our Lot-Sizing Cooperative Multi-Agent Adjustment (LSCMA) proposed method have presented promising ways of integrating conventional techniques with Reinforcement Learning. The results presented in Chapter 6 provide some pieces of evidence of the efficiency of our proposed methods compared to more mainstream methods of operations research.

The unifying notation and mathematical framework adopted in this thesis have enabled the effective modeling of the explored problems, and we hope to facilitate more transparent communication between the research communities of stochastic optimal control and Reinforcement Learning. We hope this work will inspire future research to leverage further the benefits of Deep Reinforcement Learning in finance and operations research.

In light of the findings presented in this thesis, several promising research paths have emerged that are worth further exploration in both active trading systems and operations research.

For the Active Single-Asset Trading Problem, expanding the range of assets employed could be a critical next step. Investigating a broader and more diverse range of assets would provide a more rigorous evaluation of our methods' robustness and generalization capabilities. Additionally, we propose exploring the use of different time series classification methods. While our current approach using RSLSTM-A has shown promising results, many other techniques within the vast landscape of machine learning and time series analysis could further enhance the performance. Another interesting direction would be using synthetic time series data in our evaluations, as we briefly explored in Urbinate, Felizardo and Del-Moral-Hernandez (2022). This approach would allow us to create controlled experimental conditions, enabling us to systematically evaluate the impact of the Reinforcement Learning solutions on the decision-making process, particularly in the presence of transaction costs. Synthetic data could provide a clearer understanding of our models' strengths and weaknesses and guide the refinement of our methodologies.

For the Stochastic Discrete Lot-Sizing Problem, exploring higher dimensions and com-

plexity could make Reinforcement Learning solutions increasingly important. As we increase the dimensionality and complexity of the problems, traditional methods may start to falter, and the advantage of Deep Reinforcement Learning could become more pronounced. Moreover, we need more comprehensive investigations of Perfect Information agents over shorter periods and under different demand scenarios. This exploration would enable us to understand better how different contexts and conditions influence the effectiveness of our models and guide future modifications and improvements. Lastly, based on the approaches developed in this work, integrating other model-free methods in lot-sizing problems could be a fruitful area of exploration. We have demonstrated that combining Reinforcement Learning with traditional methods can yield robust solutions. Therefore, further research into integrating different model-free methods with lot-sizing problems could yield similarly impactful results.

In conclusion, much remains to be explored, and many challenges to be addressed in applying DRL to real-world problems. This thesis has taken crucial steps forward, but the path ahead is full of potential for exciting and influential advancements in the field of Reinforcement Learning. We look forward to seeing the progress that future research will bring.

REFERENCES

ABADI, M.; AGARWAL, A.; BARHAM, P.; BREVDO, E.; CHEN, Z.; CITRO, C.; CORRADO, G. S.; DAVIS, A.; DEAN, J.; DEVIN, M.; GHEMAWAT, S.; GOODFELLOW, I.; HARP, A.; IRVING, G.; ISARD, M.; JIA, Y.; JOZEFOWICZ, R.; KAISER, L.; KUDLUR, M.; LEVENBERG, J.; MANE, D.; MONGA, R.; MOORE, S.; MURRAY, D.; OLAH, C.; SCHUSTER, M.; SHLENS, J.; STEINER, B.; SUTSKEVER, I.; TALWAR, K.; TUCKER, P.; VANHOUCKE, V.; VASUDEVAN, V.; VIEGAS, F.; VINYALS, O.; WARDEN, P.; WATTENBERG, M.; WICKE, M.; YU, Y.; ZHENG, X. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Software available from tensorflow.org.

ABOUSSALAH, A. M.; LEE, C.-G. Continuous control with Stacked Deep Dynamic Recurrent Reinforcement Learning for portfolio optimization. *Expert Systems with Applications*, Elsevier Ltd, v. 140, p. 112891, 02 2020. ISSN 09574174.

ALLEN, F.; KARJALAINEN, R. Using genetic algorithms to find technical trading rules. *Journal of Financial Economics*, v. 51, n. 2, p. 245–271, 1999. ISSN 0304-405X.

ALLEN, M. P. *Understanding Regression Analysis*. Boston, MA: Springer US, 1997. 176–180 p. ISBN 978-0-585-25657-3.

ALMAHDI, S.; YANG, S. Y. An adaptive portfolio trading system: A risk-return portfolio optimization using recurrent reinforcement learning with expected maximum drawdown. *Expert Systems with Applications*, v. 87, p. 267–279, 2017. ISSN 0957-4174.

ALMAHDI, S.; YANG, S. Y. A constrained portfolio trading system using particle swarm algorithm and recurrent reinforcement learning. *Expert Systems with Applications*, Elsevier Ltd, v. 130, p. 145–156, 2019. ISSN 09574174.

AN, B.; SUN, S.; WANG, R. Deep reinforcement learning for quantitative trading: Challenges and opportunities. *IEEE Intelligent Systems*, v. 37, n. 2, p. 23–26, 2022.

BAI, Y.; KADAVATH, S.; KUNDU, S.; ASKELL, A.; KERNION, J.; JONES, A.; CHEN, A.; GOLDIE, A.; MIRHOSEINI, A.; MCKINNON, C.; CHEN, C.; OLSSON, C.; OLAH, C.; HERNANDEZ, D.; DRAIN, D.; GANGULI, D.; LI, D.; TRAN-JOHNSON, E.; PEREZ, E.; KERR, J.; MUELLER, J.; LADISH, J.; LANDAU, J.; NDOUSSE, K.; LUKOSUITE, K.; LOVITT, L.; SELLITTO, M.; ELHAGE, N.; SCHIEFER, N.; MERCADO, N.; DASSARMA, N.; LASENBY, R.; LARSON, R.; RINGER, S.; JOHNSTON, S.; KRAVEC, S.; SHOWK, S. E.; FORT, S.; LANHAM, T.; TELLEEN-LAWTON, T.; CONERLY, T.; HENIGHAN, T.; HUME, T.; BOWMAN, S. R.; HATFIELD-DODDS, Z.; MANN, B.; AMODEI, D.; JOSEPH, N.; MCCANDLISH, S.; BROWN, T.; KAPLAN, J. *Constitutional AI: Harmlessness from AI Feedback*. [S.l.]: arXiv, 2022.

BAIRD, L. Advantage updating. *Technical Report WL-TR-93-1146*, 1993.

BELLMAN, R. *Dynamic Programming*. 1. ed. Princeton, NJ, USA: Princeton University Press, 1957.

BERALDI, P.; GHIANI, G.; GRIECO, A.; GUERRIERO, E. Fix and relax heuristic for a stochastic lot-sizing problem. *Computational Optimization and Applications*, Springer Science and Business Media LLC, v. 33, n. 2-3, p. 303–318, out. 2005.

BESSEMBINDER, H.; CHAN, K. The profitability of technical trading rules in the asian stock markets. *Pacific-Basin Finance Journal*, v. 3, n. 2, p. 257–284, 1995. ISSN 0927-538X.

BORRAGEIRO, G.; FIROOZYE, N.; BARUCCA, P. The recurrent reinforcement learning crypto agent. *IEEE Access*, v. 10, p. 38590–38599, 2022.

BOUTE, R. N.; GIJSBRECHTS, J.; JAARSVELD, W. van; VANVUCHELEN, N. Deep reinforcement learning for inventory control: A roadmap. *European Journal of Operational Research*, 2021. ISSN 0377-2217.

BOYD, S.; VANDENBERGHE, L. *Convex Optimization*. [S.l.]: Cambridge University Press, 2004. Hardcover. ISBN 0521833787.

