Optimal control in dynamic agri-food supply chains

by

Ashton Conrad Kappelman

B.S., Missouri University of Science & Technology, 2011

M.S., Kansas State University, 2018

AN ABSTRACT OF A DISSERTATION

submitted in partial fulfillment of the requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Industrial and Manufacturing Systems Engineering

Carl R. Ice College of Engineering

KANSAS STATE UNIVERSITY

Manhattan, Kansas

2022

ABSTRACT

The supply chains for agriculture and food (agri-food) related products face several challenges due to uncertainties and dynamic behaviors related to fluctuations in demand, uncontrollable environmental factors, sensitive quality concerns, and profitability within a low-margin industry. This research develops data-driven stochastic models and methods for solving important problems in agri-food supply chains.

Agri-food supply chains are well known to be dynamic and stochastic, yet most current models use simplified deterministic models. Instead, this research develops stochastic models and optimization methods that integrate ideas and techniques from machine learning, Big Data mining, and deep reinforcement learning to improve the supply chain performance and reduce food loss amidst many sources of uncertainty. Due to advances in computational capability and the availability of data in recent decades, it is now possible to create models with more details to better reflect the true supply chain dynamics and complexity.

This research first introduces a new generalized stochastic model for representing the dynamics of complex agri-food supply chains to optimize profitability by ensuring the quality of the end-product. A specific focus is placed on the tracking of the obtained quality level throughout the steps of the supply chain since this property highly predicts if the materials in the current steps are available for use in different potential final products or the final products' acceptability by the consumer.

It is recognized that these models must be able to be developed from existing data to capture the supply chain's complexity and quantify uncertain outcomes. This research accomplishes this by integrating data mining techniques with these models to determine the supply chain dynamics. Since deriving these dynamic behaviors from historical data can become computationally challenging, a novel approach that leverages Big Data mining tools and techniques is introduced and utilized to speed up running times without compromising the complexity or requiring more assumptions for the models.

Lastly, this research analyzes how traditional techniques perform versus approximation methods for agri-food supply chain models with rolling horizons and product degradation. This can be demonstrated through a mix and blend problem, a common operation in agri-food supply chains. A new neural network architecture called OR-Net is introduced as an efficient mechanism for modeling and solving sequential integer programs such as the mix and blend problem using deep reinforcement learning. OR-Net is designed specifically to focus on the orthogonal relationships that exist between an integer program's coefficients. Using numerical experiments, analysis is performed to evaluate the performance of OR-Net against stage-wise optimization and other approximation methods.

Optimal control in dynamic agri-food supply chains

by

Ashton Conrad Kappelman

B.S., Missouri University of Science & Technology, 2011
M.S., Kansas State University, 2018

A DISSERTATION

submitted in partial fulfillment of the requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Industrial and Manufacturing Systems Engineering
Carl R. Ice College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2022

Approved by:

Major Professor
Dr. Ashesh Kumar Sinha

Copyright

ABSTRACT

The supply chains for agriculture and food (agri-food) related products face several challenges due to uncertainties and dynamic behaviors related to fluctuations in demand, uncontrollable environmental factors, sensitive quality concerns, and profitability within a low-margin industry. This research develops data-driven stochastic models and methods for solving important problems in agri-food supply chains.

Agri-food supply chains are well known to be dynamic and stochastic, yet most current models use simplified deterministic models. Instead, this research develops stochastic models and optimization methods that integrate ideas and techniques from machine learning, Big Data mining, and deep reinforcement learning to improve the supply chain performance and reduce food loss amidst many sources of uncertainty. Due to advances in computational capability and the availability of data in recent decades, it is now possible to create models with more details to better reflect the true supply chain dynamics and complexity.

This research first introduces a new generalized stochastic model for representing the dynamics of complex agri-food supply chains to optimize profitability by ensuring the quality of the end-product. A specific focus is placed on the tracking of the obtained quality level throughout the steps of the supply chain since this property highly predicts if the materials in the current steps are available for use in different potential final products or the final products' acceptability by the consumer.

