Master's Thesis
for the Attainment of the Degree
Master of Science
at the TUM School of Management
of the Technische Universität München

Deep Reinforcement Learning for Multi-Age Inventory Management with Complex Production Actions

Examiner:	Prof. Dr. Martin Grunow Chair of Production and Supply Chain Management
Person in Support:	M.Sc. Alexander Pahr
Course of Study	Master in Management and Technology
Submitted by:	Thu Ha Dao St.-Martin-Straße 30 85253 Erdweg Matriculation Number 03721313
Submitted on:	01.02.2022

Abstract

Dealing with complex action spaces is a major challenge in multi-age inventory management. One example of this challenge is the inventory control problem of port wine products, in which blending wine from different age classes is allowed to flexibly serve demands and yet causes the decision-making more complicated. We tackle the researched problem with a deep reinforcement learning algorithm, namely deep Q-network (DQN). Additionally, we propose a method to integrate domain knowledge to improve the performance of DQN by adapting a published curriculum learning technique to combinatorial action spaces with sequential sub-actions. By appropriately restricting the action space in the beginning of the training process, we may guide the agent towards more meaningful experiences. The benchmarking results show that our model provides higher and more stable results than the original DQN, as well as achieves a median expected profit equal to 94% of the optimal value obtained by value iteration, yet with total running time being only 25.8% of value iteration's running time.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ANN	Artificial neural network
CAT	Conditional Action Tree
DQN	Deep Q – network
DRL	Deep reinforcement learning
FIFO	First in first out
GAS	Growing action space
LIFO	Last in first out
MDP	Markov Decision Process
RL	Reinforcement learning

1 Introduction

In the scope of this research, a multi-age product is defined as any product that is controlled based on its age, i.e., the time this product is stored in the inventory. There are two main types of multi-age products mentioned in most research: (i) goods that depreciate over time (e.g., perishable food items, blood cells and platelets, pharmaceuticals or chemicals) (Karaesmen et al. 2011) and (ii) goods that appreciate over time (e.g., forests or wine) (Lin and Buongiorno 1998, Pahr et al. 2021). Because of their importance (Chen et al. 2021, Civelek et al. 2015) and prevalence (Karaesmen et al. 2011), it is critical to manage inventories of multi-age products in an effective manner to meet demands while preventing excessive waste and reducing handling cost (Civelek et al. 2015, Karaesmen et al. 2011).

Research into inventory management for multi-age products focuses on ordering policy and issuing policy (in this thesis, issuing policy and production policy are used interchangeable). One of the common approaches to solve the inventory management problem for this type of products is using Markov Decision Process (MDP) to model the system (Cohen 1976, Haijema et al. 2017, Lin and Buongiorno 1998). However, finding the optimal policy is not easy in many cases due to the curse of dimensionality in the state space and action space of the MDP (Haijema 2008, Moor et al. 2021, Pahr et al. 2021). In detail, the size of the state space will exponentially increase with the number of age classes, while the size of the action space becomes larger in case of more complex structured actions, for example, when demands are age-differentiated and/or substitution is allowed in an issuing decision. Through the problem of port wine inventory management, we illustrate an example with the complexity in both spaces, especially in the action one due to blending decision and its combination with sales decision at the same time.

One unique step in the production process of port wines is blending. Usually, a bottle of branded port wines sold in the market was labelled with an age indication, which represents the minimum average storage time of the wine inside (hereafter called the target age). To make port wine with a specific target age, producers may mix wines from different age classes together, and this action is called blending. Figure 1 below illustrates the integration of sales and blending decisions in the final production action in a case of two products in one period. In this figure, the demand of the young port wine is intentionally underfulfilled while the demand of the old port wine is fully met by a blend of inventory units from various age classes. In

other words, in each decision epoch, producers need to decide on two things sequentially:

i. *How many* demand units will be satisfied for each product?

ii. *How* to blend those demand units?

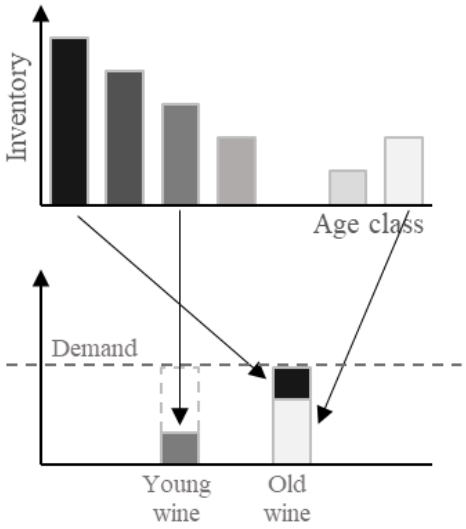

Figure 1: Final production action integrating sales and blending decisions in the researched problem. Source: adapted from Pahr et al. (2021)

Blending provides a great flexibility to serve demands but on the other hand, it also makes the decision-making more puzzling. In our numerical experiment, we show that the size of the action space with blending is 216 times larger than the size of the action space without blending. In this context, it is difficult to achieve a good policy just by using some common issuance rules such as first in first out (FIFO) or last in first out (LIFO), which motivates us to directly address the multi-age inventory management problem and thus, the intrinsic challenge of curse of dimensionality.

The curse of dimensionality specifically due to a large state space is a common challenge in all inventory control problems. In order to circumvent this challenge, application of deep reinforcement learning (DRL) into inventory management has been widely researched recently (Boute et al. 2021, Gijsbrechts et al. 2021). One of the well-known algorithms in DRL is Deep Q-network (DQN) (Mnih et al. 2015), along with its variants (Hessel et al. 2018, Schaul et al. 2015, van Hasselt et al. 2016). Since its introduction in Mnih et al. (2015) with impressive results on several arcade video games, DQN has been greatly improved and extended to apply in various fields such as robotics (Zhang and Mo 2021), transportation (Parvez Farazi et al. 2021) and production system control (Panzer and Bender 2021). In inventory management, there are also some fruitful attempts with DQN presented (Meisheri

et al. 2021, Moor et al. 2021, Oroojlooyjadid et al. 2021). Considering its straightforward implementation and high flexibility to configure and integrate with other modules (van de Wiele et al. 2020), we also choose DQN as the DRL framework to focus on in this thesis.

However, DQN, as well as other conventional DRL algorithms, still has limitations on dealing with complexity due to the action space, especially combinatorial one (Yang et al. 2021, Zahavy et al. 2018). According to Zheng and Yue (2018), even in the case of a simple underlying dynamic, there is still a challenge of how to explore in high-dimensional action spaces. The big scale of these spaces leads to a more severe difficulty in collecting informative experiences for the DRL agent, worsening the intrinsic problem of sample-inefficiency in DRL (Yang et al. 2021). Within the boundary of this thesis, we make an effort to overcome this challenge specifically in the case of multi-age inventory management by leveraging domain knowledge to assist DRL agent in exploring the action space. The motivation behind this approach is that while not knowing the exact optimal policy beforehand, we may know some of its characteristics simply based on our domain knowledge, which can come from relevant experience, past research or even an intuition. For example, in the context of multi-age inventory, one of the main wastes is due to outdating (Karaesmen et al. 2011). Therefore, it is reasonable, and does not require a hard effort at the same time, to guess that we need to minimize the outdated amount in order to achieve the optimal policy. Another example is insights obtained from the characteristics of the risk distribution or cost structure of the problem. By utilizing these pieces of information, we may extend the capability of DRL to deal with more difficult problems, such as in the example presented in this thesis.

We base our methodology on the work of Farquhar et al. (2020). They point out that if an action space is complex due to complicated tasks, a strategy of random exploration may not be able to guide the agent to learn in a meaningful way, leading to a slow converge. Therefore, a curriculum with *growing action space*, abbreviated as GAS in the paper, is proposed to tackle the problem of inefficient exploration in a specific case of hierarchical action spaces. Curriculum learning is a methodology aiming to organize the learning process of the agent in both experience level and/or task level. At the task level, it means that the order of tasks to be introduced to the agent is arranged, mostly from simple tasks to more difficult ones (Narvekar et al. 2020). Similar to this idea, Farquhar et al. (2020) define GAS as a curriculum internal to the agent by initially restricting and then gradually expanding the action space during training. It leads the agent towards more informative experiences at

the start of the learning process. After that, by transferring experiences, value estimates and state representations gained in more restricted spaces into less restricted ones, the agent may increase its exploration efficiency. This strategy is feasible whenever a hierarchy can be defined in the action space, such as those in two cases directly addressed in the original paper, namely discretized continuous control and multi-agent RL with parallel sub-actions.

In our thesis, we extend application of GAS to the case of action spaces with sequential sub-tasks, which is applicable to our problem and potentially other multi-age inventory management problems. To do so, we formulate a method to exploit domain knowledge to define a hierarchy in the action space and implement the derived learning curriculum on top of DQN algorithm. In our proposed formula, domain knowledge is embedded in the learning as mapping functions used to divide the full action space into sub-spaces that are easier to be learned. The consequent shaped exploration is expected to be more efficient than a merely random exploration in coping with a large-scale action space. Indeed, in the numerical experiment analysis, we show that our modified DQN provides better and more stable results than the original DQN. For instance, the median gap between the optimal expected profit and the expected profits obtained from our model is 6%, which equals approximately 1/5 of the median gap of the original DQN. Additionally, we also examine the robustness of our model by comparing the performances obtained when applying two different mapping functions based on different levels of understanding about the problem.

In summary, in this thesis, we provide academic contributions by answering the following research questions:

- **[RQ1]**: *How can we use domain knowledge/ understanding about problem to improve the performance of DRL in the context of a multi-age inventory management problem with complex production action?*
    - ⇨ To answer this question, we focus our research on a multi-age inventory management problem of port wine products which can be characterized by its complex production actions consisting of sales and blending sub-actions. In the context of this problem, we propose a modified version of an existing learning curriculum (i.e., GAS) which integrates domain knowledge as mapping functions in order to structure an action space into hierarchical smaller action sub-spaces which are learned step by step.

- **[RQ2]:** *How to apply GAS-based curriculum learning into a combinatorial action space with sequential sub-actions?*
  - ⇨ In our model mentioned in **[RQ1]**, we also present a mathematical formulation to define a hierarchy that enables the application of GAS in a combinatorial action space with sequential sub-actions. An example in the context of our researched problem is also provided to illustrate the concepts.
- **[RQ3]:** *How is the robustness of the proposed methodology, especially in terms of the choice of mapping functions?*
  - ⇨ Regarding this question, we design an experiment in which we evaluate the results obtained by using our model with two mapping functions that are based on different levels of problem understanding. Besides, we also provide other experiments to assess our model's robustness to some variations in scale and parameter of the problem.

The remainder of this thesis is structured as follows. The next chapter presents literature related to three topics: (i) multi-age inventory management, (ii) application of DRL in inventory management and (iii) DRL dealing with (large) combinatorial action space. Chapter 3 provides a general theoretical background on DQN and curriculum learning paradigm mentioned in Farquhar et al. (2020) to support the understanding of our proposed methodology. Chapter 4 introduces the problem of port wine inventory management and its derived MDP model, which are used throughout the thesis. Chapter 5 explains in detail the mathematical formulation and model design of our methodology. Chapter 6 demonstrates the implemented numerical experiments and analyzes the results derived from that. Finally, in chapter 7, we summarize our work and point out some areas for further research.

2 Review of Literature and Research

In this chapter, we review related literature covering three main topics of this thesis. The first section presents studies on the topic of multi-age inventory management, highlighting how the complexity is dealt with in each selected paper. The second section provides an overview on application of DRL in different inventory management settings. The last section concludes the chapter with a discussion about existing approaches to deal with a (large) combinatorial discrete action space in DRL literature.

2.1 Multi-age inventory management

In this thesis, multi-age products are defined as products of which values change, decreasingly or increasingly, over their storage time (hereafter denoted as m), which lasts more than one period to differentiate with the newsvendor problem. Among multi-age products, the one deteriorates while aging, or known as perishables, appears earlier and more extensive in academic publications due to its prevalence in consumer goods as well as industrial products (Karaesmen et al. 2011). Research into inventory management of perishables was initiated by van Zyl in 1964 (Nahmias and Pierskalla 1976) and has regained a great deal of attention especially from 2010s until now (Chaudhary et al. 2018). One possible reason to explain this trend is that in the past, deficiency in computing power and theoretical basis hampered the implementation of numerical approaches while multi-age inventory management problems are too complicated to solve with analytical methods when the number of age classes considered is high. For more general reviews on inventory management of perishable products, we refer interested readers to Pierskalla (2005), Karaesmen et al. (2011) and Chaudhary et al. (2018). Since the focus of the thesis is on complex action spaces, we direct our attention to multi-age inventory systems which have age-differentiated demands and/or complex actions (e.g., allowing substitution, combining different sub-actions, etc.). Table 1 summarizes all papers discussed in this section.

Table 1: Summary of literature of multi-age inventory management

Paper	Optimized decision	Method	Rule-based solution	Action complexity		
				AD	S	CS
Nahmias and Pierskalla (1976)	Ordering	A	No		✓	
Cohen et al. (1981)	Ordering	A	Yes			✓
Goh et al. (1993)	Issuing	A	Yes	✓	✓	
Deniz et al. (2010)	Ordering + Issuing (pair)	A	Yes	✓	✓	
Deniz et al. (2020)	Ordering + Issuing (pair)	A	Yes	✓	✓	
Haijema et al. (2007)	Ordering	N	No	✓	✓	
Civelek et al. (2015)	Ordering + Issuing	N	Yes	✓	✓	✓
Chen et al. (2021)	Ordering + Issuing	A, N	No	✓	✓	✓
Abbaspour et al. (2021)	Ordering	N	Yes	✓	✓	
Pahr et al. (2021)	Ordering + Issuing	N	No	✓	✓	✓
Hendrix et al. (2019)	Ordering	N	Yes		✓	
This approach	**Issuing**	**N**	**No**	✓	✓	✓

Abbreviations:

R – Rule-based setup, A – Analytical analysis, N – Numerical analysis

AD – Age-differentiated demands, S – Substitution, CS – Combination of sub-actions

We observe that most papers before 2000s on multi-age inventory considered perishable products with small values of m, mainly two to three periods, to simplify calculations and adopted analytical approaches to find optimal solutions. Nahmias and Pierskalla (1976) consider an inventory system which involves two products, one perishable (with demand) and one nonperishable (without demand), and allows nonperishable product to substitute perishable one in case of out of stock. The authors assume FIFO for issuing policy and present optimal solution for the case of $m = 2$. By setting a very high penalty cost for holding over-age stock to mimic perishability, Cohen et al. (1981) are able to reduce the complexity and compute the optimal order-up-to ordering level while determining allocation policy

simultaneously in a multi-echelon case with $m = 2\ or\ 3$. Goh et al. (1993) use a two-stage model to simulate an inventory system of a perishable product in two states: fresh and older, which is equivalent to considering $m = 2$, and analytically compare two settings of FIFO issuance rules under stochastic supply and stochastic age-differentiated demands. Continuing the stream of analytical research in multi-age inventory management in recent years, Deniz et al. (2010, 2020) focus on the effects of different replenishment policies on different issuance rules and vice versa in the case of $m = 2$. We deviate from the papers mentioned above by tackling a problem which is much larger in sizes and thus, hardly solvable with analytical methods.