BRANDIMARTE, P. Multi-item capacitated lot-sizing with demand uncertainty. *International Journal of Production Research*, v. 44, p. 2997–3022, 2006.

BROCK, W.; LAKONISHOK, J.; LEBARON, B. Simple technical trading rules and the stochastic properties of stock returns. *The Journal of Finance*, American Finance Association, Wiley, v. 47, n. 5, p. 1731–1764, 1992. ISSN 00221082, 15406261.

CAMPBELL, J. Y.; LO, A. W.; MACKINLAY, A. *The Econometrics of Financial Markets*. [S.l.]: Princeton University Press, 1997. ISBN 9780691043012.

CARAPUÇO, J.; NEVES, R.; HORTA, N. Reinforcement learning applied to forex trading. *Applied Soft Computing*, v. 73, p. 783–794, 2018. ISSN 1568-4946.

CARTA, S.; CORRIGA, A.; FERREIRA, A.; PODDA, A. S.; RECUPERO, D. R. A multi-layer and multi-ensemble stock trader using deep learning and deep reinforcement learning. *Applied Intelligence*, v. 51, n. 2, p. 889–905, Feb 2021. ISSN 1573-7497.

CHOI, H.; RYU, S.; KIM, H. Short-term load forecasting based on resnet and lstm. In: *2018 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*. [S.l.: s.n.], 2018. p. 1–6.

CHOLLET, F. et al. *Keras*. 2015. ⟨https://keras.io⟩.

CHOPRA, N.; LAKONISHOK, J.; RITTER, J. R. Measuring abnormal performance: Do stocks overreact? *Journal of Financial Economics*, v. 31, n. 2, p. 235–268, 1992. ISSN 0304-405X.

CLARK, A. R.; CLARK, S. J. Rolling-horizon lot-sizing when set-up times are sequence-dependent. *International Journal of Production Research*, Taylor & Francis, v. 38, n. 10, p. 2287–2307, 2000.

DANG, Q.-V. Reinforcement learning in stock trading. In: THI, H. A. L.; LE, H. M.; DINH, T. P.; NGUYEN, N. T. (Ed.). *Advanced Computational Methods for Knowledge Engineering*. Cham: Springer International Publishing, 2020. p. 311–322. ISBN 978-3-030-38364-0.

DANTAS, S. G.; SILVA, D. G. Equity trading at the brazilian stock market using a q-learning based system. In: *2018 7th Brazilian Conference on Intelligent Systems (BRACIS)*. [S.l.: s.n.], 2018. p. 133–138.

DAS, T. K.; GOSAVI, A.; MAHADEVAN, S.; MARCHALLECK, N. Solving semi-markov decision problems using average reward reinforcement learning. *Management Science*, INFORMS, v. 45, n. 4, p. 560–574, 1999. ISSN 00251909, 15265501.

DENG, Y.; BAO, F.; KONG, Y.; REN, Z.; DAI, Q. Deep direct reinforcement learning for financial signal representation and trading. *IEEE Transactions on Neural Networks and Learning Systems*, v. 28, n. 3, p. 653–664, 2017.

DUAN, Y.; CHEN, X.; HOUTHOOFT, R.; SCHULMAN, J.; ABBEEL, P. Benchmarking deep reinforcement learning for continuous control. In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. [S.l.]: JMLR.org, 2016. (ICML'16), p. 1329–1338.

ECKLES, D.; KAPTEIN, M. Thompson sampling with the online bootstrap. *CoRR*, abs/1410.4009, 2014.

FAMA, E. F. Random Walks in Stock Market Prices. *Financial Analysts Journal*, 1965. ISSN 0015-198X.

FAMA, E. F.; FRENCH, K. R. Permanent and temporary components of stock prices. *Journal of Political Economy*, v. 96, n. 2, p. 246–273, 1988. ISSN 0022-3808.

FAWAZ, H. I.; FORESTIER, G.; WEBER, J.; IDOUMGHAR, L.; MULLER, P.-A. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, Springer Science and Business Media LLC, v. 33, n. 4, p. 917–963, 03 2019. ISSN 1573-756X.

FELIZARDO, L.; OLIVEIRA, R.; DEL-MORAL-HERNANDEZ, E.; COZMAN, F. Comparative study of bitcoin price prediction using wavenets, recurrent neural networks and other machine learning methods. In: *2019 6th International Conference on Behavioral, Economic and Socio-Cultural Computing (BESC)*. [S.l.: s.n.], 2019. p. 1–6.

FELIZARDO, L. K.; MATSUMOTO, E.; DEL-MORAL-HERNANDEZ, E. Solving the optimal stopping problem with reinforcement learning: an application in financial option exercise. In: *2022 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2022b. p. 1–8.

FELIZARDO, L. K.; PAIVA, F. C. L.; COSTA, A. H. R.; DEL-MORAL-HERNANDEZ, E. *Reinforcement Learning Applied to Trading Systems: A Survey*. [S.l.]: arXiv, 2022c.

FELIZARDO, L. K.; PAIVA, F. C. L.; GRAVES, C. de V.; MATSUMOTO, E. Y.; COSTA, A. H. R.; DEL-MORAL-HERNANDEZ, E.; BRANDIMARTE, P. Outperforming algorithmic trading reinforcement learning systems: A supervised

approach to the cryptocurrency market. *Expert Systems with Applications*, v. 202, p. 117259, 2022a. ISSN 0957-4174.

FENGQIAN, D.; CHAO, L. An adaptive financial trading system using deep reinforcement learning with candlestick decomposing features. *IEEE Access*, v. 8, p. 63666–63678, 2020.

FLEISCHMANN, B.; MEYR, H. The general lotsizing and scheduling problem. *Operations-Research-Spektrum*, v. 19, n. 1, p. 11–21, 03 1997. ISSN 1436-6304.

GEURTS, M.; BOX, G. E. P.; JENKINS, G. M. Time Series Analysis: Forecasting and Control. *Journal of Marketing Research*, 1977. ISSN 00222437.

GICQUEL, C.; MINOUX, M.; DALLERY, Y. Exact solution approaches for the discrete lot-sizing and scheduling problem with parallel resources. *International Journal of Production Research*, Informa UK Limited, v. 49, n. 9, p. 2587–2603, maio 2011.

GIJSBRECHTS, J.; BOUTE, R. N.; MIEGHEM, J. A. V.; ZHANG, D. J. Can deep reinforcement learning improve inventory management? performance on lost sales, dual-sourcing, and multi-echelon problems. *Manufacturing & Service Operations Management*, v. 24, n. 3, p. 1349–1368, 2022.

GIOIA, D. G.; FELIZARDO, L. K.; BRANDIMARTE, P. Inventory management of vertically differentiated perishable products with stock-out based substitution. *IFAC-PapersOnLine*, v. 55, n. 10, p. 2683–2688, 2022. ISSN 2405-8963. 10th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2022.

GIOIA, D. G.; FELIZARDO, L. K.; BRANDIMARTE, P. Simulation-based inventory management of perishable products via linear discrete choice models. *Computers & Operations Research*, v. 157, p. 106270, 2023. ISSN 0305-0548.

HAASE, K. *Lotsizing and Scheduling for Production Planning*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994. ISBN 978-3-642-45735-7.

HAMILTON, W. P. *The stock market barometer: study of its forecast value based on Charles H. Dow's theory of the price movement. With an analysis of the market and its history since 1897*. [S.l.]: Harper & Brothers, 1922. ISBN 0471247383.