It is recognized that these models must be able to be developed from existing data to capture the supply chain's complexity and quantify uncertain outcomes. This research accomplishes this by integrating data mining techniques with these models to determine the supply chain dynamics. Since deriving these dynamic behaviors from historical data can become computationally challenging, a novel approach that leverages Big Data mining tools and techniques is introduced and utilized to speed up running times without compromising the complexity or requiring more assumptions for the models.

Lastly, this research analyzes how traditional techniques perform versus approximation methods for agri-food supply chain models with rolling horizons and product degradation. This can be demonstrated through a mix and blend problem, a common operation in agri-food supply chains. A new neural network architecture called OR-Net is introduced as an efficient mechanism for modeling and solving sequential integer programs such as the mix and blend problem using deep reinforcement learning. OR-Net is designed specifically to focus on the orthogonal relationships that exist between an integer program's coefficients. Using numerical experiments, analysis is performed to evaluate the performance of OR-Net against stage-wise optimization and other approximation methods.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I wish to thank my committee members who were more than generous with their expertise and precious time. A special thanks to Dr. Ashesh Kumar Sinha, my committee chairman and advisor for his countless hours of reflecting, reading, technical support and most of all for his encouragement of me to pursue my own research interests and ideas throughout the entire process. Thank you to Dr. Jessica Heier-Stamm for her developmental discussions and patience during my first year in the program as my work advisor, Dr. Todd Easton for convincing me to pursue this degree, Dr. Shing Chang for his guidance on appropriate technology and tools for my research, and Dr. Myong Kim for his willingness to serve on my committee on such short notice.

I would like to acknowledge and thank Dr. Bradley Kramer and the Department of Industrial and Manufacturing Systems Engineering for accepting me into the program, allowing me to conduct my research, and providing any assistance requested throughout the process. A very special thanks also goes to the members of the office staff in the IMSE department for their continued support and occasional laughter at my many failed attempts to be humorous. Finally, I would like to thank the instructors for my coursework at Kansas State University for stimulating my mind leading to many of the ideas I pursued for this research.

DEDICATION

I dedicate my dissertation work to my family and many friends, for without them this would have never been possible nor rewarding. To my loving parents, Bart and Annette Kappelman whose example of kindness and living with purpose were the guiding principles for my pursuit of this research. To my siblings Coleman, Chantel, and Brooks who have remained close to my heart and always encouraged me to be myself and follow my dreams.

I also dedicate this dissertation to my peers who have supported me throughout this process and taught me about the strength of the Kansas State family. I will always appreciate all they have done, especially Dr. Ali Tolooie for helping me explore and talk through my craziest research ideas, Dr. Cristiane Brazil for reminding me to enjoy this process and to be patient with myself, Dr. Mohammad Bisheh for pushing me with his relentless work ethic, the future Dr. Meghna Maity for helping me to trust in myself and always work with confidence, the future Dr. Sogand Sabahfar for having faith in me both inside the classroom and into industry, and Dr. Pedram Parandoush for partnering with me to arrange "the meeting" to set aside time to focus on enjoying our graduate school experience and getting to know our diverse colleagues.

To my previous supervisor at Bayer, Dr. Karl Bloss for his guidance and support in my decision to pursue this degree, and to my supervisors at US Foods, Mike Chen and Sean Dveris for their trust and flexibility in working with me to complete this dissertation in conjunction with a full-time job.

Lastly, I dedicate this work and give special thanks to my best friends and partners in crime Marissa and Sir Archers for being there with me throughout the closing of my doctorate program. You have been my greatest supporters and I simply would not have finished this work successfully without your endless love and encouragement. Thank you.

CHAPTER 1. INTRODUCTION

Feeding 9-10 billion people by the year 2050 safely and sustainably is one of the largest challenges facing mankind. The production, processing, and distribution of food through agriculture is difficult due to products that are perishable and subject to strict quality demands, regulations, and fluctuating consumer preferences. Their production is influenced by uncontrollable environmental factors, and they often incur large loss and waste factors which are of a growing global concern. Also, no two products' supply chains are the same and vary in their complexity (Georgiadis, Vlachos, & Iakovou, 2005). Addressing this challenge sufficiently requires several advancements in the areas of science, technology, and supply chain optimization.