In the last twenty years, there are more studies adopting numerical approaches to solve larger-scale problems, many of them use MDP to model the system (Abbaspour et al. 2021, Haijema et al. 2017). Haijema et al. (2007) present the production and inventory management in a blood bank as a MDP and propose a double-level order-up-to replenishment rule while assuming FIFO and LIFO-based rules on issuing side. The authors also discuss about the computational complexity of the problem and consider a down-sized problem to obtain insights into the optimal policy. Researching a similar managerial problem of blood platelets, Civelek et al. (2015) extend their work to tackle multiple types of decisions simultaneously: replenishment, allocation and issuance. They contribute by introducing the critical level policy, a heuristic to prevent excessive substitution of new items for old ones. Chen et al. (2021) define a heuristic called adaptive approximation after analytically examining various properties of the optimal policies and objective functions, as well as the multi-modularity in the dynamic programming model in a problem involving age-differentiated demands with different lost-sales penalty cost for each demand stream. Abbaspour et al. (2021) focus more on the practicality of solutions when using data-driven simulation to derive policies that are comprehensible and easy-to-use for anyone. Also emphasizing the solutions' interpretability and computing feasibility, Pahr et al. (2021) transform a large-scale multi-age inventory problem into a much smaller one and utilize the insights gained in the down-sized problem to build an optimal classification tree to apply into the original problem. While all previous research just briefly discusses about the difficulty brought by the complexity of the problem, Hendrix et al. (2019) concentrate their work on assessing the huge efforts needed to compute the optimal order-up-to level with the value iteration method. They

point out that the computational burden will increase significantly when product substitution is allowed.

Compared to the above papers, this thesis differentiates itself in the following points. Firstly, these papers mainly focus on ordering (or replenishment) policy, we are more interested in the issuing policy (or also called production policy) because of its higher complexity. Secondly, most mentioned studies do not directly deal with the whole action space but use some rules to reduce the number of possibilities to consider, such as order-up-to levels in the case of ordering actions. On the other hand, in this thesis, we consider all possible actions without common heuristic assumptions about issuing action, such as FIFO or LIFO, which may not be able to capture complex actions like in our problem. Finally, similar to Hendrix et al. (2019), we restate the huge computational cost required to achieve optimal policy with traditional approaches such as value iteration. However, in the context of the presented problem, we additionally propose a method combining DRL and some common understanding of the problem which achieves a result close to the one obtained from value iteration and yet with much less time.

2.2 DRL in inventory management

Reinforcement learning (RL) is the field of learning what to do, by trial and error, considering the current situation in order to maximize the expected reward which was affected by not only the immediate reward but also the future ones (Sutton and Barto 2018). An extension of RL using multi-layer artificial neural networks (ANNs) as approximators for the value or policy functions is called deep reinforcement learning (DRL). One advantage of DRL methods compared to RL methods is the ability to generalize from their experiences. In other words, DRL agents can select actions even in the states not yet observed, hence this approach may be able to address problems with large state spaces (Sutton and Barto 2018). Therefore, DRL has emerged in recent years as a potential approach to find near-optimal policies in many intractable inventory control problems (Boute et al. 2021). An overview of papers presented in this section can be found in Table 2 below. Besides, we refer to Boute et al. (2021) as a comprehensive review on application of DRL in the field of inventory management, where the authors not only give an overview of current research but also discuss about how to design an effective DRL paradigm for inventory control and point out potential directions for future research.

Table 2: Summary of literature of DRL in inventory management

Paper	DRL algorithm	Decision	Inventory setting	Transfer learning between / from
Gijsbrechts et al. (2021)	A3C	Replenishment + distribution	ME	
Oroojlooyjadid et al. (2021)	DQN	Independent ordering with cooperative goal	ME	Agents
Harsha et al. (2021)	PARL	Replenishment + distribution	ME	
Sultana et al. (2020)	A2C	Replenishment + distribution	MP, ME	Products
Meisheri et al. (2021)	DQN, PPO	Replenishment + distribution	MP, ME	Products
Moor et al. (2021)	DQN	Ordering	MA	Heuristics
This thesis	**DQN**	**Production (Sales + Blending)**	**MA**	**Domain knowledge**

Abbreviations:
ME – Multi-echelon, MP – Multi-product, MA – Multi-age

To the best of our knowledge, among all papers looking at the intersection of DRL and inventory management, the most researched setting is multi-echelon inventory system, which involves multiple stages and stakeholders. Gijsbrechts et al. (2021) apply Advantage Actor Critic DRL algorithm into a multi-echelon inventory model including one warehouse and many identical retailers, along with two other challenging inventory settings, i.e., lost sales and dual-sourcing. By presenting a versatile solution to various problems, the authors prove that DRL is a multi-purpose approach which does not require profound domain knowledge as traditional heuristic methods. Oroojlooyjadid et al. (2021) also demonstrate the robustness of DRL in a complicated context of a four-node beer game inventory problem. Using reward shaping to adapt DQN algorithm to a cooperative non-zero-sum game setting, they show that their trained agents can perform well regardless of whether the behaviors of other co-players follow a base-stock policy or not. Harsha et al. (2021) propose a new policy-based DRL framework called Programmable Actor Reinforcement Learning (PARL), in which ANNs still work as a value function approximator in policy evaluation, but integer programming and

sample average approximation are used to generate new greedy policies in policy improvement. By integrating optimization techniques into DRL and selecting an appropriate strategy to discretize the distributions of uncertain variables, Harsha and the co-authors lift the computational burden caused by enumerated operations when implementing existing DRL methods in a large action space, achieving near-optimal policies in various multi-echelon inventory settings. Despite sharing the same focus on the curse of dimensionality from action space like Harsha et al. (2021), in this thesis, we approach it from a different perspective, i.e. the inefficient exploration in the learning process, and hence, the framework we adopt is different too. Consequently, the curriculum learning technique proposed in this thesis does not aim to be an alternative to the DRL algorithm defined in Harsha et al. (2021). Indeed, it is potential to combine two methods and take advantage of both to better deal with the large-scale action space.

Another inventory setting that is less popular than multi-echelon but still interests us due to its resemblance to the multi-age setting is the inventory control of multiple products. Sultana et al. (2020) implement multi-agent hierarchical DRL frameworks to address parallelized decision-making problems of up to 1000 products in a two-echelon supply chain of one warehouse and three stores. Meisheri et al. (2021) extend the previous work by introducing a meta-model for individual product, which can be reused in transfer learning process to adapt to changes in the number of products without retraining. Finally, regarding the specific multi-age inventory system, there has not been much research of applying DRL in this setting. One paper strongly related to our thesis is from Moor et al. (2021). They realize the untapped potential of heuristics, which have been extensively studied for a long time, to improve the performance of DRL algorithms in the field of inventory control. In detail, with reward shaping technique, they create teacher policies from existing heuristic ordering policies, i.e. base-stock and the BSP-low-EW policy from Haijema and Minner (2019), to transfer the knowledge embedded in these heuristics to the DQN agent, leading to a faster and more stable learning process.

To sum up, most of the mentioned papers depict DRL as a powerful standalone approach which can solve various inventory control problems without dependence on the understanding of the context. In comparison, we want to highlight benefits that domain knowledge can bring to DRL to solve more challenging problems, such as complex action spaces. While Moor et al. (2021) follow a similar direction as us, their paper differentiates from ours on the challenge in focus and the method in use. To start with, they aim to overcome the sampling inefficiency in state space,

whereas we concentrate on the exploration inefficiency in action space. Additionally, their method requires a complete heuristic to be able to build a teacher policy. Regarding our approach, we only need a limited amount of domain knowledge specifically about the structure of the action space, which can be less exhaustive than a thorough solution in some cases.

2.3 DRL in (large) combinatorial action spaces

Table 3: Summary of literature of DRL in combinatorial action spaces

Paper	Methodology	Remarks
ANN configuration		
Zhao et al. (2018)	Adjusting components of DRL	Changing input and output of ANNs in DQN algorithm
Zahavy et al. (2018)	Adding supplemental modules	A separate ANN to eliminate irrelevant actions
van de Wiele et al. (2020)		A separate ANN to predict the best-known action
You et al. (2020)		ANN module to learn values of state decompositions
Delarue et al. (2020)	Partially replacing ANNs	DRL integrated with mix-integer programming
Harsha et al. (2021)		DRL integrated with mix-integer programming
Cappart et al. (2021)		DRL integrated with constraint programming
Action space transformation		
Dulac-Arnold et al. (2016)	Embedding actions	Discrete action embedded into continuous one
Kanervisto et al. (2020)	Removing actions Combining actions	Review of DRL algorithms applied in video game
Farquhar et al. (2020)	**Structuring action spaces**	Action space divided into restricted sub-spaces
Bamford and Ovalle (2021)		Action space decomposed into a sequence of sub-spaces
This thesis		**GAS extended to action space with sequential sub-tasks**

In general, large and complex action spaces has been regarded as one of the main challenges on the way to extending the application of DRL into more real-world problems (Dulac-Arnold et al. 2021). In this section, we focus on the literature on discrete combinatorial action spaces (Delarue et al. 2020, Dulac-Arnold et al. 2016), which is the case of the problem researched in this thesis as well as most of inventory management problems. As far as we research, adopted approaches can be divided into two categories: (i) ANN configuration (e.g., adjusting the input/output, adding supplemental modules, partially replacing ANNs), or (ii) action space transformation (e.g., embedding actions, removing actions, structuring the action space). Certainly, there are some methodologies relevant to both categories, but for the sake of simplicity, we will categorize them to the most related one. Table 3 above shows papers reviewed in this section with the corresponding categories and groups of methodologies.

Following the direction in which attention is turned to the structure of ANNs, Zhao et al. (2018) use future state and value estimation for state-action pair as input and output of the network in DQN respectively, instead of current state as input and estimation for all actions as output as in the original algorithm. However, this method is applicable only in the case of deterministic future states, i.e., a next state solely dependent on its previous state and action. On the other hand, Zahavy et al. (2018) design a supplemental network, specific to NLP tasks, to eliminate irrelevant actions based on an auxiliary reward and domain knowledge, creating a better context for DQN agent to learn. Also focusing on DQN algorithm, van de Wiele et al. (2020) propose to add another ANN to predict the best known action as a separate problem of explicit searching and supervised learning. The result from this network is used to suggest a set of potential actions, instead of the whole space, to DQN agent. While sharing the same idea of building a proposal of actions in each learning step, You et al. (2020) stress on the understanding of state representations and develop a new two-stage hierarchical Q-learning framework which features a new module to compute the Q-values for all state decompositions and select the most promising one. In addition, there are many research deviating from the course of pure deep learning and combining the strength of other paradigms such as mathematical programming or constraint programming to improve the performance. For example, Delarue et al. (2020) implement mixed-integer programming (MIP) with an adapted branch-and-cut approach to select action in the policy evaluation step (ANNs are still used as value function approximator in policy improvement step) in the context of a capacitated vehicle routing problem.

Harsha et al. (2021), as introduced in the previous section, extend this approach to the case that the immediate reward is uncertain and probabilistic. Cappart et al. (2021) successfully integrate two popular DRL frameworks into various constraint programming search strategies.

Another common research direction is performing transformation on the action space. One early method to do so is converting the action representation from the discrete space into a continuous space and choosing the nearest discrete actions to the optimal continuous actions computed, such as in the paper of Dulac-Arnold et al. (2016). In their review based on the submissions of top contestants in many video game competitions, Kanervisto et al. (2020) point out the three most popular modifications applied to an action space, namely removing actions, discretizing continuous actions and converting multi-dimensional actions to one-dimensional ones. According to the authors, these action state shaping techniques provide similar benefits as the reward shaping technique does. The final group of methodologies we want to discuss in this section is action space structuring. The curriculum (GAS) defined in Farquhar et al. (2020) is an example of this strategy, when a hierarchy in the action space is defined so that the action space can be restricted at first and gradually expanded later. Another example is Bamford and Ovalle (2021), in which the action space is organized into a sequence of sub-spaces to construct the Conditional Action Trees (CAT). The main differences between these two space structures are twofold. Firstly, the hierarchy in GAS is built based on the easiness/complexity of the divided spaces while the levels in CAT are defined based on the abstraction of actions. Secondly, in the former method, the relationship between divided spaces is inclusion, i.e., more restricted spaces are sub-spaces of less restricted ones; but in the latter, this relationship is considered to be sequential, i.e., actions in lower-in-rank spaces dependent on all previous links (see Figure 2).

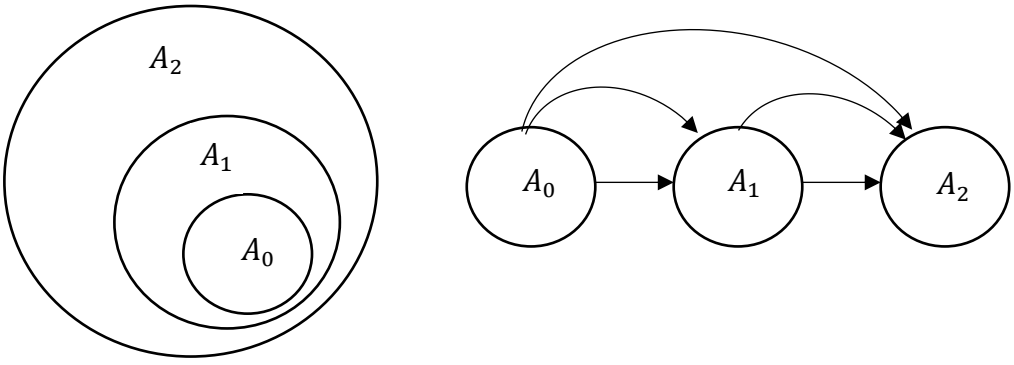

Figure 2: Structure of the action space in GAS (left) and CAT (right)

In this thesis, we inherit the idea and the general structure from GAS to establish the foundation for our methodology. Therefore, similarly, our proposed approach can be classified into the group of action space structuring methods. The difference between two papers lies in the targeted action spaces and thus, the strategies to define the hierarchies in the spaces. To be specific, there are two cases addressed in Farquhar et al. (2020): (i) continuous action spaces and the hierarchy derived by discretizing and (ii) combinatorial action spaces with *parallel* sub-tasks and the hierarchy derived by distance-based clustering. In comparison, we extend the application of GAS into a combinatorial action space with *sequential* sub-tasks. Additionally, we suggest a method to define the hierarchy in this case by utilizing the structure of the space and domain knowledge. By doing so, the sequential relationship between sub-spaces is transformed into the inclusion one, which allows us to implement GAS. In that sense, our approach stands at the intersection of GAS and CAT when it is based on the framework in the former but targets on the same type of action space like the latter.

On the other hand, we decide to choose GAS rather than CAT as the base for our model because of the two following reasons. Firstly, the implementation of CAT deploys an actor-critic DRL algorithm while GAS is built on top of a value-based one. Actor-critic methods learn two value loss functions (one for actor and the other for critic) simultaneously, resulting in an additional computation effort compared to value-based or policy-based methods. Moreover, DQN or value-based DRL algorithms are usually preferred when sampling efficiency is important (Boute et al. 2021), which is the case of our researched problem. At each step, the environment needs time to assess the validity of actions considering the new inventory state, thus, it is costly to gather new experiences. Secondly, in CAT, policies at sub-spaces are learned parallelly every time the critic is updated with trajectories collected by actors. However, in GAS, the values of sub-spaces are learned in a way that is sequential and increasing in level of difficulty. Therefore, GAS allows transfer learning from sub-spaces at lower levels to the ones at higher levels, which is not supported by CAT.

Finally, it is noticeable that two approaches defined above (i.e., ANN configuration and action space transformation) are not a substitution of each other when coming to dealing with combinatorial action spaces. Indeed, there are some presented methodologies that can be seen as the combination of both approaches. For example, all methods belonging to the group "*Adding supplemental modules"* aim to remove non-informative or invalid actions, which is a common technique in

action space transformation. Similarly, it is potential to combine our proposed method in this thesis with a compatible ANN structure among those previously mentioned to achieve a better performance, which can be considered as a direction for further research.

3 Theoretical Background

In this chapter, we provide a theoretical foundation for the methodology proposed in the thesis. The first section introduces about RL and Q-learning. Next, their extended versions, i.e., DRL and DQN, are presented in the second section. The last section summarizes main points of the original GAS approach as mentioned in Farquhar et al. (2020).