HASSELT, H. van; GUEZ, A.; SILVER, D. Deep reinforcement learning with double q-learning. In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. [S.l.]: AAAI Press, 2016. (AAAI'16), p. 2094–2100.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Deep residual learning for image recognition. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2016. ISBN 9781467388504. ISSN 10636919.

HE, R.; LIU, Y.; WANG, K.; ZHAO, N.; YUAN, Y.; LI, Q.; ZHANG, H. Automatic cardiac arrhythmia classification using combination of deep residual network and bidirectional lstm. *IEEE Access*, v. 7, p. 102119–102135, 2019.

HEITSCH, H.; RÖMISCH, W. Scenario reduction algorithms in stochastic programming. *Computational Optimization and Applications*, v. 24, n. 2, p. 187–206, 02 2003. ISSN 1573-2894.

HENDERSON, P.; ISLAM, R.; BACHMAN, P.; PINEAU, J.; PRECUP, D.; MEGER, D. Deep Reinforcement Learning That Matters. In: *32nd AAAI Conf. on Artificial Intelligence (AAAI-18)*. [S.l.: s.n.], 2018.

HEZEWIJK, L. van; DELLAERT, N.; WOENSEL, T. van; GADEMANN, N. Using the proximal policy optimisation algorithm for solving the stochastic capacitated lot sizing problem. *International Journal of Production Research*, Taylor & Francis, v. 0, n. 0, p. 1–24, 2022.

HIRCHOUA, B.; OUHBI, B.; FRIKH, B. Deep reinforcement learning based trading agents: Risk curiosity driven learning for financial rules-based policy. *Expert Systems with Applications*, v. 170, p. 114553, 2021. ISSN 0957-4174.

HOLLAND, J. H. Adaptation**research reported in this article was supported in part by the national science foundation under grant dcr 71-01997. In: ROSEN, R.; SNELL, F. M. (Ed.). *Progress in Theoretical Biology*. [S.l.]: Academic Press, 1976. p. 263–293. ISBN 978-0-12-543104-0.

HÄMÄLÄINEN, P.; BABADI, A.; MA, X.; LEHTINEN, J. *PPO-CMA: Proximal Policy Optimization with Covariance Matrix Adaptation*. [S.l.]: arXiv, 2018.

JANS, R.; DEGRAEVE, Z. Modeling industrial lot sizing problems: a review. *International Journal of Production Research*, Taylor & Francis, v. 46, n. 6, p. 1619–1643, 2008.

JEONG, G.; KIM, H. Y. Improving financial trading decisions using deep Q-learning: Predicting the number of shares, action strategies, and transfer learning. *Expert Systems with Applications*, Elsevier Ltd, v. 117, p. 125–138, mar 2019. ISSN 09574174.

KANG, Q.; ZHOU, H.; KANG, Y. An Asynchronous Advantage Actor-Critic Reinforcement Learning Method for Stock Selection and Portfolio Management. In: *Proceedings of the 2nd International Conference on Big Data Research - ICBDR 2018*. New York, New York, USA: ACM Press, 2018. p. 141–145. ISBN 9781450364768.

KARMARKAR, U. S. Equalization of runout times. *Operations Research*, INFORMS, v. 29, n. 4, p. 757–762, 1981. ISSN 0030364X, 15265463.

KENNEDY, J.; EBERHART, R. Particle swarm optimization. In: *Proceedings of ICNN'95 - International Conference on Neural Networks*. [S.l.: s.n.], 1995. v. 4, p. 1942–1948 vol.4.

KIRAN, B. R.; SOBH, I.; TALPAERT, V.; MANNION, P.; SALLAB, A. A. A.; YOGAMANI, S.; PÉREZ, P. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, v. 23, n. 6, p. 4909–4926, 2022.

KIRK, D. *Optimal control theory: an introduction*. [S.l.]: Dover Publications, 2004.

KIRKPATRICK II, C. D.; DAHLQUIST, J. A. *Technical analysis: the complete resource for financial market technicians*. [S.l.]: FT press, 2010.

KUO, R.; CHEN, C.; HWANG, Y. An intelligent stock trading decision support system through integration of genetic algorithm based fuzzy neural network and artificial neural network. *Fuzzy Sets and Systems*, v. 118, n. 1, p. 21–45, 2001. ISSN 0165-0114.

LEI, K.; ZHANG, B.; LI, Y.; YANG, M.; SHEN, Y. Time-driven feature-aware jointly deep reinforcement learning for financial signal representation and algorithmic trading. *Expert Systems with Applications*, v. 140, p. 112872, 2020. ISSN 0957-4174.

LEWIS, F. L.; VRABIE, D. L.; SYRMOS, V. L. *Optimal Control*. [S.l.]: John Wiley & Sons, Inc., 2012. ISBN 9781118122631.

LI, Y.; FADDA, E.; MANERBA, D.; TADEI, R.; TERZO, O. Reinforcement learning algorithms for online single-machine scheduling. In: *2020 15th Conference on Computer Science and Information Systems (FedCSIS)*. [S.l.: s.n.], 2020. p. 277–283.

LI, Y.; NI, P.; CHANG, V. Application of deep reinforcement learning in stock trading strategies and stock forecasting. *Computing*, v. 102, n. 6, p. 1305–1322, 06 2020. ISSN 1436-5057.

LI, Y.; ZHENG, W.; ZHENG, Z. Deep robust reinforcement learning for practical algorithmic trading. *IEEE Access*, v. 7, p. 108014–108022, 2019.

LILLICRAP, T. P.; HUNT, J. J.; PRITZEL, A.; HEESS, N.; EREZ, T.; TASSA, Y.; SILVER, D.; WIERSTRA, D. Continuous control with deep reinforcement learning. In: BENGIO, Y.; LECUN, Y. (Ed.). *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. [S.l.: s.n.], 2016.

LIU, Y.; LIU, Q.; ZHAO, H.; PAN, Z.; LIU, C. Adaptive quantitative trading: An imitative deep reinforcement learning approach. *Proceedings of the AAAI Conference on Artificial Intelligence*, v. 34, n. 02, p. 2128–2135, Apr. 2020.

MA, C.; ZHANG, J.; LIU, J.; JI, L.; GAO, F. A parallel multi-module deep reinforcement learning algorithm for stock trading. *Neurocomputing*, v. 449, p. 290–302, 2021. ISSN 0925-2312.

MARINGER, D.; RAMTOHUL, T. Threshold recurrent reinforcement learning model for automated trading. In: CHIO, C. D.; BRABAZON, A.; CARO, G. A. D.; EBNER, M.; FAROOQ, M.; FINK, A.; GRAHL, J.; GREENFIELD, G.; MACHADO, P.; O'NEILL, M.; TARANTINO, E.; URQUHART, N. (Ed.). *Applications of Evolutionary Computation*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 212–221. ISBN 978-3-642-12242-2.

MILLEA, A. Hierarchical model-based deep reinforcement learning for single-asset trading. *Analytics*, v. 2, n. 3, p. 560–576, 2023. ISSN 2813-2203.

MNIH, V.; BADIA, A. P.; MIRZA, M.; GRAVES, A.; LILLICRAP, T. P.; HARLEY, T.; SILVER, D.; KAVUKCUOGLU, K. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

MNIH, V.; KAVUKCUOGLU, K.; SILVER, D.; GRAVES, A.; ANTONOGLOU, I.; WIERSTRA, D.; RIEDMILLER, M. A. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

MNIH, V.; KAVUKCUOGLU, K.; SILVER, D.; RUSU, A. A.; VENESS, J.; BELLEMARE, M. G.; GRAVES, A.; RIEDMILLER, M.; FIDJELAND, A. K.; OSTROVSKI, G.; PETERSEN, S.; BEATTIE, C.; SADIK, A.; ANTONOGLOU, I.; KING, H.; KUMARAN, D.; WIERSTRA, D.; LEGG, S.; HASSABIS, D. Human-level control through deep reinforcement learning. *Nature*, v. 518, n. 7540, p. 529–533, 02 2015. ISSN 1476-4687.