Access to food can be improved by increasing food production output or by reducing inefficiencies in the supply chain and implementing practices that maintain quality. These suboptimal results are significant as approximately 25% of all food produced for human consumption in the US is lost in the supply chain before reaching the customer (Dou, et al., 2016). Food loss is characterized as a decrease in edible food intended for human consumption caused by inefficiencies in the food supply chains. This includes losses in both mass and quality from production, processing, storage, and distribution prior to retail. Food waste is food for human consumption being discarded at the retail or consumer level due to spoilage, supply imbalance, or human choice.

Agri-food supply chains also consume many precious resources including land, water, energy, and human capital, which means food loss and waste squander these resources at a time when they are also necessary for other societal needs such as health, sanitation, and transportation. If food wastage was a country, its carbon footprint would rank 3rd globally, putting it behind only the US and China (FAO, 2013). Its ground and surface water consumption would be 250 km^3 or almost half the annual discharge of the Mississippi River. It also occupies roughly 1.4 billion hectares of land, which is 8 times the arable land area in the US. Lastly, the estimated economic impact of food wastage is $1 trillion USD, which would rank it 19th in GDP among countries in 2019.

During the COVID-19 pandemic, consumers learned the harsh reality that agri-food supply chains are subject to disruptions the same as many other sectors of business, even though food and agricultural products are critical for daily life (Aday & Aday, 2020). Actions such as the restrictions placed on restaurants and food service workers, drastic and immediate changes in consumer behaviors, and occasional interruptions of food production and processing operations all had massive impacts to agri-food supply chains. Those impacts became abundantly clear when many grocery stores were found with empty shelves or began placing purchase limits on certain everyday necessities. Redirecting substantial food supplies from restaurants or schools to grocery for household consumption also led to increases in food wastes, especially for low shelf-life items like fresh produce and dairy.

From a financial impact perspective, agri-food supply chains also account for a large component of the global economy. Agricultural products contribute a noticeable portion of the US export economy, totaling nearly $140 billion in 2018 (BIFAD, IFPRI, & APLU, 2019). Those exports generated an additional $260 billion in economic activity within the US, netting a total increase in economic output of $400 billion. The estimates for these exports have grown substantially even in the past 4 years to a value of over $196 billion in 2022 (Kenner, Jiang, & Russell, 2022). Granular products like soybeans, corn, wheat, feed, etc. accounted for over 195 million metric tons in exports. Most of these exports must also travel long distances with 72% (in value) of these exports leaving the US going somewhere other than Canada or Mexico. These figures prove that even small incremental improvements in these supply chains would have economic impacts on the order of hundreds of millions of dollars. In summary, pushing agri-food supply chains closer to their global optimal performance targets can have very large societal, environmental, and economic benefits.

Much-needed improvements in agri-food supply chains might come in the form of increases in quality or production output, reductions in supply chain inefficiencies, or operations that avoid myopic decisions to remove unwanted food loss and waste. Developing a stronger understanding of the impact of decisions that affect the output quantity and quality requires both data and methods that make use of this data for the purpose of optimizing the supply chain. Data collection practices have greatly improved over the last decade, so effective techniques that take advantage of this data are a next major step. Most of the existing models for optimization within agri-food supply chains are overly simple and deterministic, lack the flexibility for many applications, or they are not computationally efficient for larger problems. This thesis aims to improve performance in agri-food supply chains through the development of stochastic optimization models and advanced data-driven methods to help address these issues as they relate to food and agriculture production, processing, and distribution problems.

1.1. Research Motives and Objectives

The objective of this research is to introduce a new stochastic model for representing the dynamics of complex agri-food supply chains to optimize profitability by ensuring the quality of the end-product. A specific focus is placed on the tracking of the obtained quality level throughout the steps of the supply chain since this property highly influences when the materials in the current steps are available for use in different potential final products and the final product's acceptability by the consumer. These considerations make quality the ideal tracking mechanism for optimizing the supply chains’ profit and marketability while reducing food loss (Nagurney, Besik, & Yu, 2018).