3.1 Reinforcement learning & Q-learning

A broad definition of RL has been given previously in section 2.2. According to Sutton and Barto (2018), there are two subjects in a RL system: the *agent* who makes the decision and the *environment* which interacts with the agent. Additionally, other components, namely a *policy*, a *reward signal* and a *value function,* are also important when constructing the RL system. A *policy* works like a "code of conduct" to the agent. It tells the agent what to do considering the current perceived state of the environment. A *reward signal* is immediate feedback from the environment after receiving an interaction from the agent, telling the agent whether its behavior is good or bad, and in many times, to what extent the behavior is good or bad too. A *value function*, on the other hand, focuses on a long-term return by indicating the potential future rewards the agent may receive if it stands at a certain state. While the reward can be observed directly from the environment, the value function must be iteratively estimated based on a record of past observations.

In this chapter, we consider a RL problem in an environment modelled as a MDP with states $s \in S$ and actions $a \in A$. Then, the state and action taken in a period t are s_t and a_t respectively. When the agent takes action a in state s, the expected reward is denoted as $r(s, a)$ and the probability of a transition to a state s' is denoted as $p(s'|s, a)$. A discount factor γ with $0 < \gamma < 1$ is also assumed.

For any policy π, there are two types of value functions we need to consider. A *state-value function* for policy π, as a function of states s, represents the expected return received when the agent is standing at a state s and follows the policy π. The formulation of this value function is presented below

$$V_\pi(s) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r(s_{t+k}, a_{t+k}|a_{t+k} \sim \pi(s_{t+k}))\middle| s_t = s\right]$$

Likewise, an *action-value function* for policy π, as a function of state-action pair (s, a), defines the expected return of taking an action a in a state s under a policy π. Its formulation is shown as below

$$Q_\pi(s,a) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r(s_{t+k}, a_{t+k}) \middle| s_t = s, a_t = a\right]$$

In a common RL problem, the goal is usually defined as finding an *optimal policy* which ensures the agent to achieve the maximum expected discounted reward. Mathematically, an optimal policy π^* is defined as any policy that satisfies $V_{(\pi^*)}(s) \geq V_\pi(s) \; \forall \, s, \pi$.

In order to find the optimal policy, one can solve the Bellman optimality equation for the optimal value functions, which are shown as below.

$$V^*(s) = \max_a \mathbb{E}[r(s_t, a_t) + \gamma \, V^*(s_{t+1}) | s_t = s, \; a_t = a]$$

$$= \max_a \left(r(s,a) + \gamma \sum_{s'} p(s'|s,a) \, V^*(s') \right)$$

$$Q^*(s,a) = \mathbb{E}\left[r(s_t, a_t) + \gamma \max_{a'} Q^*(s_{t+1}, a') \, | s_t = s, \; a_t = a \right]$$

$$= r(s,a) + \gamma \sum_{s'} p\,(s'|s,a) \; \max_{a'} Q^*(s', a')$$

Dynamic programming algorithms, such as policy iteration and value iteration, are able to compute the optimal policy by transforming Bellman equations into update rules to iteratively approximate the value functions. In comparison, Monte Carlo methods are learning methods based on actual experience sampling that does not require any prior knowledge of underlying dynamics of the environment like dynamic programming algorithms. Combining ideas of two above-mentioned approaches, temporal-difference learning is a simple and powerful technique to solve the RL problem. In detail, temporal-difference learning uses experience sampling (similar to Monte Carlo methods) and bootstrapping technique (similar to dynamic programming algorithms) to iteratively update the estimate of the state-value function. With α being a constant step-size parameter, the simplest temporal-difference update rule has the form of:

$$V(s_t) \leftarrow V(s_t) \; + \alpha[r(s_t, a_t) + \gamma \, V(s_{t+1}) \; - \; V(s_t)]$$

The term inside brackets is called temporal-difference error, which shows the difference between the new estimate $r(s_t, a_t) + \gamma \, V(s_{t+1})$ and the current estimate $V(s_t)$.

Q-learning is one of many methods built based on temporal-difference learning. This algorithm was regarded as a breakthrough in RL at the time it was introduced

in Watkins (1989) and it still plays a fundamental role in RL research at present (Sutton and Barto 2018). Q-learning is an off-policy algorithm, in which the policy is improved by using experiences from other policies. Therefore, its approximation process is independently from the policy being followed. The update rule in Q-learning is formulated as below:

$$Q(s_t,\, a_t) \leftarrow\ Q(s_t,\, a_t)\ + \alpha \left[r(s_t, a_t) + \gamma \max_a Q(s_{t+1},\, a) -\ Q(s_t,\, a_t)\right]$$

From the formula, we can see that Q-learning attempts to update the action-value function instead of the state-value one, which explains the name of this algorithm. It is proved that the estimate updated by this rule converges to the optimal action-value function (Sutton 1988). We refer readers to Sutton and Barto (2018) and Mazyavkina et al. (2021) for more detailed information on Q-learning and RL in general.

3.2 Deep reinforcement learning & Deep Q – network

Deep reinforcement learning is a combination of reinforcement learning and deep artificial neural networks. A DRL algorithm can be categorized into one of three main groups: value-based methods, policy-based methods, and hybrid methods, depending on which component is approximated by ANNs in the algorithm. Methods in the first group attempt to find the optimal policy through the approximation of value functions. The optimized loss is the temporal-difference error between the current estimate and the target value. On the other hand, policy-based methods directly approximate the optimal policy without going through the intermediate step of computing the optimal value functions. It is done by maximizing the expected cumulative reward obtained at the end of the run, i.e., solving the equation $\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_\pi[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)]$ with T being the length of one run. Finally, hybrid methods, such as actor-critic algorithms, integrate both value-based and policy-based methods. For example, in actor-critic methods, there are a policy-based actor (or actors) used to select actions and a critic network estimating value function to provide feedback on actions made by the actor(s).

Deep Q-network, proposed by Mnih et al. (2015), is a representative algorithm of value-based methods. In DQN, the optimal action-value function is approximated by an ANN, i.e., $\boldsymbol{Q(s, a; \theta)} \approx Q^*(s, a)$ with θ being parameters of the network. The authors solve the pre-existing problems of instability and divergence when using nonlinear approximators in RL methods by introducing two ideas. Firstly, motivated by the behavior of human memory, they define a mechanism termed

experience replay to randomly sample from a pool of stored past experiences and use these samples (or minibatches) to update the approximation. Secondly, a separate network called the target network is added to generate the target values in updating steps. This network has exactly the same structure as the online network (i.e., the one we want to update) but it is only updated every C step(s) by copying parameters from the online network. The update rule of the parameters θ_t with the integration of the target network θ^- is

$$\theta_{t+1} \leftarrow \theta_t + \alpha \left[r(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a; \theta_t^-) - Q(s_t, a_t; \theta_t) \right] \nabla_{\theta_t} Q(s_t, a_t; \theta_t)$$

Since being published, DQN has attracted a special attention from academia. There are many improved variants of DQN being introduced, among which we present two variants related to our thesis in this section.

- *Double Q-learning* (van Hasselt et al. 2016): In this variant, the value estimate in DQN becomes less over-optimistic. It is achieved by transforming the max operator in the target value in DQN into two separate steps which use two separate networks: the action selection with the online network $\operatorname{argmax}_a Q(S_{t+1}, a; \boldsymbol{\theta_t})$ and the action valuation with the target network $Q(s_{t+1}, a; \boldsymbol{\theta_t^-})$ (see Table 4).

Table 4: Comparison between DQN and Double DQN

Algorithm	Target value
DQN	$Y_t^{DQN} = r(s_t, a_t) + \gamma \max_{\boldsymbol{a}} \boldsymbol{Q}(\boldsymbol{s_{t+1}}, \boldsymbol{a}; \boldsymbol{\theta_t}) = r(s_t, a_t) + \gamma \boldsymbol{Q}(\boldsymbol{s_{t+1}}, \operatorname{\mathbf{argmax}}_{\boldsymbol{a}} \boldsymbol{Q}(\boldsymbol{s_{t+1}}, \boldsymbol{a}; \boldsymbol{\theta_t}); \boldsymbol{\theta_t})$
Double DQN	$Y_t^{Dbl\_DQN} = r(s_t, a_t) + \gamma \boldsymbol{Q}(\boldsymbol{s_{t+1}}, \operatorname{\mathbf{argmax}}_{\boldsymbol{a}} \boldsymbol{Q}(\boldsymbol{s_{t+1}}, \boldsymbol{a}; \boldsymbol{\theta_t}); \boldsymbol{\theta_t^-})$

- *Prioritized experience replay* (Schaul et al. 2015): In this variant, instead of being randomly uniformly sampled, an experience is more frequently selected if it has higher absolute temporal-difference error. In other words, the sampling places more importance on experiences with higher expected learning progress (see Table 5).

Table 5: Comparison between DQN and Prioritized DQN

Algorithm	Sampling strategy
DQN	transition $j \sim \boldsymbol{U(D)}$
Prioritized DQN	transition $j \sim \boldsymbol{P(j) = p_j^\alpha / \sum_i p_i^\alpha}$, with $p_j > 0$ being the priority of j and iteratively updated with the error $\lvert\delta_j\rvert$

Figure 3 shows an integrated version of DQN with modifications from Double DQN and Prioritized DQN.

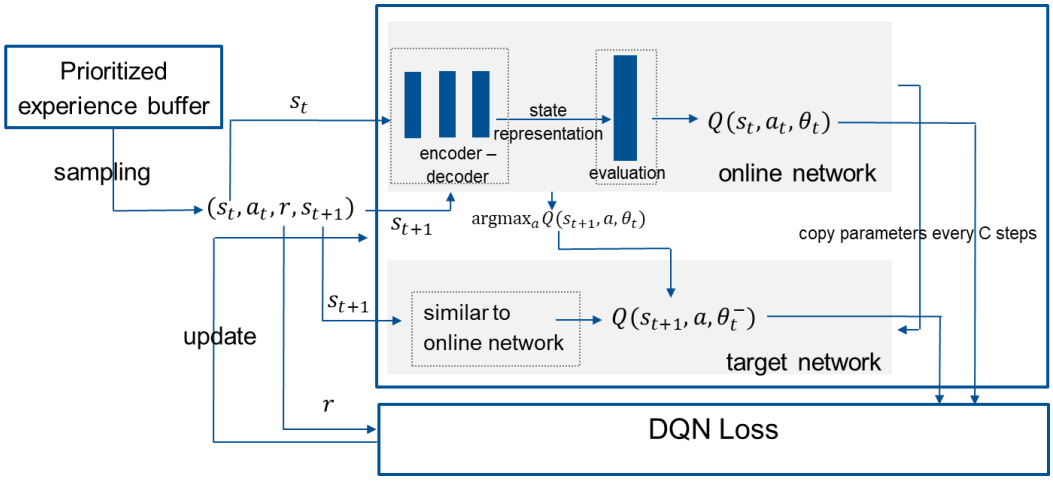

Figure 3: Example of DQN implementation with Double DQN and Prioritized DQN

3.3 Growing action space

GAS is a curriculum learning paradigm introduced by Farquhar et al. (2020) aiming to take advantage of the hierarchy in action space to accelerate Q-learning. This paper is motivated by the idea of "starting small" (Elman 1993), which describes the way human behave and learn in the real world. To address the inefficiency of the random exploration in an environment with complicated behavior, the authors suggest a learning schedule which can be summarized as follows.

- initially training the agent in more restricted action spaces to guide the learning toward meaningful experiences
- gradually expanding the action space being learned to the full space to find the optimal policy

Therefore, the implementation of GAS requires three requirements to be satisfied. The first one is that a hierarchy in the action space, i.e., rules to divide the whole action space into sub-spaces, must be defined. These rules need to strictly ensure inclusion and connection between sub-spaces. Inclusion relationship means that

more restricted action sub-spaces must be subsets of less restricted ones. Additionally, connections between sub-spaces are represented by the possibility to identify a *parent action* in more restricted sub-spaces for every action in less restricted action sub-spaces. While the first requirement enables the feasibility of implementing GAS, the others describe the conditions for the superiority of GAS over a random exploration strategy. To be specific, it is more efficient to apply GAS if learning in restricted action sub-spaces is easy and also able to lead to experiences which are more meaningful than the ones obtained by randomly exploring in the whole action space.

Example

To illustrate the concept of a hierarchy in GAS, we explain how GAS is used to solve Mountain Car game, which is an example provided in the original paper. The visualization of the example is provided in Figure 4. Mountain Car game is a game where a player takes control of a car standing at a foot between two steep mountains. The goal of the game is to drive the car to climb up the mountain on the right side to reach the flag. Because the car does not have an engine strong enough to allow it to finish the climb in one single pass, the player needs to drive the car back and forth multiple times to accumulate momentum in order to achieve the goal. To do so, in each step, the player chooses to apply a force, represented by any number ranging from -1 to 1, into the car. The sign of the number indicating the direction (0 means no force applied) and the absolute value is equivalent to the strength of the force.

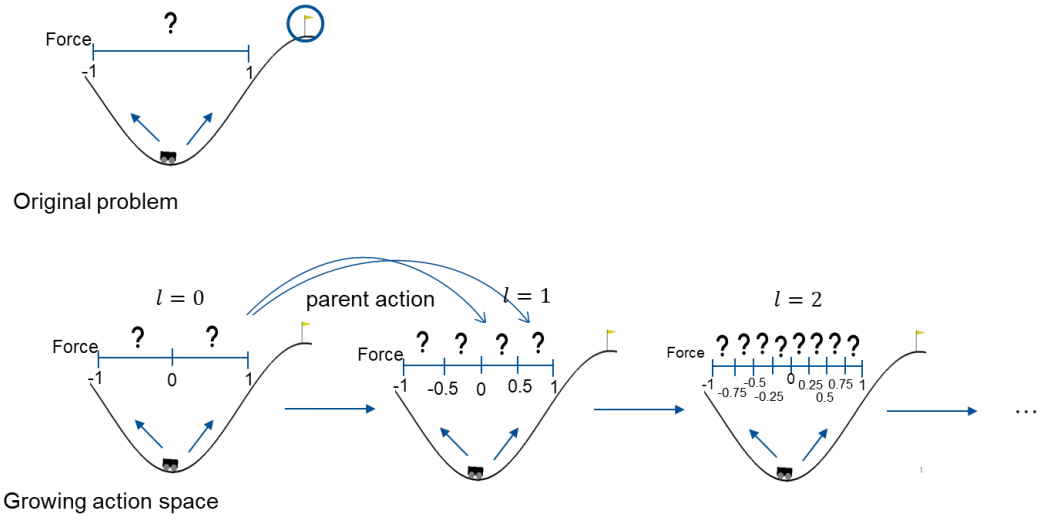

Figure 4: Example of using GAS to solve Mountain Car game

Mountain Car game can be modelled as a RL problem with a continuous action space. In this problem, a hierarchy in the action space can be simply created by discretizing the space with an increasing degree of granularity. In that sense, the most restricted action sub-space (i.e., at level $l = 0$) is considered as a discrete space with only two actions: turning left/no moving (i.e., a force in the range [-1, 0]) or turning right (i.e., a force in the range (0, 1]). Then, from this simplest action sub-space, at the next level $l = 1$, we divide each range of force by half, which leads to a less restricted action sub-space consisting of four discrete actions corresponding to four ranges of force. Similarly, we can continuously define less and less restricted action sub-spaces by improving the level of granularity. With this hierarchy, the inclusion relationship between derived sub-spaces is guaranteed. In addition, for any discretized continuous action in a sub-space at a level $l \neq 0$, we can define its parent action as an action at level $l - 1$ which corresponds to the range of force that includes the considered action's range. For example, actions with the ranges of force (0, 0.5] and (0.5, 1] at $l = 1$ have the same parent action which is the one with the range of force (0, 1] at $l = 0$ (see Figure 4).