MOODY, J.; SAFFELL, M.; LIAO, Y.; WU, L. *Decision Technologies for Computational Finance: Proceedings of the fifth International Conference Computational Finance*. Boston, MA: Springer US, 1998. 129–140 p. ISBN 978-1-4615-5625-1.

MOODY, J.; WU, L. Optimization of trading systems and portfolios. In: *Proceedings of the IEEE/IAFE 1997 Computational Intelligence for Financial Engineering (CIFEr)*. [S.l.]: IEEE, 1997. p. 300–307. ISBN 0-7803-4133-3.

MOODY, J.; WU, L.; LIAO, Y.; SAFFELL, M. Performance functions and reinforcement learning for trading systems and portfolios. *Journal of Forecasting*, v. 17, n. 5-6, p. 441–470, 1998.

MULA, J.; POLER, R.; GARCÍA-SABATER, J.; LARIO, F. Models for production planning under uncertainty: A review. *International Journal of Production Economics*, v. 103, n. 1, p. 271–285, 2006. ISSN 0925-5273.

NAKANO, M.; TAKAHASHI, A.; TAKAHASHI, S. Bitcoin technical trading with artificial neural network. *Physica A: Statistical Mechanics and its Applications*, v. 510, p. 587–609, 2018. ISSN 0378-4371.

NASSIRTOUSSI, A. K.; AGHABOZORGI, S.; WAH, T. Y.; NGO, D. C. L. Text mining for market prediction: A systematic review. *Expert Systems with Applications*, v. 41, n. 16, p. 7653–7670, 11 2014.

NEUNEIER, R. Optimal asset allocation using adaptive dynamic programming. In: TOURETZKY, D.; MOZER, M.; HASSELMO, M. (Ed.). *Advances in Neural Information Processing Systems 8*. [S.l.: s.n.], 1995. (2, v. 32), p. 952–958.

NGUYEN, T. T.; NGUYEN, N. D.; NAHAVANDI, S. Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications. *IEEE Transactions on Cybernetics*, v. 50, n. 9, p. 3826–3839, 2020.

PAIVA, F. C. L.; FELIZARDO, L. K.; BIANCHI, R. A. d. C.; COSTA, A. H. R. Intelligent trading systems: A sentiment-aware reinforcement learning approach. In: *Proceedings of the Second ACM International Conference on AI in Finance*. New York, NY, USA: Association for Computing Machinery, 2022. (ICAIF '21). ISBN 9781450391481.

PARBHOO, S.; BOGOJESKA, J.; ZAZZI, M.; ROTH, V.; DOSHI-VELEZ, F. Combining kernel and model based learning for HIV therapy selection. *AMIA Jt Summits Transl Sci Proc*, United States, v. 2017, p. 239–248, 07 2017.

PARK, H.; SIM, M. K.; CHOI, D. G. An intelligent financial portfolio trading strategy using deep Q-learning. *Expert Systems with Applications*, Elsevier Ltd, v. 158, p. 113573, 11 2020. ISSN 09574174.

PASZKE, A.; GROSS, S.; MASSA, F.; LERER, A.; BRADBURY, J.; CHANAN, G.; KILLEEN, T.; LIN, Z.; GIMELSHEIN, N.; ANTIGA, L.; DESMAISON, A.; KOPF, A.; YANG, E.; DEVITO, Z.; RAISON, M.; TEJANI, A.; CHILAMKURTHY, S.; STEINER, B.; FANG, L.; BAI, J.; CHINTALA, S. Pytorch: An imperative style, high-performance deep learning library. In: *Advances in Neural Information Processing Systems 32*. [S.l.]: Curran Associates, Inc., 2019. p. 8024–8035.

PATERNINA-ARBOLEDA, C. D.; DAS, T. K. A multi-agent reinforcement learning approach to obtaining dynamic control policies for stochastic lot scheduling problem. *Simulation Modelling Practice and Theory*, v. 13, n. 5, p. 389–406, 2005. ISSN 1569-190X.

PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, v. 12, p. 2825–2830, 2011.

PENDHARKAR, P. C.; CUSATIS, P. Trading financial indices with reinforcement learning agents. *Expert Systems with Applications*, Elsevier Ltd, v. 103, p. 1–13, 08 2018. ISSN 09574174.

PINEAU, J.; VINCENT-LAMARRE, P.; SINHA, K.; LARIVIÈRE, V.; BEYGELZIMER, A.; D'ALCHÉ-BUC, F.; FOX, E.; LAROCHELLE, H. Improving reproducibility in machine learning research: a report from the neurips 2019 reproducibility program. *Journal of Machine Learning Research*, Microtome Publishing, v. 22, 2021.

PONOMAREV, E. S.; OSELEDETS, I. V.; CICHOCKI, A. S. Using reinforcement learning in the algorithmic trading problem. *Journal of Communications Technology and Electronics*, v. 64, n. 12, p. 1450–1457, 12 2019. ISSN 1555-6557.

POWELL, W. B. *Approximate Dynamic Programming: Solving the curses of dimensionality*. [S.l.]: John Wiley & Sons, 2007. v. 703.

POWELL, W. B. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. 2nd. ed. Hoboken, NJ, USA: Wiley, 2011. (Wiley Series in Probability and Statistics).

POWELL, W. B. *Handbook of Reinforcement Learning and Control*. Cham: Springer International Publishing, 2021. 29–74 p. ISBN 978-3-030-60990-0.

PUTERMAN, M. L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. 1st. ed. USA: John Wiley &; Sons, Inc., 1994. ISBN 0471619779.

RAFFIN, A.; HILL, A.; GLEAVE, A.; KANERVISTO, A.; ERNESTUS, M.; DORMANN, N. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, v. 22, n. 268, p. 1–8, 2021.

ROSSUM, G. V.; DRAKE, F. L. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN 1441412697.

RUMMERY, G. A.; NIRANJAN, M. *On-line Q-learning using connectionist systems*. [S.l.]: University of Cambridge, Department of Engineering Cambridge, UK, 1994. v. 37.

RUMMUKAINEN, H.; NURMINEN, J. K. Practical reinforcement learning -experiences in lot scheduling application ∗∗this work was in part supported by business finland, through project engineering rulez. *IFAC-PapersOnLine*, v. 52, n. 13, p. 1415–1420, 2019. ISSN 2405-8963. 9th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2019.

SAMUEL, A. L. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, v. 3, n. 3, p. 210–229, 1959.

SANG, C.; PIERRO, M. D. Improving trading technical analysis with tensorflow long short-term memory (lstm) neural network. *The Journal of Finance and Data Science*, v. 5, n. 1, p. 1–11, 2019. ISSN 2405-9188.

SARKER, I. H. Machine learning: Algorithms, real-world applications and research directions. *SN Computer Science*, v. 2, n. 3, p. 160, 03 2021. ISSN 2661-8907.

SCHAUL, T.; QUAN, J.; ANTONOGLOU, I.; SILVER, D. Prioritized experience replay. In: BENGIO, Y.; LECUN, Y. (Ed.). *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. [S.l.: s.n.], 2016.

SCHULMAN, J.; MORITZ, P.; LEVINE, S.; JORDAN, M.; ABBEEL, P. *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. 2018.