Agri-food supply chains are well known to be stochastic, yet most current models use simplified deterministic models. Instead of using deterministic techniques, this proposed research develops stochastic models and optimization methods that integrate machine learning, big data mining, and deep reinforcement learning to improve supply chain performance and reduce food loss. Due to advances in computational capability and the availability of data in recent decades, it is now possible to create models with more details to better reflect the true supply chain dynamics and complexity. The decision options considered in these supply chains could include the utilization of different raw materials, innovative technologies, processing equipment, transportation modes, and storage options available that are often unique to each specific step. For an example of the decisions that affect agri-food supply chains, see Figure 1.1.

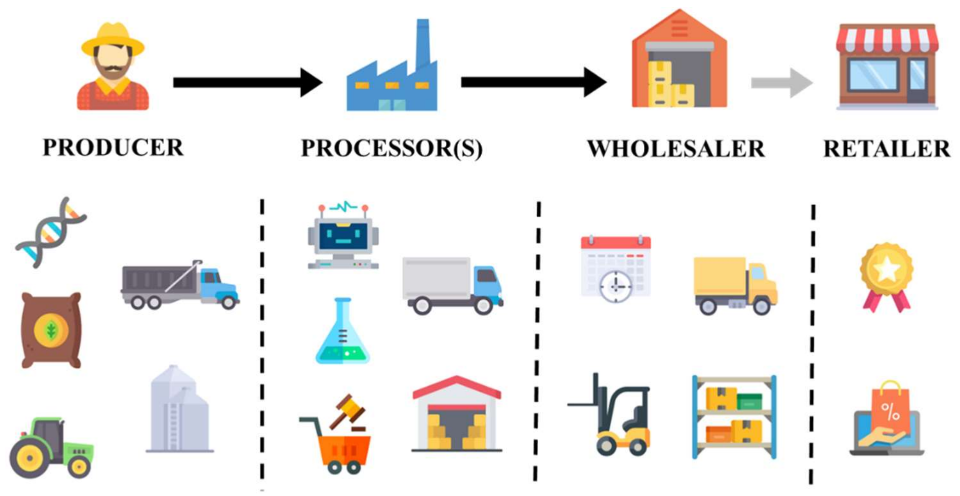


Figure 1.1 Example of steps and variety of decisions involved in agri-food supply chains

The first goal of this research is to design a generalized dynamic agri-food supply chain model that considers uncertainty in production, storage, and demand processes and provides optimal set of sequential decisions to optimize the quality of the final end-product. Next, we recognize that we desire for this model to be developed and informed from existing data to capture the supply chain's complexity and uncertain outcomes. Integrating data mining techniques enable this model to be truly data-driven. Since this is computationally challenging, big data mining techniques will be utilized to speed up running times.

Lastly, we consider a set of problems that many of the most common solution methods are unable to handle, such as those with continuous state spaces and unforeseeable or uncertain future outcomes over many periods. This final research task analyzes how traditional techniques found in discrete optimization problems perform versus approximation methods for a large-scale food supply chain model with rolling horizon. This can be demonstrated through a mix and blend problem, which is a very common operation in agri-food supply chains.

The merit of this proposed research is to advance and develop data-driven stochastic models in agri-food supply chains with generalized models and novel techniques. The models when paired with these techniques can serve as a framework for many problems in agri-food supply chains without loss of the supply chain's complexity or dynamics. These approaches can also be applied to other supply chains and sequential optimization problems that are present in many industries.

1.1.1. Develop a generalized stochastic model for agri-food supply chains

The first objective is to develop a general stochastic model for dynamic agri-food supply chains and identify challenges that industry practitioners would have when trying to solve this model. This model needs to have the flexibility for being applied to problems including multiple decision-making stages and consider rolling time horizons. The following tasks detail how this research objective can be met.