Formulation

In the next part, we express GAS in its mathematical formulation. We consider an action space A divided into N action sub-spaces A_l, with $l \in \{0, \dots, N-1\}$, satisfying the hierarchical condition:

$$A_0 \subset A_1 \subset \dots \subset A_{N-1} \subseteq A$$

For every action $a \in A_l$ with $0 < l \leq N-1$, a parent action $parent_{l-1}(a)$ in the sub-space A_{l-1} is defined. Besides, for any restricted action sub-space A_l, the optimal action-value and state-value functions achieved by selecting actions in this sub-space are denoted as $Q_l^*(s,a)$ and $V_l^*(s) = \max_a Q_l^*(s,a)$, respectively. Accordingly, the estimated values functions are $\hat{Q}_l^*(s,a)$ and $\hat{V}_l^*(s)$ for an action sub-space A_l. Based on the structure of the action space, we obtain the following points.

- Considering a more restricted action sub-space A_i and a less restricted one A_j, because A_i is a strict subset of A_j, the optimal policy derived in the former is also a feasible policy in the latter. Meanwhile, the optimal policy in any space is defined as the one that is better than or equal to all other feasible policies. Therefore, we can conclude that the optimal value obtained in A_i is always less than or equal to that in A_j. In short, we have

$$V_i^*(s) \leq V_j^*(s) \;\; \forall s \;\; if \; i < j \text{ with } i, j \in 0, \dots, N-1 \text{ as } \; A_i \subset A_j$$

- The point above enables an iterative decomposition of the estimated value function, i.e.,

$$\hat{Q}^*_{l+1}(s,a) = \hat{Q}^*_l\big(s, parent_l(a)\big) + \ \triangle_l\,(s,a)$$

From the equation, we can see that there is a trade-off between the speed and the accuracy of the learning. When we let the agent learn in a simpler action sub-space, it is able to learn quicker, but the obtained result may be worse than that when the agent learns in a larger action sub-space. In that sense, the last term $\triangle_l\,(s,a)$ can be regarded as the benefit gained by the agent's effort to explore in a less restricted action sub-space. Another thing we can see from the equation is the role of parent actions in connecting divided sub-spaces. As action a may not be available in A_l, we can imagine a parent action $parent_l(a)$ like a reflection of its child action a (available in A_{l+1}) in the lower action sub-spaces A_l.

4 Problem Definition and MDP Model

The problem presented in this thesis is a modification of the one discussed in Pahr et al. (2021). As we focus our research on complex production action, we only consider blending states and actions, and use base-stock to select purchasing actions. For the purpose of completeness, we will briefly introduce the context of the problem as well as the derived MDP model in this chapter.

4.1 Problem definition

The problem described here is the inventory management of port wine products. Port wines are fortified wines that are produced from grapes grown only within the Douro region of Portugal. Like any other wine, port wines also go through the aging process, mostly in wooden casks and vats. Wines made in each harvest year are labelled and controlled separately. The quality, and thus the value, of port wines increases over the storage time (Cristovam and Paterson 2003). During maturation, the wine inventory faces a risk of decay. If an inventory unit is assessed to be decayed, it will be sold off in supermarkets (Pahr et al. 2021). Additionally, similar to Pahr et al. (2021), we assume that demands for branded products are stationary and deterministic, while demands for wine which is sold off in supermarkets due to being decayed or outdated are unlimited. Any unsatisfied demand in each period is lost without any penalty cost. We also assume that there is no loss during storage except for the decayed amount.

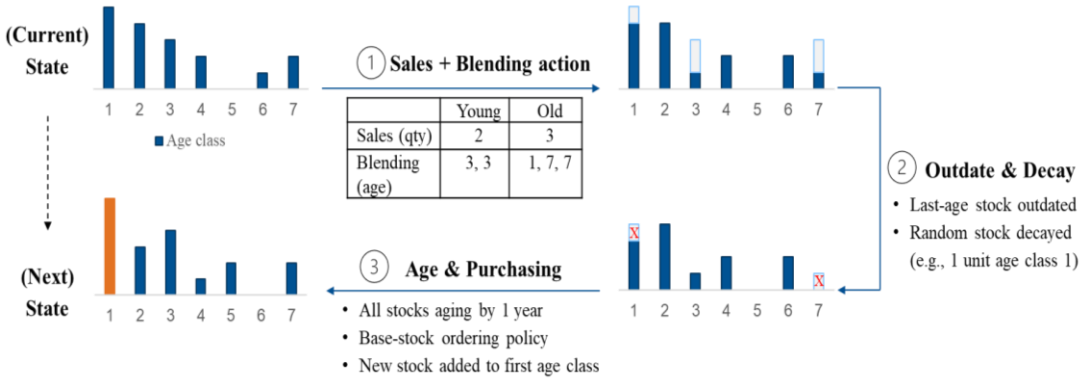

Figure 5: Changes in the port wine inventory system during one period

Figure 5 depicts main events during one period, here equivalent to one year, in the port wine inventory system. Every year, in January, sales quantities and blends are decided by producers to serve demands for branded port wine products, as shown

in the step 1 in the figure. For example, we assume there are two branded products, represented by two different demand streams. Producers will decide *how many* and *how* each demand stream is served. Any selected blend must satisfy two conditions: (i) the average age of the blend is at least the target age of the corresponding branded product, and (ii) the number of inventory units in the blend is equal to the number of sales units of the product. It is noticeable that strategic underfulfillment is possible and producers can use blending to change the age structure of inventory. Then, the remaining stock of the oldest wine, if any, becomes outdated and is sold in supermarkets with a cheap price to leave the space for a new batch. Besides, before ordering, the whole inventory is reassessed and if there is any inventory unit prone to decay, it will be sold off in supermarkets too. In the figure, these two events are summarized in the step 2 with removed inventory units non-filled and marked with red crosses. Next, in the harvest season in September, producers use base-stock ordering policy to decide the purchased amount of grape. The new port wine stock made from these grapes is regarded as the first age class in the inventory system, which is highlighted in orange color in the step 3 in Figure 5. Meanwhile, the other age classes mature and move to the older classes correspondingly by one year, i.e., to the right of the figure by one column. At the end of the year, a new state of the inventory is obtained, and the same process takes place again. In next sections, we will formulate this inventory system with the MDP approach and explain about main components of the derived model.

4.2 MDP model

This section presents states, actions, transitions, and value function of the MDP model derived from the case study mentioned previously.

4.2.1 States

The port wine inventory system is controlled based on age classes denoted as $i \in I = \{1, \dots, i^{max}\}$ with the granularity of years and i^{max} being the maximum age class kept in the inventory. We model a certain inventory state as a vector of length $|I|$, represented by $s = (s_1, \dots, s_{i^{max}}) \in S$ in which s_i indicates the discrete inventory level in age class i of state s. Because the capacity of the warehouse is limited, we assume that s_i must be between 0 and inv^{max} (i.e., the maximum inventory level for each age class), and so $s_i \in Inv = \{0, \dots, inv^{max}\}$. Then, it is

straightforward to obtain the size of the state space, i.e., $|S| = (inv^{max} + 1)^{i^{max}}$, which is exponentially dependent on the size of the age class set.

It is noticeable that in this problem, whether a production action is realizable or not depends on the current inventory levels. Instead of defining specific action spaces for each inventory state as in Pahr et al. (2021), we separate the action validation from action space establishment by considering the former as a part of the state of the environment. To be specific, we let the environment return a masking vector indicating the validity of every action in the space in accordance with the current inventory. Then, when an action is implemented, the environment provides feedback including a new state, which consists of the new inventory and its new derived action masking vector. By doing this way, the action space is stable regardless of the inventory state, which is required to build the neural network presented in the next chapters.

4.2.2 Actions

As mentioned in the previous section 4.1, the final production action consists of two sequential sub-tasks: (i) deciding sales quantities for branded products and (ii) deciding the corresponding blending recipes. Therefore, we denote the production action accordingly as $a = \left(sl_a, bl_{(sl_a)}\right) \in A$ with sl_a and bl_{sl_a} representing sales and blending sub-actions in a certain whole production action a respectively.

Next, we go further into each sub-action. If the port wine producers sell multiple branded products $b \in B = \{1, \dots, b^{max}\}$ and each of these products has the target age i_b^{trg} and faces a demand d_b, a sales sub-action in that case can be defined as $sl = \left(f_{1,sl}, \dots, f_{b^{max},\, sl}\right) \in SL$ with $f_{b,\, sl}$ being the demand fulfillment for branded product b in sales sub-action *sl*. Because the demand for branded products d_b is stationary and deterministic, the size of the sales sub-action set is equivalent to the product of all demands, i.e., $|SL| = \prod_{b \in B} d_b$. Meanwhile, we denote a blend corresponding to a sales sub-action sl as $bl_{sl} = \left(h_{1,\, bl_{sl}}, \dots, h_{i^{max},\, bl_{sl}}\right) \in BL_{sl}$ with $h_{i,\, bl}$ being the quantity of stock age class i used in the blending sub-action bl. While it is difficult to establish a formulation to compute the size of each blending sub-action set, there is one noticeable comment we can make here. Because the number of age classes i^{max} is larger than the number of branded products b^{max}, and the maximum inventory level inv^{max} is also larger than the demand of any branded product d_b, the size of the blending sub-action set is much larger than that

of the sales one. In short, blending brings a trade-off between flexibility and complexity in the decision-making process to the problem.

4.2.3 Transitions

After being subtracted the wine amount indicated by blending sub-action, the inventory goes through four steps before reaching a new state:

- **Outdating**: The remaining last-age stock after blending is outdated and sold in supermarkets, which is usually at a loss because of low price and high accumulated holding cost.

  Considering an inventory state $s = (s_1, \dots, s_{i^{max}})$ and a valid production action $a = \left(sl_a, bl_{sl_a}\right)$ which includes a blending sub-action $bl_{sl_a} = \left(h_{1,\, bl_{sl_a}}, \dots, h_{i^{max},\, bl_{sl_a}}\right)$, we denote the outdated quantity as $o_{s,a}$ with

  $$o_{s,a} = o_{s,(sl_a,\, bl_{sl_a})} = s_{i^{max}} - h_{i^{max},\, bl_{sl_a}}$$

  Additionally, the inventory state obtained immediately after outdating is called pre-decay inventory state and denoted as $s^{pre}(s,a)$. Pre-decay state is deterministic for each pair (s,a) and

  $$s^{pre}(s,a) = \left(s_1 - h_{1,\, bl_{sl_a}}, \dots, s_{i^{max}-1} - h_{i^{max}-1,\, bl_{sl_a}},\ 0\right)$$

- **Decaying**: If there is a probability $p^{decay}_{i,\,s,\,s'}$ that stock of age class i will change from s_i to s_i' because of decay ($p^{decay}_{i,\,s,\,s'} = 0$ if $s_i' > s_i$), the transition probability from an inventory state s to an inventory state s' if taking action a is given as follows

  $$p(s'|s,\ a) = \prod_{i \in I} p^{decay}_{i,s^{pre}(s,a),\,s'}$$

  It is noticeable that the decay risk of one port wine inventory unit decreases significantly over time. Based on that, similar to Pahr et al. (2021), we model the distribution of the relative risk of decay $p^{rD}_{i,\,s,\,s'}$ using the truncated discrete Weibull distribution (Nakagawa and Osaki 1975), i.e., $p^{rD}_{i,\,s,\,s'} \sim TDWei(q^{rD}, \beta^{rD})$. Then by multiplying the relative decay probability with a corrective term ol that represents the overall probability that an inventory unit decays during its lifetime, we obtain the actual decay probability for each inventory, i.e., $p^{decay}_{i,s^{pre}(s,a),\,s'} = p^{rD}_{i,\,s^{pre}(s,a),\,s'} \cdot ol$.

- **Aging**: The remaining stock in all age classes is transferred to the right next one, i.e., stock of age class i ($i < i^{max}$) is considered as age class $i + 1$ in the next period.
- **Purchasing**: In this problem, the ordering lead time is assumed to be zero. Every period, following the base-stock ordering policy, port wine producers need to replenish their inventory so that the total inventory level of all ages is equivalent to an order-up-to level BS. If the inventory state after decaying is s', the purchased quantity $g_{s'}$ is calculated as follows.

$$g_{s'} = BS - \sum_{i \in I} s'_i$$

4.2.4 Value function

Port wine producers sell two types of wine in the market: branded products and lower-class products made from outdated and decayed inventory units. The latter are sold in supermarkets at much lower prices compared to the former. Additionally, while the costs of goods sold increase linearly with the increase in the age classes, the prices in the supermarket channel only increase until a certain threshold age i^{thres}. Therefore, dependent on the age class of wine, contributions from lower-class products may be positive or negative.

Regarding the costs, there are two main streams of cost in this problem: storage and purchase. For each branded product, a holding cost corresponding to the target age, which represents the expected storage time, is considered when computing the profit contribution. Nevertheless, in the case that the average age of the blended products is higher than the desired target age, an additional holding cost must be deducted from the profit to reflect an increase in the actual storage time consumed to produce goods.

To sum up, the expected reward (or profit) by taking action a in state s is defined as below.

$$\begin{aligned} r(s,a) &= r\left(s, \left(sl_a, bl_{sl_a}\right)\right) \\ &= \sum_{b \in B} cb_b \cdot f_{b,\, sl_a} \\ &\quad + cs_{i^{max}} \cdot o_{s,\, a} + \sum_{s'} \left(p(s'|s,a) \cdot \sum_{i \in I} cs_i \cdot \left(s_i^{pre}(s,a) - s'_i\right) \right) \\ &\quad - hc \cdot \left(\sum_{i \in I} i \cdot h_{i,\, bl_{sl_a}} - \sum_{b \in B} i_b^{trg} \cdot f_{b,\, sl_a} \right) - \sum_{s'} p(s'|s,a) \cdot pp \cdot g_{s'} \end{aligned}$$

with cb_b = profit contribution of brand product b

cs_i = supermarket profit contribution of one unit of age class i

hc = holding costs

pp = purchasing price

Then, with γ denoting the discounted rate ($0 < \gamma < 1$), the state-value function for any policy π in state s is shown as below.

$$V_\pi(s) = r(s,a) + \sum_{s'} p(s'|s,a) \cdot \gamma \cdot V_\pi(s'')$$

with state s' being obtained from state s after taking action a and realizing decay risk, and state s'' being the deterministic state obtained from s' after purchasing.

In general, similar to most other research (Chen et al. 2021, Harsha et al. 2021, Meisheri et al. 2021, Moor et al. 2021), we define the goal here as finding a policy π^* maximizing the expected discounted value of future profit. It can be found by solving the derived Bellman optimality equation for the optimal value function V^*:

$$V^*(s) = \max_a \left\{ r(s,a) + \sum_{s'} p(s'|s,a) \cdot \gamma \cdot V^*(s'') \right\}$$

5 Methodology

In this chapter, we apply GAS into the case of a combinatorial action space with sequential sub-actions, such as the one in the problem researched in this thesis. Using domain knowledge as mapping functions, we are able to define a hierarchy in the action space enabling the implementation of GAS. The chapter is divided into two sections. The first one provides the mathematical formulation of our methodology and the second one describes the implemented model that integrates GAS and DQN, which hereafter is called GAS-DQN.

5.1 General formulation

At first, we formulate the actions and discrete action spaces considered in this chapter as follows. We focus on an action consisting of multiple sub-actions (or multiple sub-tasks), i.e., $a = (a_0^{sub}, a_1^{sub}, \ \dots, a_{N-1}^{sub})$. The corresponding action space, which also is a Cartesian production of multiple sub-spaces, is represented by $A = A_0^{sub} \times A_1^{sub} \times \ \dots \times A_{N-1}^{sub}$ and $a \in A$, $a_l^{sub} \in A_l^{sub}$, $l \in \{0, \ \dots, N-1\}$. The sub-actions are interdependent and implemented in the following order $a_0^{sub} \rightarrow a_1^{sub} \rightarrow a_2^{sub} \rightarrow \ \dots \rightarrow a_{N-1}^{sub}$. We also denote the sequence of sub-actions until l as $seq_l = \left(a_0^{sub}, a_1^{sub}, \ \dots, a_l^{sub}\right)$.