SCHULMAN, J.; WOLSKI, F.; DHARIWAL, P.; RADFORD, A.; KLIMOV, O. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

SHAVANDI, A.; KHEDMATI, M. A multi-agent deep reinforcement learning framework for algorithmic trading in financial markets. *Expert Systems with Applications*, v. 208, p. 118124, 2022. ISSN 0957-4174.

SI, W.; LI, J.; DING, P.; RAO, R. A Multi-objective Deep Reinforcement Learning Approach for Stock Index Future's Intraday Trading. In: *2017 10th International Symposium on Computational Intelligence and Design (ISCID)*. [S.l.]: IEEE, 2017. p. 431–436. ISBN 978-1-5386-3675-6.

SILVA, F. L. D.; COSTA, A. H. R. A survey on transfer learning for multiagent reinforcement learning systems. *Journal of Artificial Intelligence Research*, AI Access Foundation, v. 64, p. 645–703, mar. 2019.

SILVER, D.; HUANG, A.; MADDISON, C. J.; GUEZ, A.; SIFRE, L.; DRIESSCHE, G. van den; SCHRITTWIESER, J.; ANTONOGLOU, I.; PANNEERSHELVAM, V.; LANCTOT, M.; DIELEMAN, S.; GREWE, D.; NHAM, J.; KALCHBRENNER, N.; SUTSKEVER, I.; LILLICRAP, T.; LEACH, M.; KAVUKCUOGLU, K.; GRAEPEL, T.; HASSABIS, D. Mastering the game of go with deep neural networks and tree search. *Nature*, v. 529, n. 7587, p. 484–489, 04 2016. ISSN 1476-4687.

SILVER, D.; HUBERT, T.; SCHRITTWIESER, J.; ANTONOGLOU, I.; LAI, M.; GUEZ, A.; LANCTOT, M.; SIFRE, L.; KUMARAN, D.; GRAEPEL, T.; LILLICRAP, T.; SIMONYAN, K.; HASSABIS, D. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, v. 362, n. 6419, p. 1140–1144, 2018.

STENGEL, R. F. *Optimal control and estimation*. [S.l.]: Courier Corporation, 1994.

SUN, S.; XUE, W.; WANG, R.; HE, X.; ZHU, J.; LI, J.; AN, B. Deepscalper: A risk-aware reinforcement learning framework to capture fleeting intraday trading opportunities. In: *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. New York, NY, USA: Association for Computing Machinery, 2022. (CIKM '22), p. 1858–1867. ISBN 9781450392365.

SUTTON, R. S. Learning to predict by the methods of temporal differences. *Machine Learning*, v. 3, n. 1, p. 9–44, Aug 1988. ISSN 1573-0565.

SUTTON, R. S.; BARTO, A. G. *Reinforcement Learning: An Introduction*. 2nd. ed. Cambridge, MA, USA: A Bradford Book, 2018. ISBN 0262039249.

SUTTON, R. S.; MCALLESTER, D.; SINGH, S.; MANSOUR, Y. Policy gradient methods for reinforcement learning with function approximation. In: SOLLA, S.; LEEN, T.; MüLLER, K. (Ed.). *Advances in Neural Information Processing Systems*. [S.l.]: MIT Press, 1999. v. 12.

TAVAKOLI, A.; PARDO, F.; KORMUSHEV, P. Action branching architectures for deep reinforcement learning. *CoRR*, abs/1711.08946, 2017.

TESAURO, G. Practical issues in temporal difference learning. *Machine Learning*, v. 8, n. 3, p. 257–277, 05 1992. ISSN 1573-0565.

TESAURO, G. Td-gammon: A self-teaching backgammon program. In: _____. *Applications of Neural Networks*. Boston, MA: Springer US, 1995. p. 267–285. ISBN 978-1-4757-2379-3.

TESAURO, G.; GALPERIN, G. R. On-line policy improvement using monte-carlo search. In: *Proceedings of the 9th International Conference on Neural Information Processing Systems*. Cambridge, MA, USA: MIT Press, 1996. (NIPS'96), p. 1068–1074.

THÉATE, T.; ERNST, D. An application of deep reinforcement learning to algorithmic trading. *Expert Systems with Applications*, v. 173, p. 114632, 2021. ISSN 0957-4174.

TRIPPI, R. R.; DESIENO, D. Trading equity index futures with a neural network. *The Journal of Portfolio Management*, Institutional Investor Journals Umbrella, v. 19, n. 1, p. 27–33, 1992. ISSN 0095-4918.

TSANTEKIDIS, A.; PASSALIS, N.; TEFAS, A. Diversity-driven knowledge distillation for financial trading using deep reinforcement learning. *Neural Networks*, v. 140, p. 193–202, 2021. ISSN 0893-6080.

URBINATE, E.; FELIZARDO, L.; DEL-MORAL-HERNANDEZ, E. Deep learning stacking for financial time series forecasting: an analysis with synthetic and real-world time series. In: *Anais do XIX Encontro Nacional de Inteligência Artificial e Computacional*. Porto Alegre, RS, Brasil: SBC, 2022. p. 106–117. ISSN 2763-9061.

WANG, J.; LI, X.; ZHU, X. Intelligent dynamic control of stochastic economic lot scheduling by agent-based reinforcement learning. *International Journal of Production Research*, Taylor & Francis, v. 50, n. 16, p. 4381–4395, 2012.

WATKINS, C. J. C. H. *Learning from Delayed Rewards*. Tese (Doutorado) — King's College, Cambridge, UK, May 1989.

WERBOS, P. Advanced forecasting methods for global crisis warning and models of intelligence. *General System Yearbook*, p. 25–38, 1977.

WERBOS, P. J. Building and understanding adaptive systems: A statistical/numerical approach to factory automation and brain research. *IEEE Transactions on Systems, Man, and Cybernetics*, v. 17, n. 1, p. 7–20, 1987.

WERBOS, P. J. Using adp to understand and replicate brain intelligence: the next level design. In: *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*. [S.l.: s.n.], 2007. p. 209–216.

WU, J.; WANG, C.; XIONG, L.; SUN, H. Quantitative trading on stock market based on deep reinforcement learning. In: *2019 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2019. p. 1–8.

WU, X.; CHEN, H.; WANG, J.; TROIANO, L.; LOIA, V.; FUJITA, H. Adaptive stock trading strategies with deep reinforcement learning methods. *Information Sciences*, v. 538, p. 142–158, 2020. ISSN 0020-0255.

XIAO, J.; YANG, H.; ZHANG, C.; ZHENG, L.; GUPTA, J. N. A hybrid lagrangian-simulated annealing-based heuristic for the parallel-machine capacitated lot-sizing and scheduling problem with sequence-dependent setup times. *Computers & Operations Research*, v. 63, p. 72–82, 2015. ISSN 0305-0548.

YE, Y.; PEI, H.; WANG, B.; CHEN, P.-Y.; ZHU, Y.; XIAO, J.; LI, B. Reinforcement-Learning Based Portfolio Management with Augmented Asset Movement Prediction States. *Proceedings of the AAAI Conference on Artificial Intelligence*, v. 34, n. 01, p. 1112–1119, apr 2020. ISSN 2374-3468.

YU, C.; LIU, J.; NEMATI, S.; YIN, G. Reinforcement learning in healthcare: A survey. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 55, n. 1, 11 2021. ISSN 0360-0300.

YU, P.; LEE, J. S.; KULYATIN, I.; SHI, Z.; DASGUPTA, S. *Model-based Deep Reinforcement Learning for Dynamic Portfolio Optimization*. 2019.