Dynamic agri-food supply chain processes are modeled as a series of sequential steps. Common steps for any problem could include producer, one or more processors, wholesaler, and retailer. Steps for an operational problem might describe the individual stages of development within the decision maker’s control. The decisions at each step include supplier selection and settings for their process parameters and they occur at the decision epochs which can be at specified time periods or supply chain steps. This series of sequential decisions can be formulated as a Markov decision process (MDP) where the resulting quality level of the product at every step is stochastic. Also, the decision maker must consider that this quality level at each supply chain step has a minimum accepted level and any product that does not meet the minimum accepted level is rejected.

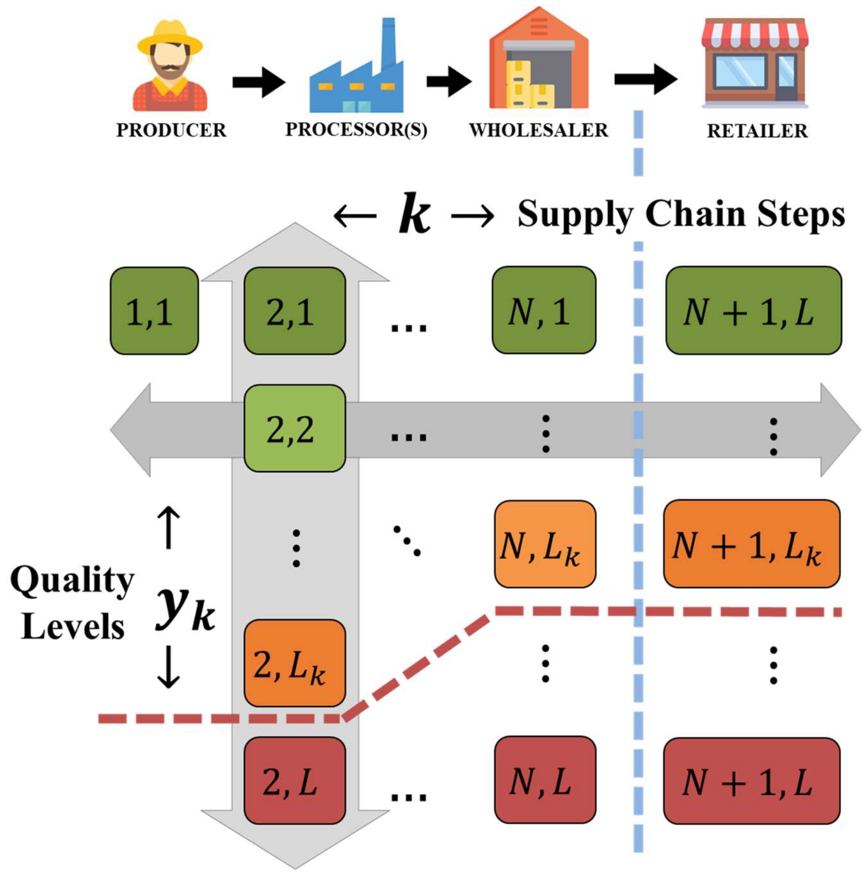


Figure 1.2 State space diagram of our supply chain.

The goal of this MDP model is to solve for an optimal policy, which is mapping of optimal actions to all possible future states. To ensure faster convergence to the optimal policy, this work should show monotonicity in the optimal value function by proving that it is non-decreasing for all states. This MDP model is then solved for the optimal policy using Policy Iteration.

1.1.2. Integrate data mining methods for understanding supply chain dynamics

It is recognized that a common weakness with using a MDP as a modeling framework is that most related problems in the literature assume a transition matrix is provided. In this research task, data mining methods are integrated into the MDP model to determine the supply chain dynamics and probabilistic outcomes that connect states and actions. The result is a data-driven model that can discover optimal decision policies purely from historical data.

To determine the transition matrix from historical data, this task first investigates Bayesian networks. The goal of this Bayes net is to learn a classification function that is trained on data elements. By appropriately arranging the input and output data vectors that we train our Bayes net on, the outcome classification function the Bayes net learns is simply our transition probability distribution. One of the challenges with solving for exact inference with a Bayes net is the computation time (Roth, 1996). The next goal of this task is to develop a novel technique that leverages the power of Big Data mining techniques and parallel computation to improve the running time when solving for the transition matrix of the MDP.