Additionally, we make an important assumption that it is possible to use domain knowledge to predict the next (not necessarily optimal) sub-action a_{l+1}^{sub} if the sequence of previous sub-actions seq_l is known. The assumption can also be expressed in the mathematical form as

$$\exists\, map\colon (seq_l\,,\ \cdot) \mapsto a_{l+1}^{sub}\ \forall\, l \in 0,\ \dots, N-2$$

Here, $(\cdot)$ denotes additional parameter(s) of the function (if any), such as current state of the environment. In this formulation, domain knowledge is converted into map, a mapping function that pairs a sequence of sub-actions with its next sub-action. With this assumption, we can define a hierarchy in the original action space to derive a sequence of restricted action sub-spaces satisfying the conditions of GAS as follows. A restricted action sub-space A_l consists of actions a_l satisfying

$$\begin{aligned} a_l &= \left(a_0^{sub}, \dots, a_l^{sub}, a_{l+1}^{sub}, \dots, a_{N-1}^{sub}\right) \\ &= \left(a_0^{sub}, \dots, a_l^{sub}, map_{(1)}\left(seq_l^{a_l},\ \cdot\right), \dots, map_{(N-1-l)}\left(seq_l^{a_l},\ \cdot\right)\right) \\ &\quad \text{where } seq_l^{a_l} = \left(a_0^{sub}, \dots, a_l^{sub}\right) \end{aligned}$$

with $seq_l^{a_l}$ being the sequence of sub-actions until l followed in action a_l and $map_{(1)}(\cdot) = map(\cdot)$ and $map_{(n+1)}(\cdot) = map\left(map_{(n)}(\cdot)\right)\ for\ n \geq 1$. In other words, each action a_l is formed by two parts: the first part includes all sub-actions originally **decided** (represented by the sequence $seq_l^{a_l}$), and the second part includes sub-actions **derived** by applying function map to the first part, i.e., $map_1\left(seq_l^{a_l},\ \cdot\right), \dots$. During training, only the decided part is learned and optimized, while the derived part is automatically built based on the mapping function defined in advance.

For an action $a_l \in A_l$, its parent action in the action space A_{l-1} is the one that has the same decided sequence of sub-actions as a_l but only until $l-1$, i.e.,

$$parent_{l-1}(a_l) = \left(a_0^{sub}, \dots, a_{l-1}^{sub}, map_1\left(seq_{l-1}^{a_l},\ \cdot\right), \dots, map_{N-l}\left(seq_{l-1}^{a_l},\ \cdot\right)\right)$$

With this definition, the inclusion relationship between restricted action sub-spaces at different levels is established, i.e., $A_0 \subset A_1 \subset \cdots \subset A_{N-1} \subseteq A$. Additionally, because $|A_l| = \left|A_0^{sub} \times A_1^{sub} \times \dots \times A_l^{sub}\right|$, the size of a certain action sub-space A_l may be much smaller than that of the full space, especially in the case that there exists an action sub-space $A_{l'}$ that is subsequent to A_l and much larger than A_l, i.e.,

$$|A_l| \ll |A| \quad if\ \exists l' : l' > l\ and\ |A_{l'}| \gg |A_l|$$

Example

To illustrate what we present above, we provide an example using the problem previously defined in section 4.1. We consider a case that there are two branded products b_1 and b_2 with target ages $i_{b_1}^{trg}, i_{b_2}^{trg} = 3,\ 5$ and demands $d_1, d_2 = 2, 2$. Besides, we also set $i^{max} = 7$. A final production action in the original problem includes two sub-actions: sales and blending. In this part, we also further divide the sales sub-actions by considering sales decision of each branded product separately. In other words, we consider a production action that has the form of $a = \left(f_1, f_2, bl_{(f_1,f_2)}\right)$, of which the terms inside the parentheses mean sales quantity of b_1, sales quantity of b_2 and blending sub-action corresponding the completed sales sub-action, respectively.

In this case, we define a mapping function based on two following rules. First, if the sales quantity of b_1 is known, we select the highest possible sales quantity for b_2. Second, when the sales decision is decided for both products, we use stock only from the target age classes to blend. It is noticeable that because the validity of a production action depends on the inventory state, a completed mapping function must also take the inventory state as an input. In this example, to simplify, we

assume that the inventory is full for all age classes. Thus, all actions are valid, and we can neglect the inventory state in the formula of our mapping function. The mathematical expression of this function is shown as below.

$$map(seq_l): \begin{cases} seq_0 = (f_1) \mapsto 2 \\ seq_1 = (f_1, , f_2) \mapsto (0{,}0, f_1{,}0, f_2{,}0{,}0) \end{cases}$$

With this mapping function, we can divide the full action space into three sub-spaces $A_0 \subset A_1 \subset A_2 = A$ with corresponding actions as follows

$$a_0 = \left(f_1, map(f_1), map\left(map(f_1)\right)\right) = \left(f_1{,}2, \left(0{,}0, f_1{,}0{,}2{,}0{,}0\right)\right) \in A_0$$

$$a_1 = \left(f_1,\ f_2,\ map\left((f_1, f_2)\right)\right) = \left(f_1, f_2, \left(0{,}0, f_1{,}0, f_2{,}0{,}0\right)\right) \in A_1$$

$$a_2 = a = \left(f_1, f_2, bl_{(f_1,f_2)}\right) \in A_2$$

An agent starts its exploration in a greatly restricted action sub-space A_0, which represents the sales decision of the younger branded product. At this stage, the agent focuses on learning value of each sub-action in the current sub-space. The value of a sales decision is estimated to be the value obtained when the agent selects this sales quantity and makes subsequent decisions (selling older branded product and blending) based on output of the mapping function. After learning in A_0 for a number of periods, the agent moves to an expanded sub-space A_1 where it learns the sales decisions for both products. Finally, at the later part of the learning, the agent is allowed to access the whole action space. Figure 6 summarizes what is mentioned above. It also highlights how parent actions are defined and provides examples in each sub-space level.

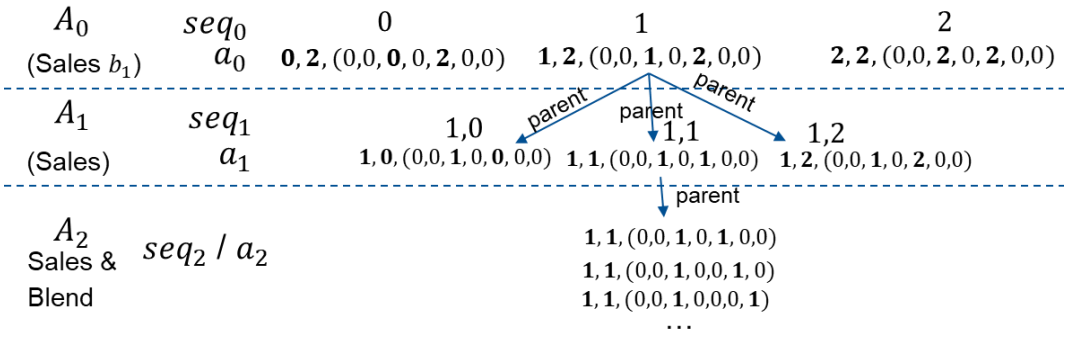

Figure 6: Example of GAS applied into a combinatorial space with sequential sub-actions

5.2 Model description

In our experiment, we divide the whole action space into two levels by combining the first two sub-space levels described in the previous example (see 5.1) into one. Because the combined sales sub-space can still be learned in a relatively short time,

we do so to reduce the number of layers in our neural network. Consequently, we obtain a new restricted action sub-space A_0 consisting of actions which are composed of a decided sales sub-action of both branded products and a derived blending sub-action. At the next level, we expand into the full action space $A_1 = A$, in which both sales and blending sub-actions need deciding.

We base our implementation of GAS-DQN on the original paper of Farquhar et al. (2020). The main components of the model are illustrated in Figure 7 below.

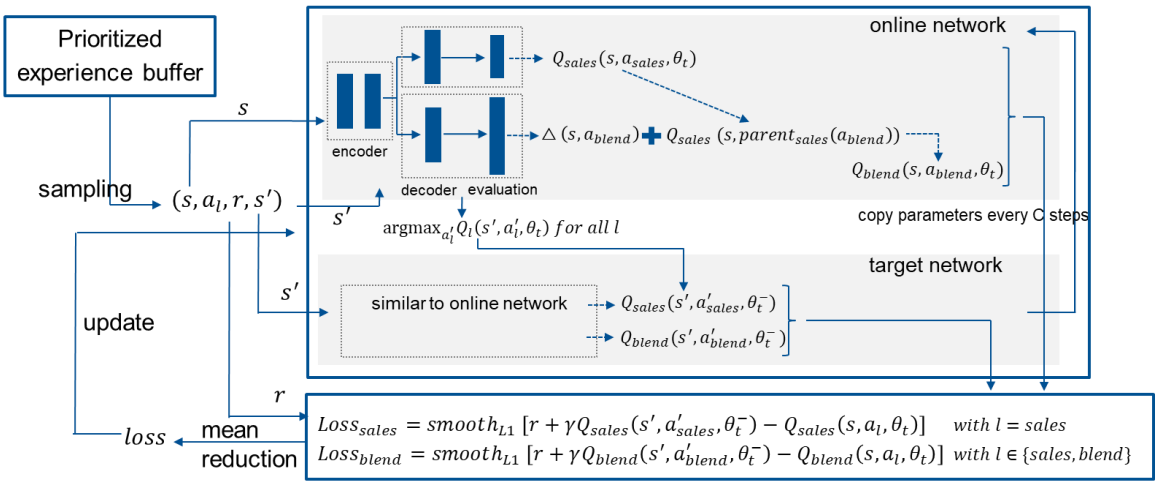

Figure 7: Implementation of GAS-DQN in the researched problem

The model consists of multiple fully connected ReLU layers. The input (inventory state) is encoded by the first two layers. After that, for each level of sub-space, there are a decoder and an evaluation layer correspondingly. By doing so, we can jointly learn the state representation and share it between all sub-space levels. The decoder takes the embedding obtained from the encoder and outputs a new embedding, which is subsequently used in the evaluation layer to produce the Q-value estimate. The dimension of the output of each evaluation layer is equal to the size of its corresponding action sub-space. It is noted that the estimate value of the restricted action sub-space A_0 is also used to estimate the value function of the larger action space A_1, as explained in section 3.3.

At every updating step, a batch of stored transitions is randomly sampled from the replay buffer to update the parameters of the online network. A transition having action drawn from the action sub-space A_l is used to update the value function approximation for the sub-space A_l and all higher-level ones $A_{\geq l}$. At the beginning of the training process, the agent acts and gains experiences in the most restricted action sub-space. After a number of periods, the agent moves to the next-level sub-

space and continues its learning. The number of periods that the agent spends to explore each sub-space is considered as a hyperparameter and defined in advance. A loss is calculated separately at each sub-space level by applying smooth L1 loss (PyTorch documentation 2019b) on the corresponding target and current estimate. The final loss is a mean reduction of losses at all sub-space levels.

In our model, we consider the exploration at two levels: *global* exploration and *hierarchical* exploration. Generally, exploration indicates the behavior of the agent when the agent selects actions without considering its current value function estimate. This behavior is opposite to exploitation, which means that the agent selects actions in a greedy manner (Sutton and Barto 2018). Based on this basic definition, in the case of a hierarchical sequence of action sub-spaces, we regard global exploration as exploration on the scale of the whole action (sub-)space currently considered. Meanwhile, hierarchical exploration refers to less extensive exploration that is restricted to only a part of the current action sub-space of which all actions have the same parent action as the greedy action. In other words, hierarchical exploration stands halfway between complete exploration (i.e., global exploration) and complete exploitation. In hierarchical exploration, the learned knowledge of the sub-actions at lower levels (i.e., parent action) is still taken into account.

The implementation of two-level exploration can be achieved by a simple modification in the common ϵ-greedy method. In the new algorithm which is called double ϵ-greedy, we consider an additional parameter ϵ^h and define $(1-\epsilon^h)$ as the conditional probability that the exploration is performed at a hierarchical level when an exploratory is required. Similar to ϵ, ϵ^h is also decreased gradually to represent the increasing preference for exploitation during the training process. For comparison purpose, we present pseudocode of both ϵ-greedy and double ϵ-greedy algorithms in Figure 8 and Figure 9 below.

Algorithm: ϵ-greedy action selection

initialize ϵ;
$p \leftarrow random()$;
if $p \geq \epsilon$ **then**
  $action \leftarrow \arg\max_a Q(a)$;
else
  $action \leftarrow$ a random action;
end
return $action$;

Figure 8: Pseudocode of ϵ-greedy algorithm

Algorithm: Double ϵ-greedy action selection in action sub-space A_l

initialize ϵ, ϵ^h;
if $l = 0$ **then**
  perform ϵ-greedy; `// double ϵ-greedy is not applicable in` A_0
else
  $greedy\_action \leftarrow \arg\max_{a \in A_l} Q(a)$;
  $p \leftarrow random()$;
  if $p \geq \epsilon$ **then**
    `// fully exploitation`
    $action \leftarrow greedy\_action$
  else
    `// exploration`
    $p^h \leftarrow random()$;
    if $p^h \geq \epsilon^h$ **then**
      `// hierarchical exploration`
      $A_l^h \leftarrow$ set of actions $a^h : a^h \in A_l, parent_{l-1}(a^h) = parent_{l-1}(greedy\_action)$;
      $action \leftarrow$ a random action in A_l^h;
    else
      `// global exploration`
      $action \leftarrow$ a random action in A_l;
    end
  end
  return $action$;
end

Figure 9: Pseudocode of double ϵ-greedy algorithm used in the model

6 Numerical Experiment

In order to assess the performance of GAS-DQN, we implement a numerical experiment and compare the results obtained from value iteration, DQN and GAS-DQN. The design of this experiment is presented in the first section and the results are reported in the next one. The last section is used to analyze the robustness of GAS-DQN approach by changing various factor in the initial experiment design and commenting on the corresponding changes in the results.

6.1 Experiment design

6.1.1 MDP model setting

Generally, in our numerical experiment, we consider a multi-age inventory system with a parameter setting as described in Table 6.

Table 6: Parameter setting of MDP model

Notation	Parameter	Value
i^{max}	Number of age classes	7
inv^{max}	Maximum discrete inventory level	6
b^{max}	Number of branded products	2
d_1, d_2	Demands of branded products	3, 3
$i_{b_1}^{trg}, i_{b_2}^{trg}$	Target ages of branded products	3, 5
BS	Order-up-to level	25
cb_1, cb_2	Profit contributions of branded products	333, 1000
i^{thres}	Threshold age until which supermarket price increases accordingly to age	3
hc	Holding cost	24
pp	Purchasing price	150
q^{rD}	Parameter in decay risk distribution	0.875
β^{rD}	Parameter in decay risk distribution	0.8
ol	Overall decay probability of an inventory unit during its lifetime	0.3

The first four parameters decide the size of the problem. In this setting, we have the size of the state space being $|S| = (inv^{max} + 1)^{i^{max}} = 7^7 = 823543$. The size of

the production action space is $|A| = 3456$, which is 216 times larger than the size of sales action space, i.e., $|SL| = 16$. This huge difference implies the complexity brought by blending action, which is a trade-off for the flexibility in serving demands.

The order-up-to level BS is decided based on a non-exhaustive analysis on the risk structure of the problem. In a static setting that does not consider decay risk and substitution, the steady inventory states before sales and blending and before purchase are presented as follows (see Table 7).