ZARKIAS, K. S.; PASSALIS, N.; TSANTEKIDIS, A.; TEFAS, A. Deep Reinforcement Learning for Financial Trading Using Price Trailing. In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. [S.l.]: IEEE, 2019. v. 117, p. 3067–3071. ISBN 978-1-4799-8131-1. ISSN 09574174.

ZHAO, W.; QUERALTA, J. P.; WESTERLUND, T. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In: *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. [S.l.: s.n.], 2020. p. 737–744.

ZHOU, Z.; KEARNES, S.; LI, L.; ZARE, R. N.; RILEY, P. Optimization of molecules via deep reinforcement learning. *Scientific Reports*, v. 9, n. 1, p. 10752, 07 2019. ISSN 2045-2322.

APPENDIX A – STOCHASTIC DISCRETE LOT-SIZING PROBLEM AUXILIARY MATERIAL

A.1 Environment setting tables

Table 9 shows the configuration settings for the lower number of steps employed in the discrete lot-sizing problem experiments. The table presents the different parameter settings for four scenarios with varying numbers of time steps, number of items, number of machines, initial setup, machine production, maximum inventory level, initial inventory, holding costs, lost sales costs, demand distribution, setup costs, and setup loss. The table also indicates which parameters are randomly generated within a specified interval or matrix shape.

Table 9: Scenario configuration settings for **lower** number of steps setting. Since we generate random numbers for some of the environment settings, we only present the interval and the matrix shape.

Parameter	**I2 M1 T20**	**I4 M2 T20**	**I10 M5 T10**	**I15 M5 T10**
Time horizon	20	20	10	10
Number of items	2	4	10	15
Number of machines	1	2	5	5
Initial setup	0	$1 \times m$ array $\in [0, n]$	$1 \times m$ array $\in [0, n]$	$1 \times m$ array $\in [0, n]$
Machine production	3	$m \times n$ array $\in [0, n]$	$m \times n$ array $\in [0, n]$	$m \times n$ array $\in [0, n]$
Max inventory level	10	10	10	10
Initial inventory	0	$1 \times n$ array $\in [0, 10]$	$1 \times n$ array $\in [0, 10]$	$1 \times n$ array $\in [0, 10]$
Holding costs	0.01	0.1	0.1	0.1
Lost sales costs	1	$1 \times n$ array $\in [1, 3]$	$1 \times n$ array $\in [1, 3]$	$1 \times n$ array $\in [1, 3]$
Demand distribution	Binomial, $d_{i,t,2}$	Binomial $d_{i,t,4}$	Binomial $d_{i,t,4}$	Binomial $d_{i,t,4}$
Setup costs	1	2	2	2
Setup loss	1	1	1	1

Table 10 presents the scenario configuration settings for the experimental setting with more time steps. The table shows the parameters used in each scenario, including the

time horizon, number of items and machines, initial setup, machine production, maximum inventory level, initial inventory, holding costs, lost sales costs, demand distribution, setup loss, and setup costs. The table also indicates the shape and range of the initial setup, machine production, and inventory values. The table provides three scenarios, each with varying numbers of items, machines, and time steps, as well as different demand distributions, lost sales costs, and inventory levels. These scenarios are used to test the performance of different agents and compare them to the proposed LSCMA and ADP methods.

Table 10: Scenario configuration settings for **higher** number of steps setting. Since we generate random numbers for some of the environment settings, we only present the interval and the matrix shape.

Parameter	**I15 M5 T100 M10**	**I15 M5 T100 M100**	**I25 M10 T100 M100**
Time horizon	100	100	100
Number of items	15	15	15
Number of machines	5	5	5
Initial setup	$1 \times m$ array $\in [0, n]$	$1 \times m$ array $\in [0, n]$	$1 \times m$ array $\in [0, n]$
Machine production	$m \times n$ array $\in [0, n]$	$m \times n$ array $\in [0, n]$	$m \times n$ array $\in [0, n]$
Max inventory level	10	10	10
Initial inventory	$1 \times n$ array $\in [0, 10]$	$1 \times n$ array $\in [0, 100]$	$1 \times n$ array $\in [0, 100]$
Holding costs	0.1	0.1	0.1
Lost sales costs	$1 \times n$ array $\in [1, 3]$	$1 \times n$ array $\in [1, 3]$	$1 \times n$ array $\in [1, 3]$
Demand distribution	Binomial $d_{i,t,4}$	Binomial $d_{i,t,20}$	Binomial $d_{i,t,20}$
Setup loss	1	1	1
Setup costs	2	2	2

We also provide a table with the hyperparameters employed for the PPO and A2C algorithms:

Table 11: Hyperparameters for PPO and A2C models

Hyperparameter	**PPO**	**A2C**
Batch size	256	-
Number of steps	256	100
Gamma (γ)	0.96	0.95
GAE lambda	0.9	-
Update delay (epochs)	20	-
Entropy coefficient	0.0	-
Max gradient norm	0.5	-
Value function coef.	0.5	0.7
Learning rate α	5e-3	0.002
Use SDE	False	-
Clip range ϵ	0.4	-
Policy net architecture	[300, 300]	[300, 300]

APPENDIX B – ACTIVE SINGLE-ASSET TRADING PROBLEM AUXILIARY MATERIAL

B.1 Evaluation procedure in the ASATP

The code has been modified to represent a trading algorithm in general, as it is agnostic to any specific algorithm. The code initializes a storage array for contributions, then initializes the environment and gets a random initial state. A loop is then executed over a range of time steps, during which the next decision is obtained from the autonomous agent, the next state is obtained from the environment, and the contribution of the current decision is observed and added to the storage. After the loop, evaluation metrics are calculated from the storage of contributions. Finally, the evaluation metrics are returned.

Algorithm 12 Online execution and Evaluation of a trading agent

procedure TRADING AGENT TESTING(θ)
    Initialize storage of contributions: $\mathbf{C^h}$
    Initialize the environment and get a random initial state: $s_0 \in \mathcal{S}0$
    for t in (0, N] **do**
        ▷ Get the next decision from the RL algorithm
        $x_t \leftarrow X^{\pi}(s_t, \theta)$
        ▷ Get the next state from the environment
        $st + 1 \leftarrow S^M(s_t, x_t)$
        ▷ Observe the contribution of the current decision
        $c_t \leftarrow C(s_t, x_t)$
        ▷ Add the current contribution to the storage
        $\mathbf{C^h} \leftarrow \mathbf{C^h} \cup c_t$
    ▷ Calculate evaluation metrics
    $\langle SR, AR, ACR \rangle \leftarrow f_{val.}(\mathbf{C^h})$ ▷ SR is the Sharpe Ratio, AR is the average return, ACR is the area under the AR curve.
    ▷ Return the evaluation metrics
    return $\langle SR, AR, ACR \rangle$

B.2 Recurrent Reinforcement Learning auxiliary functions

Algorithm 13 calculates the Sharpe Ratio (SR) for a given sequence of returns $r_t, ..., r_0$. This proceedure calculates the SR regarding the momentum's A and B. The SR measures risk-adjusted performance and is commonly used in finance. The higher the SR, the better the performance of the investment.

The algorithm first computes the cumulative returns R and the cumulative squared returns $R2$ over the given sequence of returns. Then, it calculates the average return A and the average squared return B over the same sequence of returns. Finally, it computes the SR C as the ratio of A to the standard deviation of the returns, given by the square root of the difference between B and A^2.