Table 7: Steady inventory states in the static setting

	Age						
	1	2	3	4	5	6	7
Before sales & blending	6	6	6	3	3	0	0
(at time *t*)	*Fulfill all demands + Outdating + Aging*						
Before purchase		6	6	3	3	0	0
(at time *t*)	*Order a quantity equal to sold quantity*						
Before sales & blending	6	6	6	3	3	0	0
(at time *t* + 1)	*Repeat the action cycle*						

It is easy to see that the optimal order-up-to level in the static setting is 24 (i.e., sum of inventory levels of all age classes after purchase). Then because of the existence of a small decay risk in our considered problem, we expect the optimal order-up-to level to be higher to cover for the risk. Therefore, we choose the order-up-to level $BS = 25$ to use in our experiment. Certainly, it is possible to do a more exhaustive search by running value iteration with a couple of potential values of BS and choosing the order-up-to level with the highest return. However, because ordering policy is not in our focus, we decide to follow a simpler approach.

The remaining parameters mentioned in the table are mostly similar to the setting in Pahr et al. (2021). Finally, in the same way, the supermarket profit contribution of one unit of age class i is calculated as

$$cs_i = -max(0, i - i^{thres}) \cdot hc \quad \forall i \in I$$

6.1.2 DQN & GAS-DQN setting

In our experiment, we implement two versions of DQN: common DQN algorithm as described in section 3.2 and GAS-DQN as in chapter 5. In both models, the

number of hidden units in the encoder are 128 and 64 in the first and second layer respectively. The numbers of hidden units in the decoder and evaluation are 64 each. The values of hyperparameters of DQN and GAS-DQN models in this thesis are selected mostly based on previous studies and/or non-exhaustive manual tunning. The setting of common hyperparameters shared between both models is shown in Table 8 while Table 9 presents hyperparameters used only in GAS-DQN model.

Table 8: Setting of hyperparameters shared between DQN and GAS-DQN

Hyperparameter	Value	Hyperparameter	Value
Total number of periods	100000	Adam learning rate	0.001
No. of periods / episode	1000	Adam epsilon	0.01
No. of runs	5	Percentile in AutoClip	10
Batch size	128	α start	0.5
Discount factor	0.999	α final	0.0
Replay buffer size	20000	ϵ initial	1
Target update interval	400	ϵ final	0.1
Env periods per model update	4	ϵ decayed until	20000

In each run, we organize the learning process of the agent into several episodes. When the agent starts a new episode, the initial inventory state is randomly selected, and the reward is also reset to 0. Each episode is equivalent to 1000 consecutive periods and is repeated 100 times in one run. Therefore, the total number of periods per one run is 100000. We perform 5 runs for each model.

The experiences obtained during the training process is stored in a reply buffer of length 20000 periods. After the first episode, the agent starts to update its online network every 4 periods. The weights of the target network are changed every 400 periods. Compared to the original GAS paper, we consider larger values of the replay buffer size and the interval to update the target network in order to increase the stability of the model (Fedus et al. 2020, Mnih et al. 2015). We use Adam optimizer to optimize the loss function. The initial learning rate is set to 0.001. After each updating step, we gradually decrease the learning rate by multiplying it with a factor equal to $1 - \frac{1}{Total\ number\ of\ periods}$ to prevent the oscillation at the end of the learning process (You et al. 2019). In order to stabilize the optimizer, we select a high value of Adam epsilon (i.e., 0.01) (PyTorch documentation 2019a) and apply

the adaptive gradient clipping technique AutoClip (Seetharaman et al. 2020) with 10^{th}-percentile cutoff.

We copy the setting of the hyperparameter α in prioritized experience replay from the original paper of Schaul et al. (2015). The value of α decreases linearly from the initial value of 0.5 at the beginning to 0 at the end of the training process to reduce the bias introduced to the estimate. It is noted that in our GAS-DQN model, since the restricted action sub-space is small and easy to learn, prioritized experience replay is not helpful at the start of the learning process and is only used when the agent acts in the full action space.

The value of ϵ (in both normal ϵ-greedy algorithm used in DQN and double ϵ-greedy algorithm used in GAS-DQN) is set to decay linearly from 1.0 to 0.1 over the first 20000 periods.

Table 9: Setting of hyperparameters used only in GAS-DQN

Hyperparameter	Value
Expanding action space after	20000
ϵ^h initial	0.5
ϵ^h final	0.1
ϵ^h decayed until	40000 (after the action space expansion)

Because we only consider two action space levels, the schedule that we use to grow the action space is simpler than that in the original GAS paper. We let the agent learn in the restricted action sub-space for the first 20000 periods and change into the full action space after that. As explained in the model description (see section 5.2), we use double ϵ-greedy algorithm with an additional hyperparameter ϵ^h. It decreases linearly from the initial value of 0.5 to 0.1 within 40000 periods after the action space is expanded (since the hierarchical exploration is only applicable in A_l with $l > 0$).

All models (including value iteration presented later) are trained using a computer with CPU of 2.7GHz x 8 cores and 16 GB RAM. The implementation of GAS-DQN and DQN is written in Python 3.8 using PyTorch framework and mainly adapted from the code provided in the original GAS paper.

6.2 Results

6.2.1 Value iteration & Optimal policy

We can find the optimal policy by applying the popular value iteration algorithm on the Bellman optimality equation for optimal value function V^* defined in the section 4.2.4. We use the termination condition as described in Odoni (1969) with the value of epsilon (i.e., the gap between minimum and maximum difference of two value estimates) equivalent to 0.01. Similar to previous research (Haijema et al. 2007, Hendrix et al. 2019), we observe that the value iteration algorithm takes a lot of time to compute the optimal policy. In our experiment, the solution converges after 17 iterations with the total running time being nearly 9 hours. At the final step, a simulation in which the agent follows the optimal policy is run for 5 million iterations. More insights into the behavior of the optimal policy are presented in Table 10.

Table 10: Result of the optimal policy from value iteration

Inventory state analysis		
Number of visited inventory state (% of state space)	9658 (1.2%)	
Most frequently visited inventory state (% of iterations)	[6, 4, 4, 4, 3, 2, 2] (1.6%) [6, 4, 4, 3, 3, 2, 2] (1.4%)	
Average inventory levels	[6.0, 4.5, 4.0, 3.5, 2.8, 1.7, **1.5**]	
Blending sub-action analysis		
Most frequently taken blending actions (% of iterations)	[2, 0, 0, 0, 1, 0, 2] (19.1%) [2, 0, 1, 0, 1, 0, 2] (10.5%)	
Average blending sub-action	[**1.55**, 0.07, 0.32, 0.54, 1.04, 0.24, **1.46**]	
Average outdating (in units)	**0.0**	
Sales sub-action analysis	**3yo wine**	**5yo wine**
Average sales (in units)	**2.2**	**3.0**
Underfulfillment (% of iterations)	**54.00%**	**0.09%**
Production action analysis		
Number of taken production (% of action space)	**111 (3.2%)**	
Average reward / CoV	**2831 / 10.5%**	

Looking at the two most frequently taken blending sub-actions, we observe a common pattern of using mainly stock from the youngest and oldest age classes. The reasons behind the preference of broad blending age spread (see Figure 10) are twofold. First, by using the oldest stock to blend, the agent reduces the number of outdated units which are sold at a loss in supermarkets. Second, in this problem, wine of younger age classes is significantly more prone to decay than wine of older age classes. Therefore, the usage of the youngest stock leads to a possible reduction in the number of decayed units.

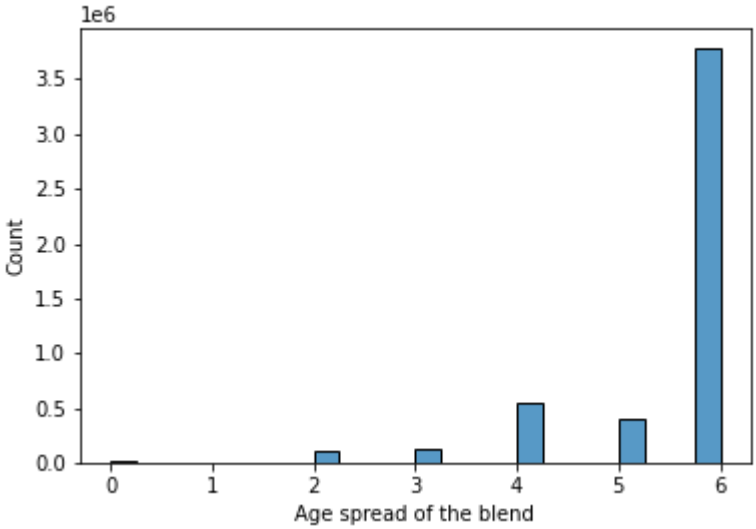

Figure 10: Distribution of age spread of the selected blends in the optimal policy

From Table 10, we also see that there is a strategic underfulfillment of the younger branded product to ensure the fulfillment of the older one. Because the profit contribution from the older branded product is much higher than that from the younger one, the agent gives priority to demands of the older in the case of a low total inventory level. Figure 11 shows a positive correlation between the sales quantity of the younger product and the current total inventory level before making production decision. When the total inventory level reaches the highest possible value (i.e., 25), demand is mostly fully served. Meanwhile, when the total inventory level decreases by one unit (between the range from 22 to 24 units), the number of younger wine units that are sold also decreases by one on average. Similarly, the sales quantity of the younger product is also positively correlated with the total sum of age of the current inventory, i.e., $\sum_{i \in I} i \cdot s_i$ (see Figure 12). We can see that if the total age sum of the current inventory increases, the sales quantity of the younger product is likely to increase too.

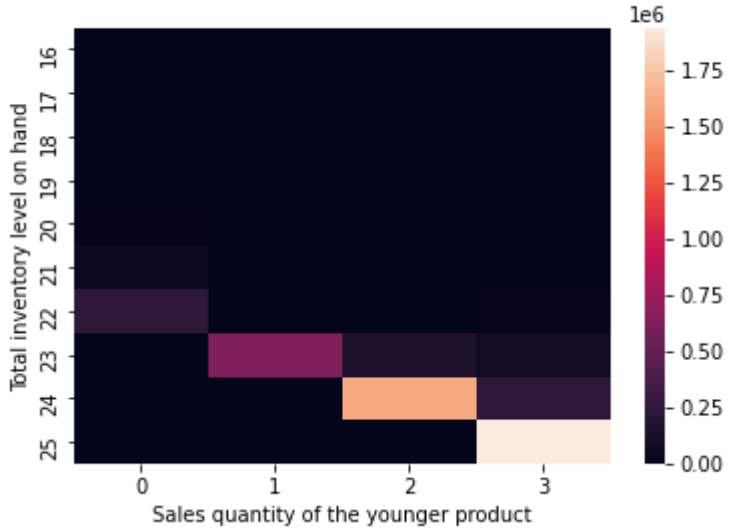

Figure 11: Relationship between sales quantity of the younger product and total inventory level

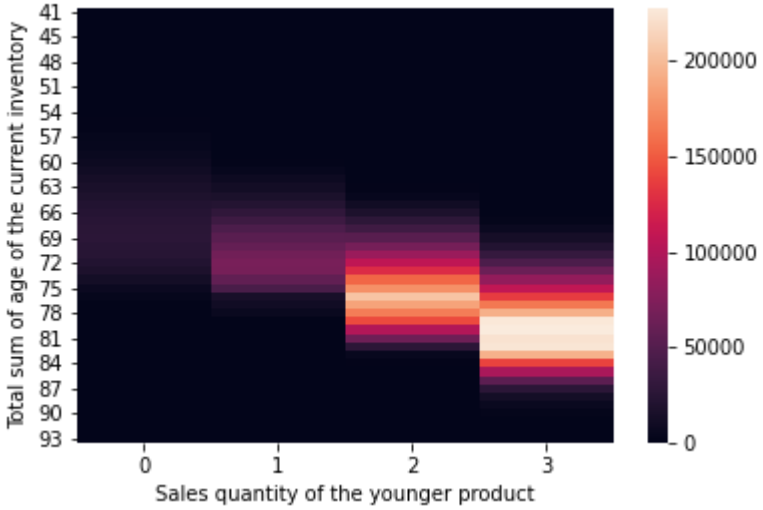

Figure 12: Relationship between sales quantity of the younger product and total sum of age

Finally, we notice that during the simulation, the agent visits only a small proportion of the state space. It suggests that a method searching over all the state space, such as value iteration, may be too expensive. Similarly, the fact that only a small number of actions are taken implies that a random strategy is inefficient in exploring the action space in this problem.

6.2.2 DQN & GAS-DQN

In this section, we present the results obtained by applying DQN and GAS-DQN algorithms into the researched problem.

One important component in the GAS-DQN is the mapping function, which represents the domain knowledge integrated into the algorithm. In all experiments

we present later, we use a mapping function defined by a searching process described as follows. Considering a current inventory state $s = (s_1, \dots, s_{i^{max}})$, for a specific sales sub-action $sl = (f_{1,sl}, f_{2,sl})$, the mapping function map returns a corresponding blending sub-action $bl_{sl} = (h_{1,\,bl_{sl}}, \dots, h_{i^{max},\,bl_{sl}})$ which is feasible in the current inventory state, i.e., $h_{i,\,bl_{sl}} \leq s_i \; \forall i \in \{1, \dots, i^{max}\}$. Then, we search among all potential blending sub-actions to find one satisfying the following criteria in the listed order.

- *Criterion 1*: Selected action has the highest possible number of inventory units from the last age class, i.e., $h_{i^{max},\,bl_{sl}} = \min(s_{i^{max}}, f_{1,sl} + f_{2,sl})$
- *Criterion 2*: Selected action has the lowest possible sum of ages, i.e., $bl_{sl} = \underset{bl}{\operatorname{argmin}} \sum_{i \in \{1,\dots,i^{max}\}} i \cdot h_{i^{max},\,bl}$ and $\sum_{i \in \{1,\dots,i^{max}\}} i \cdot h_{i^{max},\,bl_{sl}} \geq \sum_{b \in \{1,2\}} i_b^{trg} \cdot f_{b,sl}$
- *Criterion 3*: Selected action has the lowest total decay risk (or in other words, the highest probability that there is no decayed unit), i.e.,

$$bl_{sl} = \underset{bl}{\operatorname{argmax}}\, p^{decay}_{i,s^{pre}(s,a),\,s^{pre}(s,a)} \text{ with } a = (sl, bl)$$

The first criterion comes from the general knowledge of multi-age products' characteristic. The other two criteria are based on the understanding that is more related to the specific researched problem, i.e., the cost structure (which cost is considered) and the risk distribution (relation between decay probability and age classes). Overall, all mentioned domain knowledge is general and is not difficult to acquire.

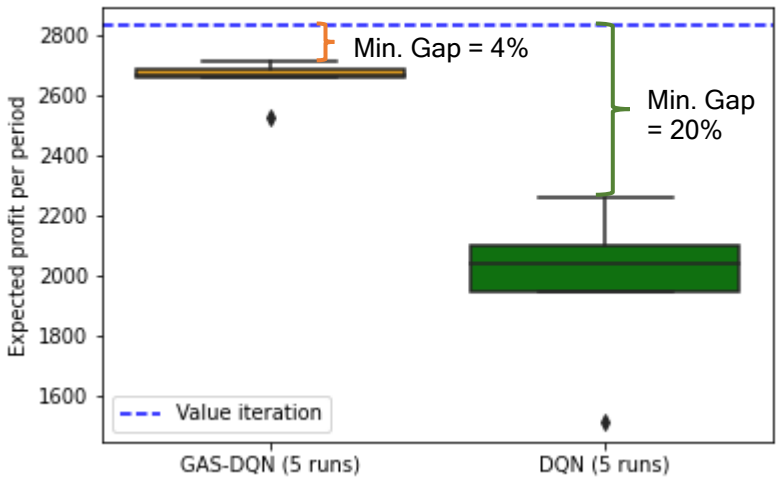

Figure 13: Performance of GAS-DQN and DQN

Using this defined mapping function, we run GAS-DQN, along with original DQN, with the setting and MDP as described in the previous section 6.1. Figure 13 shows the results obtained from running a simulation of 100000 periods with the model obtained in each run for each algorithm. We can see that the results of GAS-DQN are both higher and more stable than that of DQN. The median gap between the optimal expected profit obtained from value iteration and the expected profit from GAS-DQN is 6%, and the lowest gap is 4%. In comparison, the median gap and lowest one from DQN are 30% and 20% respectively, which are 5 times as large as those from GAS-DQN. Additionally, the standard deviation of the expected profit from GAS-DQN is also one-quarter the one from DQN, which proves the significant increase in stability of GAS-DQN.