Algorithm 13 Calculate the Sharpe Ratio

procedure $SR(r_t, ..., r_0)$
  ▷ Calculate the contribution as Sharpe Ratio
  $R \leftarrow [r_1, r_1 + r_2, r_1 + r_2 + r_3, ..., r_1 + r_2 + r_3 + ... + r_t]$
  $R2 \leftarrow [r_1^2, (r_1 + r_2)^2, (r_1 + r_2 + r_3)^2, ..., (r_1 + r_2 + r_3 + ... + r_t)^2]$
  $A \leftarrow R/T$
  $B \leftarrow R2/T$
  $C \leftarrow A/\sqrt{(B - A^2)}$
  return SR

Algorithm 14 calculates the gradient of the SR with respect to the model parameters in a Reinforcement Learning setting. The procedure takes as input the cumulative return R, the agent's actions X^π, the position size μ, and the transaction cost δ. It uses these values to calculate the gradient of SR with respect to the model parameters. The algorithm calculates the partial derivatives of SR with respect to two variables, A and B. Then it calculates the partial derivatives of A and B with respect to the discount rate r and the agent's actions X^π. Finally, it calculates the partial derivative of SR with respect to the model parameters using the chain rule. The algorithm returns the gradient of SR with respect to the model parameters. The gradient can be used to update the model parameters using a gradient-based optimization algorithm such as stochastic gradient descent. The algorithm aims to optimize the model parameters to maximize the Sharpe Ratio, which is a measure of risk-adjusted return.

Algorithm 15 outlines the procedure for validating a model in a validation environment. The algorithm takes as input the state transition function $S^{M,V}$, the current policy parameters θ_i, the validation environment distribution μ, and the discount factor δ.

Algorithm 14 Calculate the Sharpe Ratio gradient of the model parameters

procedure $f_{\ dSRd\theta}(R, X^\pi, \mu, \delta)$
    ▷ Calculate the gradient of SR with respect to the model parameters
    ▷ Commission (transaction cost): δ
    ▷ Position size: μ
    $dSRdA \leftarrow SR * (1 + SR^2)/A$
    $dSRdB \leftarrow -SR^3/2/A^2$
    $dAdr \leftarrow 1.0/T$
    $dBdr \leftarrow 2.0/T * r$
    $drdX^\pi \leftarrow -\mu\delta[r_T, ..., r_1][X^\pi(s_t) - X^\pi(s_{t-1})..., X^\pi(s_2) - X^\pi(s_1)]$
    $dX^\pi d\theta \leftarrow f_{dX^\pi d\theta}(X^\pi)$
    $dSRd\theta \leftarrow dSRdA * dAdr * drdX^\pi * dX^\pi d\theta$
    return $dSRd\theta$

The validation starts by initializing the initial state S_0 and the memory of the past state $\mathbf{S}^h$. Then, for each time step t in the validation environment, the algorithm generates $T-M$ episodes, where T is the length of the validation environment and M is the episode's length. At each time step, the current state S_t is updated by concatenating a 1 to the beginning of the state vector and appending the action chosen by the current policy $X^{\pi_\theta}(S_{t-1})$. The state transition function $S^{M,V}$ is then used to generate the next state S_{t+1}, and the immediate reward R_t is obtained. The cumulative reward C_t is computed using the discount factor δ and the past rewards.

Once all episodes are completed, the average cumulative reward C^V is computed, and the validation is completed. The algorithm returns the average cumulative reward to measure the model's performance in the validation environment.

Algorithm 15 Run in the validation environment

procedure VALIDATE($S^{M,V}, \theta_i, \delta$)
    ▷ Validate the model on the validation environment
    Initialize the initial state: S_0
    Initialize the past states memory: $\mathbf{S}^h$
    for $t \in [0, T)$ **do**
        for $j \in [1, T - M)$ **do**
            $s_t \leftarrow 1|s_t|X^\pi(s_{t-1}, \theta_i)$
            $x_t \leftarrow X^\pi(s_t, \theta_i)$
            $s_{t+1} \leftarrow S^{M,V}(s_t, x_t)$
            $c_t \leftarrow SR(r_t, ..., r_0, \delta)$
    $C^V \leftarrow \sum_{t=0}^{T} c_t/T$
    return C^V

APPENDIX C – LIST OF THE PUBLISHED ACADEMIC PAPERS DURING THE PH.D. WITH THE RESPECTIVE ABSTRACTS

This Appendix enumerates the academic works written by the candidate that were published (or are still in peer review) during the Ph.D. program. These publications are closely related to this thesis and have significantly contributed to the research and findings. Some of the work's results are directly reflected in this thesis, and others were preliminary results obtained to guide the focus of this thesis. We divide the papers into the two problems explored in this thesis: Active Single-Asset Trading Problem and Stochastic Discrete Lot-Sizing Problem.

Single asset trading problems related papers

1. L. Felizardo, F. Paiva, E. Y. Matsumoto, C. de Vita Graves, A. Realli, E. Del-Moral-Hernandez, "Outperforming algorithmic trading Reinforcement Learning systems: A supervised approach to the cryptocurrency market", **Expert Systems with Applications**, Volume 202, 15 September 2022, 117259,
   DOI: https://doi.org/10.1016/j.eswa.2022.117259

2. E. Urbinate, L. Felizardo, E. Del-Moral-Hernandez "Deep learning stacking for financial time series forecasting: an analysis with synthetic and real-world time series", **2022: Anais do XIX Encontro Nacional de Inteligência Artificial e Computacional**, 28 November 2022, DOI: https://doi.org/10.5753/eniac.2022

3. L. Felizardo, E.Y. Matsumoto, E. Del-Moral-Hernandez, "Solving the optimal stopping problem with Reinforcement Learning: an application in financial option exercise", **2022 International Joint Conference on Neural Networks (IJCNN)**, 18-23 July 2022, DOI: 10.1109/IJCNN55064.2022.9892333

4. F. Paiva, L. Felizardo, A. Realli, R. Bianchi, "Intelligent Trading Systems: A Sentiment-Aware Reinforcement Learning Approach", **2021 6th International Conference on AI in Finance (ICAIF)**, 2021, pp. 1-8, DOI: https://doi.org/10.1145/3490354.3494445

5. L. Felizardo, R. Oliveira, E. Del-Moral-Hernandez, and F. Cozman, "Comparative study of Bitcoin price prediction using WaveNets, Recurrent Neural Networks and other Machine Learning Methods", **2019 6th International Conference on Behavioral, Economic and Socio-Cultural Computing (BESC)**, 2019, pp. 1-6, doi: 10.1109/BESC48373.2019.8963009

Stochastic Discrete Lot-Sizing Problems related papers

1. L. Felizardo, E. Fadda, E. Del-Moral-Hernandez, P. Brandimarte, "Reinforcement Learning approaches for the Stochastic Discrete Lot-Sizing Problem on parallel machines", **Submitted and waiting for the peer review feedback of the International Journal Expert Systems With Applications**

2. D. Gioia, L. Felizardo, P. Brandimarte, "Simulation-Based Inventory Management of Perishable Products Via Linear Discrete Choice Models", **Computers & Operations Research**: www.sciencedirect.com/science/article/pii/S030505482300134X, DOI: doi.org/10.1016/j.cor.2023.106270

3. D. Gioia, L. Felizardo, P. Brandimarte, "Inventory management of vertically differentiated perishable products with stock-out based substitution", **Manufacturing Modelling, Management and Control - 10th MIM 2022**, DOI: https://doi.org/10.1016/j.ifacol.2022.10.115

Here we present the abstracts extracted from their original source papers published during the Ph.D. program:

Single asset trading problems related papers

1. Title of the paper: Outperforming algorithmic trading Reinforcement Learning systems: A supervised approach to the cryptocurrency market