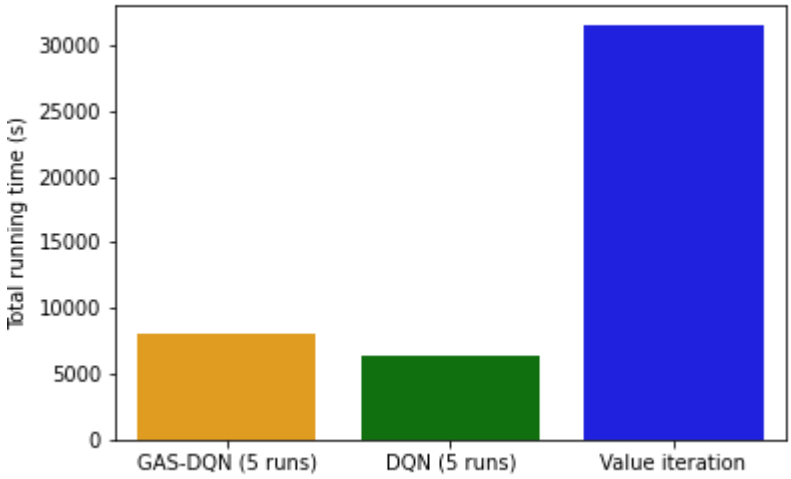

Figure 14: Running time of GAS-DQN, DQN and Value iteration

The running time of all three algorithms are presented in Figure 14. GAS-DQN requires more time than original DQN to complete training due to the introduction of new sub-spaces that leads to additional layers in the network, causing a small increase in computing time (i.e., nearly 6 minutes per run). Both deep reinforcement learning methods have much less running time than value iteration. It is reasonable because value iteration is a numerical exact method, which iterates over the whole state and action space, while (GAS-)DQN algorithms are approximation methods, which update non-exhaustively only based on sampled experiences.

Finally, Table 11 shows an overview into the approximated policies obtained by GAS-DQN in 5 runs. Similar to the optimal policy from value iteration, all policies from GAS-DQN have nearly no outdating and have high fulfillment levels of the older branded product. However, the average spreads in GAS-DQN policies are not as high as that in the value iteration's policy, which may explain the

underperformance of GAS-DQN compared to value iteration since we can see that the best run (run 1) has a blending spread significantly higher than the others. It is also noticeable that the coefficients of variation (CoV) of immediate rewards between 5 GAS-DQN runs are varied largely. The best GAS-DQN runs has CoV of immediate rewards equal to 6.8%, which is the lowest CoV value among all runs and also lower than the CoV obtained in value iteration.

Table 11: Summary of policies obtained by GAS-DQN in 5 runs

	Run				
	1 (best)	**2**	**3**	**4**	**5**
Avg. total inventory level	24.5	24.9	23.2	24.0	24.3
Avg. blending spread	4.6	3.7	3.5	3.0	3.7
Avg. outdated units	0.0	0.0	0.0	0.0	0.0
Avg. sales quantity of younger branded product	1.8	1.1	2.4	2.0	2.1
Avg. sales quantity of older branded product	3.0	3.0	2.8	2.9	2.9
Avg. reward	2714	2526	2663	2687	2669
CoV of immediate rewards in one run	6.8%	7.0%	29.6%	20.5%	16.5%

6.3 Robustness analysis

6.3.1 Mapping functions

In this section, we examine the robustness of GAS-DQN with regard to its mapping function. A mapping function represents understanding of a decision-maker about a specific problem. This understanding can come from various sources which have different levels of reliability, e.g., past research, experience, or intuition. Consequently, the gap between the output of a mapping function and the optimal decision depends on the level of the understanding supporting this function. In GAS-DQN, since the agent utilizes a mapping function to select subsequent sub-actions when learning at lower-level action sub-spaces (i.e., more restricted action sub-spaces), the choice of mapping functions affects the accuracy of the model's estimate at these levels. However, because at some point during the learning process, GAS-DQN still explores the whole action space regardless of the mapping

function, we expect that our model is robust to the choice of mapping functions to some extent.

To verify our guess, we conduct an experiment to evaluate the performance of two GAS-DQN models which use two different mapping functions to pair a sales decision with a corresponding blending decision. Table 12 provides an overview on these mapping functions.

Table 12: Comparison between two mapping functions

	Mapping function 1	**Mapping function 2**
Similarity	Take the outdating into account	
Difference	➢ Consider problem-specific factors (cost structure and risk distribution) ➢ Take more computing effort to find the best blend	➢ NOT consider problem-specific factors ➢ Easy to implement, less running time
Searching method	1. Maximum number of wine units from the last age class 2. Minimum sum of ages 3. Minimum total decay probability	1. Maximum number of wine units from the last age class 2. Take stock from the oldest age class to the youngest one

We consider the mapping function defined previously in section 6.2.2 as *mapping function 1*. According to the results from the previous study (Pahr et al. 2021), we know that mapping function 1 includes all important characteristics of the optimal decision, thus its output is very close to the optimal decision. Then, we simplify the first mapping function to create a new function (called *mapping function 2*) which is more straightforward and easier to implement. In mapping function 2, we only consider the first criterion (i.e., minimizing the number of outdated units) which is a general knowledge of multi-age inventory management and drop the last two criteria which are more specific to the researched problem. Additionally, instead of carrying out a thorough search like in mapping function 1, we select wine units in descending order of age class. To be specific, considering a current inventory state $s = (s_1, \dots, s_{i^{max}})$ and a sales sub-action $sl = (f_{1,sl}, f_{2,sl})$, mapping function 2 returns a corresponding blending $bl_{sl} = (h_{1,\,bl_{sl}}, \dots, h_{i^{max},\,bl_{sl}})$ of which each element is defined as follow.

$$h_{i,\,bl_{sl}} = \begin{cases} \min\left(s_{i^{max}}, f_{1,sl} + f_{2,sl}\right), & i = i^{max} \\ \max\left(0, f_{1,sl} + f_{2,sl} - \sum_{j=i+1}^{i^{max}} h_{j,\,bl_{sl}}\right), & otherwise \end{cases}$$

Although the outdating cost is minimized in mapping function 2, the cost of the derived blending action is high due to an additional holding cost caused by excessive usage of wine of older age classes. Therefore, we expect that the decision returned from mapping function 2 is not as good as that from mapping function 1.

Using these mapping functions, we train two GAS-DQN models and report the expected profits (obtained by simulation) in Figure 15. From the figure, we can see that the median and maximum value of expected profit of all runs between two mapping functions are almost the same. A more significant difference lies in the variability between runs. The model using the better mapping function (mapping function 1) achieves results which are more stable than the one using mapping function 2. The standard deviation of expected profits in the case of mapping function 1 is about 61% as large as that from mapping function 2.

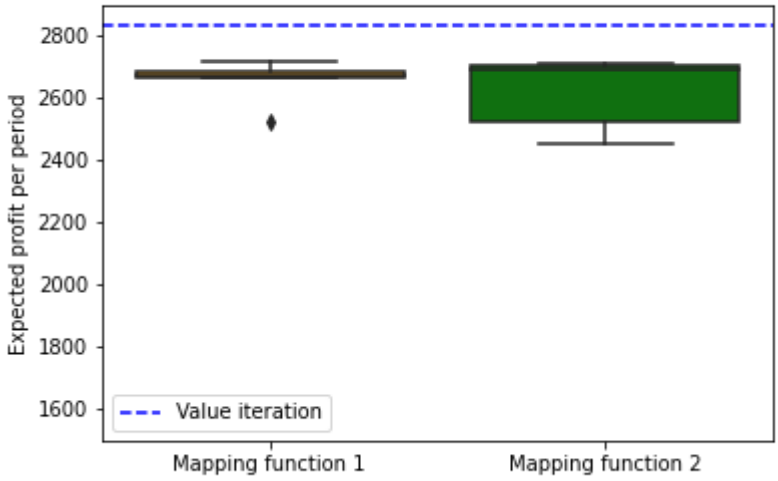

Figure 15: Performance of GAS-DQN with two different mapping functions

Figure 16 shows a closer look into the training process of each model by reporting the ranges of episode return (during training) in 5 runs, which is smoothed by calculating a moving average over the past 10 episodes.

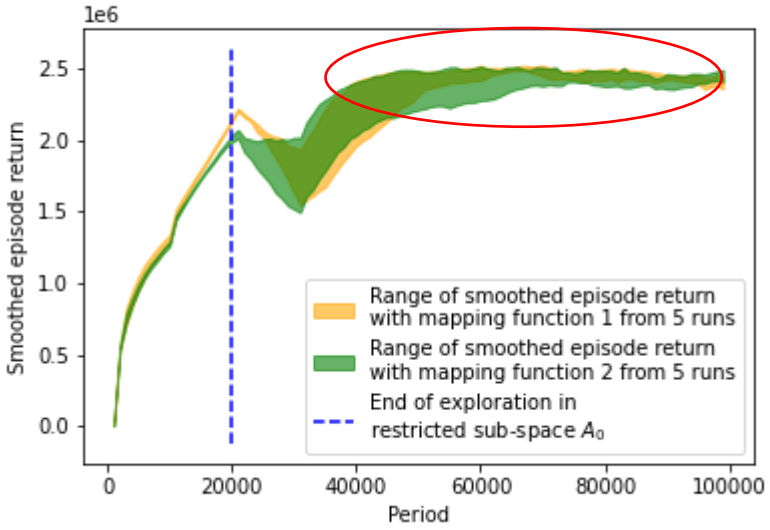

Figure 16: Range of smoothed episode return during training of GAS-DQN with two different mapping functions

As expected, when the agent completes exploration in the restricted action sub-space A_0, the episode return gained in the case of model with mapping function 1 is higher than that with mapping function 2. After that, both models experience a phase of exploration in the full action space with the episode returns varying greatly and after that, gradually improve their returns over time. However, we can see that the GAS-DQN model with mapping function 2 takes more time to reach a steady state than the one with mapping function 1 does, which also highlights the higher variance of the former, similarly to what we find from the previous figure.

6.3.2 Scalability

In this section, we validate the robustness of our model to the problem size by increasing the number of age classes stored in the inventory $|I|$ from 7 to 10. Accordingly, to adapt to the new range of age classes, we also change target ages of branded products $i_1^{trg}, i_2^{trg} = 4, 7$, as well as the order-up-to level $BS = 34$ (defined in the same way as presented in section 6.1.1).

Table 13: Space sizes and model's running times in two MDP settings

Setting	$\|S\|$	$\|A\|$	Running time of GAS-DQN (s)
$\|I\| = 7$	823543	3456	8130
$\|I\| = 10$	10000000	17020	31033

With these changes, the sizes of the new inventory state space and action space are $|S| = 10^7$ and $|A| = 17020$, which are nearly 5 times as large as those in the initial case of $|I| = 7$ (see Table 13). Due to this jump up in the dimensionality of the problem, it is not feasible to implement value iteration algorithm using current computing resource.

Regarding GAS-DQN, the model is still solvable in the new problem setting, yet its running time significantly increases due to the growth of the action space. As the number of nodes in the output layer of a Q-network is equivalent to the number of actions, the computational effort has a positive correlation with the size of action space. It may hamper the application of GAS-DQN in extremely large problems.

Next, we evaluate the performance of GAS-DQN in this new setting and compare it with the result from a greedy heuristic. The greedy heuristic is defined to also use the mapping function defined in section 6.2.2 to select the blending decision. Regarding the sales decision, it follows a greedy approach by choosing the highest possible sales quantity for each product with priority given to the older branded product. We illustrate the results of both methods in Figure 17 below. All GAS-DQN runs except one report significant higher expected profits than the greedy heuristic. The median gain from applying GAS-DQN instead of using the greedy heuristic is about 10.8% of the heuristic expected profit, with the maximum value being 15%. Nevertheless, the variance between expected profits obtained in different GAS-DQN runs increases significantly compared to the case of smaller number of age classes. It implies that as the problem becomes larger and more difficult, the agent is more likely to stuck in local optima.

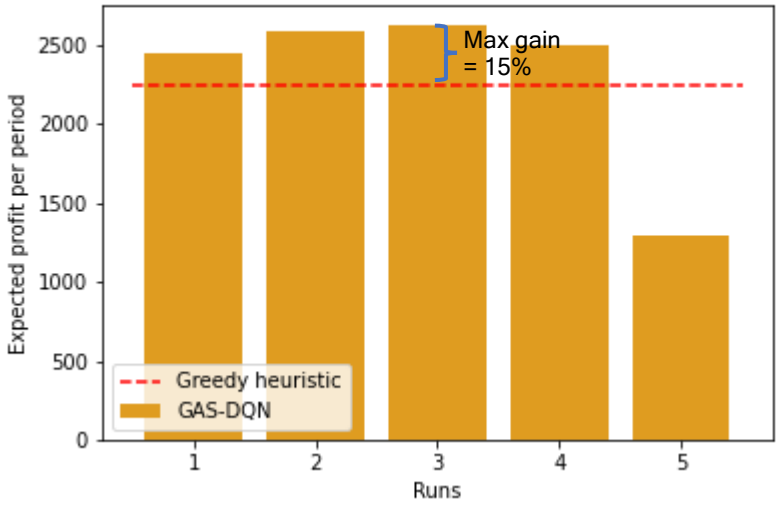

Figure 17: Performance of GAS-DQN and a greedy heuristic in the case of $|I| = 10$

6.3.3 Parameter variation: difference between profit contributions of two branded products

In all previous experiments, we observe a common behavior that the agent gives higher priority to the older branded product because of its much higher profit contribution in comparison to the other product. It even leads to a strategic underfulfillment of the younger product in order to reserve the stock to serve the future demand of the older one (see Table 10). In this section, we will examine how the performance of GAS-DQN changes when we adjust this factor.

To be specific, in the following experiment, we raise the profit contribution of the younger branded product cb_1 from 333 to 500 and reduce the contribution of the older one cb_2 from 1000 to 600. Consequently, the difference between cb_2 and cb_1 decreases, which means that the relative benefit gained by selling one unit of the older product instead of the younger one is lower.

By investigating the new optimal policy obtained from value iteration, we can see the significant impact of this new setting on the sales decision (see Table 14). The smaller difference in profit contributions of two products lessens the importance of fulfilling the older product's demand. In the new setting, more units of the younger wine are sold on average and the underfulfillment rate of this product is also lower, while the opposite trends are seen in the older product.

Table 14: Comparison of sales decisions in the optimal policies in two settings of profit contributions of branded products

	$cb_1 = 333$ $cb_2 = 1000$	$cb_1 = 500$ $cb_2 = 600$
Avg. sales quantity of younger product	2.2	2.3
Underfulfillment (% of iterations)	54.0%	48.1%
Avg. sales quantity of older product	3.0	2.9
Underfulfillment (% of iterations)	0.09%	5.7%

After training GAS-DQN with the new profit contribution setting, we still observe a great performance of our model (see Figure 18). The gap between the optimal expected profit and the expected profit obtained from GAS-DQN has the median

value of 7% and the minimum one of 1%. It shows that our proposed model is robust to the change in the structure of profit contributions of two products.