The interdisciplinary relationship between machine learning and financial markets has long been a theme of great interest among both research communities. Recently, Reinforcement Learning and deep learning methods gained prominence in the active asset trading task, aiming to achieve outstanding performances compared with classical benchmarks, such as the Buy and Hold strategy. This paper explores both the Supervised Learning and Reinforcement Learning approaches applied to active asset trading, drawing attention to the benefits of both approaches. This work extends the comparison between the supervised approach and Reinforcement Learning by using state-of-the-art strategies with both techniques. We propose adopting the ResNet architecture, one of the best deep learning approaches for time series classification, into the ResNet-LSTM actor (RSLSTM-A). We compare RSLSTM-A against classical and recent Reinforcement Learning techniques, such as Recurrent Reinforcement Learning, Deep Q-Network, and advantage actor–critic. We simulated a currency exchange market environment with the price time series of the Bitcoin, Litecoin, Ethereum, Monero, Next, and Dash cryptocurrencies to run our tests. We show that our approach achieves better overall performance, confirming that Supervised Learning can outperform Reinforcement Learning for trading. We also present a graphic representation of the features extracted from the ResNet neural network to identify which type of characteristics each residual block generates.

2. Title of the paper: Deep learning stacking for financial time series forecasting: an analysis with synthetic and real-world time series

   The forecasting problem is one of the main applications arising from the synergy between finance and artificial intelligence. With the advancement in the field of deep learning, some ANN achieved very satisfactory results and gained more attention. One approach to increase the time series forecasting model's performance is ensemble models, combining each model's prediction (stacking). However, there are some difficulties in combining and evaluating these models for a good performance in financial time series. We use synthetic and real-world time series to evaluate the model stacking, trying to understand the main financial time series components. Using this ensemble method, we reduced the prediction error for both scenarios.

3. Title of the paper: Solving the optimal stopping problem with Reinforcement Learning: an application in financial option exercise

   The optimal stopping problem is a category of decision problems with a specific constrained configuration. It is relevant to various real-world applications such as

finance and management. To solve the optimal stopping problem, state-of-the-art algorithms in Dynamic Programming, such as the Least-Squares Monte Carlo (LSMC), are employed. This type of algorithm relies on path simulations using only the last price of the underlying asset as a state representation. In addition, the LSMC is designed for the valuation of options where risk-neutral probabilities can be employed to explain uncertainty. However, the general optimal stopping problem goals may not fit the requirements of the LSMC showing auto-correlated prices. We employ a data-driven method that uses Monte Carlo simulation to train and test Artificial Neural Networks (ANN) to solve the optimal stopping problem. Using ANN to solve decision problems is not entirely new. We propose a different architecture that uses Convolutional Neural Networks (CNN) to deal with the dimensionality problem that arises when we transform the whole history of prices into a Markovian state. We present experiments that indicate that our proposed architecture improves results over the previous implementations under specific simulated time series function sets. Lastly, we employ our proposed method to compare the optimal exercise of the financial options problem with the LSMC algorithm. Our experiments show that our method can capture more accurate exercise opportunities when compared to the LSMC. We have an outstandingly higher (above 974% improvement) expected payoff from these exercise policies under the many Monte Carlo simulations that used the real-world return database on the out-of-sample (test) data.

4. Title of the paper: Intelligent trading systems: a sentiment-aware Reinforcement Learning approach

   The feasibility of making profitable trades on a single asset on stock exchanges based on patterns identification has long attracted researchers. Reinforcement Learning (RL) and Natural Language Processing have gained notoriety in these single-asset trading tasks, but only a few works have explored their combination. Moreover, some issues are still not addressed, such as extracting market sentiment momentum through the explicit capture of sentiment features that reflect the market condition over time and assessing the consistency and stability of RL results in different situations. Filling this gap, we propose the Sentiment-Aware RL (SentARL) intelligent trading system that improves profit stability by leveraging market mood through an adaptive amount of past sentiment features drawn from textual news. We evaluated SentARL across twenty assets, two transaction costs, and five different periods and initializations to show its consistent effectiveness against baselines. Subsequently,

this thorough assessment allowed us to identify the boundary between news coverage and market sentiment regarding the correlation of price-time series above which SentARL's effectiveness is outstanding.

5. Title of the paper: Comparative study of Bitcoin price prediction using WaveNets, Recurrent Neural Networks and other Machine Learning Methods

   Forecasting time series data is an important subject in economics, business, and finance. Traditionally, there are several techniques such as univariate Autoregressive, univariate Moving Average, Simple Exponential Smoothing, and more notably Autoregressive Integrated Moving Average with their many variations that can effectively forecast. However, with the recent advancement in the computational capacity of computers and more importantly developing more advanced machine learning algorithms and approaches such as deep learning, new algorithms have been developed to forecast time series data. This article compares different methodologies such as ARIMA, Random Forest, Support Vector Machine, Long Short-Term Memory and WaveNets for estimating the future price of Bitcoin.

Stochastic Discrete Lot-Sizing Problems related papers

1. Title of the paper: Reinforcement Learning approaches for the Stochastic Discrete Lot-Sizing Problem on parallel machines

   This paper addresses the Stochastic Discrete Lot-Sizing Problem on parallel machines which is a computationally challenging problem also for relatively small instances. We propose two heuristics to deal with it leveraging Reinforcement Learning. In particular, we propose a technique based on approximate Value Iteration around post-decision state variables and one based on multi-agent Reinforcement Learning. We compare these two approaches with other Reinforcement Learning methods and more classical solution techniques, showing their effectiveness in addressing realistic size instances.

2. Title of the paper: Simulation-based inventory management of perishable products via linear discrete choice models

   Retail inventory management of perishable items, like fresh food, is a relevant and complex problem. It is relevant in the light of trends towards the reduction of food waste, and because of potential cross-sales interaction with other item categories. It is complex, because of multiple sources of uncertainty in supply, demand, and quality, and other complicating factors like seasonality within the week, First in, First

out and Last in, Last out consumer behaviors, and potential substitutions between items, possibly because of a stockout. Similar items may be vertically differentiated due to intrinsic quality, which is also related with item age, or brand image, as it could be the case when a retail chain stocks both a brand item and a private label one. In the paper, we adapt a simple discrete choice model to represent consumers' heterogeneity and different tradeoffs between price and quality, and apply simulation-based optimization to learn simple ordering rules for two vertically differentiated items, adapted to a seasonal case, in order to maximize long-term average profit under a lost sales assumption. While well-known constant and base-stock policies need not be optimal, they are simple to communicate and apply. We explore combinations of such rules for the two items, obtaining some useful managerial insights.

3. Title of the paper: Inventory management of vertically differentiated perishable products with stock-out based substitution

   The need for optimal inventory control strategies for perishable items is of the utmost importance to reduce the large share of food products that expire before consumption and to achieve responsible food stocking policies. Our study allows for a multi-item setting with substitution between similar goods, deterministic deterioration, delivery lead times and seasonality. Namely, we model demand by a linear discrete choice model to represent a vertical differentiation between products. The verticality assumption is further applied in a novel way within product categories. Specifically, the same product typology is vertically decomposed according to the age of the single stock-keeping unit in a quality-based manner. We compare two different policies to select the daily size of the orders for each product. On the one hand, we apply one of the most classical approaches in inventory management, relying on the Order-Up-To policy, modified to deal with the seasonality. On the other hand, we operate a state-of-the-art actor-critic technique: Soft Actor-Critic. Although similar in terms of performance, the two policies show diverse replenishment patterns, handling products differently.