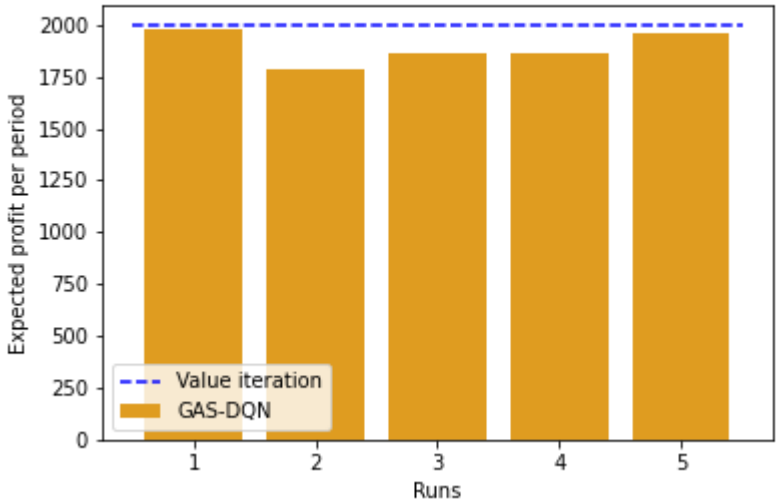

Figure 18: Performance of GAS-DQN in the case of $cb_1 = 500, cb_2 = 600$

7 Conclusion

In this thesis, we investigate the inventory management problem of port wine products. Because of the complexity brought by blending decision, this problem is a typical example of a multi-age inventory system with complex production action. To solve the defined problem, we take advantage of available domain knowledge and propose a methodology which adapts an existing learning curriculum called Growing Action Space (GAS) to the case of a combinatorial action space with sequential sub-actions. In our formulation, domain knowledge is embedded into a mapping function to output a next sub-action of a defined sequence of sub-actions. By doing so, we manage to structure the whole action space into several hierarchical restricted action sub-spaces that are easier to learn. By gradually expanding the level of action sub-space for the agent to explore, we are able to guide the agent toward more meaningful experiences at the beginning and transfer knowledge gained in more restricted to less restricted action sub-spaces. We evaluate the performance of our model using a numerical experiment designed based on the MDP model derived from our researched problem. The results present a significant improvement of our model compared to the original Deep Q-network. They also show a close gap between the optimal expected profit from value iteration and the expected profit from our model which is obtained in much less time than the former. Other experiments indicate that our model is robust to the choice of mapping function and changes in parameters to a certain extent.

The limitations of our model are twofold. The first limitation lies in the assumption of GAS model that the sub-spaces derived need to be easy and beneficial to be learned. In other words, there is a trade-off between benefits gained from structuring the action space and additional computation cost due to adding more layers to the network. In our researched problem, the nature of the problem allows an effective division of the whole action space. Because the sales action sub-space is very small compared to the compound production space consisting of sales and blending sub-spaces, we can let the agent learn in the sales action sub-space to accelerate the overall learning process without sacrificing a lot of computational time. Therefore, if we consider an action space that does not provide an appropriate hierarchy, it is not efficient to implement GAS. The second limitation is an intrinsic drawback of DQN. Because the number of nodes of the output layer in DQN is equivalent to the number of actions, the computation cost increases together with the increase in the size of action space. However, it may be solved by integrating GAS with other value-based DRL algorithms that specifically target large action

spaces such as Delarue et al. (2020) or Harsha et al. (2021), which is considered as a potential direction for future research.

In addition to what mentioned above, future work may focus on extending the application of GAS-DQN by considering inventory management problems of other multi-age products or problems with more complexity in its action, such as those integrating also ordering decision and/or allocation decision. Besides, it is also beneficial to investigate other approaches to solve the researched problem and compare the performance with our model.

References

Abbaspour A, Jahan A, Rezaiee M (2021) A simple empirical model for blood platelet production and inventory management under uncertainty. *J Ambient Intell Human Comput*. 12(2):1783–1799.

Bamford C, Ovalle A (2021) Generalising Discrete Action Spaces with Conditional Action Trees. *2021 IEEE Conference on Games (CoG). 2021 IEEE Conference on Games (CoG)* (Copenhagen, Denmark), 1–8.

Boute RN, Gijsbrechts J, van Jaarsveld W, Vanvuchelen N (2021) Deep reinforcement learning for inventory control: A roadmap. *European Journal of Operational Research*. Forthcoming.

Cappart Q, Moisan T, Rousseau L-M, Prémont-Schwarz I, Cire AA (2021) Combining Reinforcement Learning and Constraint Programming for Combinatorial Optimization. *Proceedings of the AAAI Conference on Artificial Intelligence*. 35(5):3677–3687.

Chaudhary V, Kulshrestha R, Routroy S (2018) State-of-the-art literature review on inventory models for perishable products. *JAMR*. 15(3):306–346.

Chen S, Li Y, Yang Y, Zhou W (2021) Managing Perishable Inventory Systems with Age-differentiated Demand. *Prod Oper Manag*. 30(10):3784–3799.

Civelek I, Karaesmen I, Scheller-Wolf A (2015) Blood platelet inventory management with protection levels. *European Journal of Operational Research*. 243(3):826–838.

Cohen MA (1976) Analysis of Single Critical Number Ordering Policies for Perishable Inventories. *Operations Research*. 24(4):726–741.

Cohen MA, Pierskalla WP, Yen H (1981) An analysis of ordering and allocation policies for multi-echelon, age-differentiated inventory systems. *TIMS studies in the Management Sciences*. 16(1):353–378.

Cristovam E, Paterson A (2003) PORT | The Product and its Manufacture. *Encyclopedia of Food Sciences and Nutrition* (Elsevier), 4630–4638.

Delarue A, Anderson R, Tjandraatmadja C (2020) Reinforcement Learning with Combinatorial Actions: An Application to Vehicle Routing. H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, H. Lin, eds. *Advances in Neural Information Processing Systems*, 609–620.

Deniz B, Karaesmen I, Scheller-Wolf A (2010) Managing Perishables with Substitution: Inventory Issuance and Replenishment Heuristics. *M&SOM*. 12(2):319–329.

Deniz B, Karaesmen I, Scheller-Wolf A (2020) A comparison of inventory policies for perishable goods. *Operations Research Letters*. 48(6):805–810.

Dulac-Arnold G, Evans R, van Hasselt H, Sunehag P, Lillicrap T, Hunt J, Mann T, Weber T, Degris T, Coppin B (2016) Deep Reinforcement Learning in Large Discrete Action Spaces. *CoRR*. abs/1512.07679.

Dulac-Arnold G, Levine N, Mankowitz DJ, Li J, Paduraru C, Gowal S, Hester T (2021) Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*. 110(9):2419–2468.

Elman JL (1993) Learning and development in neural networks: the importance of starting small. *Cognition*. 48(1):71–99.

Farquhar G, Gustafson L, Lin Z, Whiteson S, Usunier N, Synnaeve G (2020) Growing Action Spaces. III HD, Singh A, eds. *Proceedings of the 37th International Conference on Machine Learning,* Proceedings of Machine Learning Research, 3040–3051.

Fedus W, Ramachandran P, Agarwal R, Bengio Y, Larochelle H, Rowland M, Dabney W (2020) Revisiting Fundamentals of Experience Replay. III HD, Singh A, eds. *Proceedings of the 37th International Conference on Machine Learning,* Proceedings of Machine Learning Research, 3061–3071.

Gijsbrechts J, Boute RN, van Mieghem JA, Zhang D (2021) Can Deep Reinforcement Learning Improve Inventory Management? Performance on Lost Sales, Dual-Sourcing, and Multi-Echelon Problems. *M&SOM*. Forthcoming.

Goh C-H, Greenberg BS, Matsuo H (1993) Two-Stage Perishable Inventory Models. *Management Science*. 39(5):633–649.

Haijema R (2008) Solving large structured Markov Decision Problems for perishable inventory management and traffic control. PhD thesis, Amsterdam School of Economics Research Institute.

Haijema R, Minner S (2019) Improved ordering of perishables: The value of stock-age information. *International Journal of Production Economics*. 209:316–324.

Haijema R, van der Wal J, van Dijk NM (2007) Blood platelet production: Optimization by dynamic programming and simulation. *Computers & Operations Research*. 34(3):760–779.

Haijema R, van Dijk NM, van der Wal J (2017) Blood Platelet Inventory Management. Boucherie RJ, van Dijk NM, eds. *Markov Decision Processes*

in Practice, International Series in Operations Research & Management Science (Springer International Publishing, Cham), 293–317.

Harsha P, Jagmohan A, Kalagnanam J, Quanz B, Singhvi D (2021) *Math Programming based Reinforcement Learning for Multi-Echelon Inventory Management* (Appeared at the Deep Reinforcement Learning Workshop, NeurIPS 2021).

Hendrix EMT, Ortega G, Haijema R, Buisman ME, García I (2019) On computing optimal policies in perishable inventory control using value iteration. *Comp and Math Methods*. 1(4).

Hessel M, Modayil J, van Hasselt H, Schaul T, Ostrovski G, Dabney W, Horgan D, Piot B, Azar M, Silver D (2018) Rainbow: Combining Improvements in Deep Reinforcement Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*. 32(1).

Kanervisto A, Scheller C, Hautamaki V (2020) Action Space Shaping in Deep Reinforcement Learning. *2020 IEEE Conference on Games (CoG). 2020 IEEE Conference on Games (CoG)* (Osaka, Japan), 479–486.

Karaesmen IZ, Scheller–Wolf A, Deniz B (2011) Managing Perishable and Aging Inventories: Review and Future Research Directions. Kempf KG, Keskinocak P, Uzsoy R, eds. *Planning Production and Inventories in the Extended Enterprise,* International Series in Operations Research & Management Science (Springer US, New York, NY), 393–436.

Lin C-R, Buongiorno J (1998) Tree Diversity, Landscape Diversity, and Economics of Maple-Birch Forests: Implications of Markovian Models. *Management Science*. 44(10):1351–1366.

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

Meisheri H, Sultana NN, Baranwal M, Baniwal V, Nath S, Verma S, Ravindran B, Khadilkar H (2021) Scalable multi-product inventory control with lead time constraints using reinforcement learning. *Neural Comput & Applic*. Forthcoming.

Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, Petersen S, Beattie C, Sadik A, Antonoglou I, King H, Kumaran D, Wierstra D, Legg S, Hassabis D (2015) Human-level control through deep reinforcement learning. *Nature*. 518(7540):529–533.

Moor BJ de, Gijsbrechts J, Boute RN (2021) Reward shaping to improve the performance of deep reinforcement learning in perishable inventory management. *European Journal of Operational Research*. Forthcoming.

Nahmias S, Pierskalla WP (1976) A Two-Product Perishable/Nonperishable Inventory Problem. *SIAM J. Appl. Math.* 30(3):483–500.

Nakagawa T, Osaki S (1975) The Discrete Weibull Distribution. *IEEE Trans. Rel.* R-24(5):300–301.

Narvekar S, Peng B, Leonetti M, Sinapov J, Taylor ME, Stone P (2020) Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey. *Journal of Machine Learning Research*. 21(181):1–50.

Odoni AR (1969) On Finding the Maximal Gain for Markov Decision Processes. *Operations Research*. 17(5):857–860.

Oroojlooyjadid A, Nazari M, Snyder LV, Takáč M (2021) A Deep Q-Network for the Beer Game: Deep Reinforcement Learning for Inventory Optimization. *M&SOM*. Forthcoming.

Pahr A, Grunow M, Amorim P (2021) Deriving interpretable decision rules from optimal purchasing and blending policies in port wine inventory management. Forthcoming.

Panzer M, Bender B (2021) Deep reinforcement learning in production systems: a systematic literature review. *International Journal of Production Research*:1–26.

Parvez Farazi N, Zou B, Ahamed T, Barua L (2021) Deep reinforcement learning in transportation research: A review. *Transportation Research Interdisciplinary Perspectives*. 11:100425.

Pierskalla WP (2005) Supply Chain Management of Blood Banks. Brandeau ML, Sainfort F, Pierskalla WP, eds. *Operations Research and Health Care,* International Series in Operations Research & Management Science (Kluwer Academic Publishers, Boston), 103–145.

PyTorch documentation (2019a) Adam, https://pytorch.org/docs/stable/generated/torch.optim.Adam.html.

PyTorch documentation (2019b) SmoothL1Loss, https://pytorch.org/docs/1.9.1/generated/torch.nn.SmoothL1Loss.html.

Schaul T, Quan J, Antonoglou I, Silver D (2015) Prioritized Experience Replay.

Seetharaman P, Wichern G, Pardo B, Le Roux J (2020) AutoClip: Adaptive Gradient Clipping for Source Separation Networks. *2020 IEEE 30th*

International Workshop on Machine Learning for Signal Processing (MLSP) (IEEE).

Sultana N, Meisheri H, Baniwal V, Nath S, Ravindran B, Khadilkar H (2020) *Reinforcement Learning for Multi-Product Multi-Node Inventory Management in Supply Chains*.

Sutton RS (1988) Learning to Predict by the Methods of Temporal Differences. *Machine Learning*. 3(1):9–44.

Sutton RS, Barto AG (2018) *Reinforcement learning. An introduction*. Adaptive computation and machine learning (The MIT Press, Cambridge, Massachusetts).

van de Wiele T, Warde-Farley D, Mnih A, Mnih V (2020) *Q-Learning in enormous action spaces via amortized approximate maximization* (Appeared at the Deep Reinforcement Learning Workshop, NeurIPS 2018).

van Hasselt H, Guez A, Silver D (2016) Deep Reinforcement Learning with Double Q-Learning. *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence,* AAAI'16, 2094–2100.

van Zyl GJJ (1964) Inventory control for perishable commodities. North Carolina State University.

Watkins CJCH (1989) Learning from delayed rewards. Ph.D.Thesis, Cambridge, Univ.

Yang T, Tang H, Bai C, Liu J, Hao J, Meng Z, Liu P (2021) *Exploration in Deep Reinforcement Learning: A Comprehensive Survey*.

You K, Long M, Wang J, Jordan MI (2019) *How Does Learning Rate Decay Help Modern Neural Networks?*

You Y, Li L, Guo B, Wang W, Lu C (2020) Combinatorial Q-Learning for Dou Di Zhu. *Proceedings of the Sixteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment,* AIIDE'20.

Zahavy T, Haroush M, Merlis N, Mankowitz DJ, Mannor S (2018) Learn What Not to Learn: Action Elimination with Deep Reinforcement Learning. *Proceedings of the 32nd International Conference on Neural Information Processing Systems,* NIPS'18, 3566–3577.

Zhang T, Mo H (2021) Reinforcement learning for robot research: A comprehensive review and open issues. *International Journal of Advanced Robotic Systems*. 18(3):172988142110073.

Zhao Z, Liang Y, Jin X (2018) Handling large-scale action space in deep Q network. *2018 International Conference on Artificial Intelligence and Big*

Data (ICAIBD). 2018 International Conference on Artificial Intelligence and Big Data (ICAIBD) (Chengdu), 93–96.

Zheng S, Yue Y (2018) *Structured Exploration via Hierarchical Variational Policy Networks*.

Declaration of Authorship

I hereby declare that the thesis submitted is my own unaided work. All direct or indirect sources used are acknowledged as references.
I am aware that the thesis in digital form can be examined for the use of unauthorized aid and in order to determine whether the thesis as a whole or parts incorporated in it may be deemed as plagiarism. For the comparison of my work with existing sources I agree that it shall be entered in a database where it shall also remain after examination, to enable comparison with future theses submitted. Further rights of reproduction and usage, however, are not granted here.
This paper was not previously presented to another examination board and has not been published.

Munich, 01.02.2021

Thu Ha Dao