Association rule mining is a method in machine learning that aims to discover relationships between variables of interest. It does this by exploring data and uncovering strong rules between these variables based on the frequency of their common existence in a set of items, called a transaction. These strong rules, called association rules (AR), are directional and can be used to discover the likelihood that one item will exist in a random transaction given another item of interest is in the transaction (Agrawal, Imielinski, & Swami, 1993). These rules are commonly extracted from point-of-sale systems, such as those in supermarkets and are also used for recommendation systems. The computing framework Spark is utilized for extracting these rules from data using procedure can be seen in Figure 1.3. The key step in this procedure is the transaction assignment, which is where data elements are preprocessed into transactions. The independent elements of these transactions are connected by association rules whose properties are equivalent to their unique probability distributions in the P-matrix (i.e., our MDP's transition model).

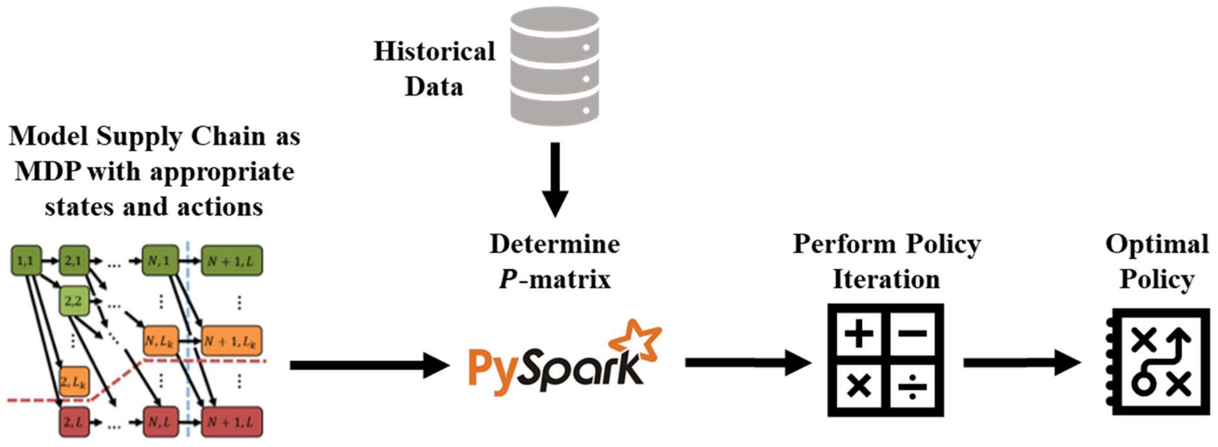


Figure 1.3 Integration of Big Data methods for determining the supply chain's dynamics

1.1.3. Investigate approximation methods like DRL for optimal control

This objective of the last research task is to investigate the performance of approximation methods such as deep reinforcement learning for their application to problems in agri-food supply chains. These methods are not limited to discrete state spaces and do not require previous knowledge of the transition model or reward function. To demonstrate the potential for these approaches, they will be applied to a common problem of mixing and blending grain and will be compared using simulation studies. Although the general formulation of this binary integer program is well known, the real-world problem is not so simple. The goal for this objective is to evaluate these methods as the mix and blend problem becomes stochastic and continues to operate over a rolling time horizon. Also, the state space for this problem is continuous, and tradition forms of optimal control such as Policy Iteration or Q-learning require the problem to be discretized. Instead, by leveraging a neural network to observe our state, we intend to evaluate these states directly from their current continuous form.

This task requires the development of a few key pieces. The first is a simulation environment that considers a standard mix and blend problem and generates many instances of the problem. Extending the problem across a rolling time horizon also requires learning about the transitional behavior of product quality characteristics and consideration of future states that must also be optimized. The matrix of quality characteristics defines the product attributes contained within each bin, which are observable in the current state but not easily predicted for future states. Each attribute has limits or bounds for its acceptability in the final mixture. The final mixture contains exactly β bins worth of material. Note, this is a specific version of the multidimensional 0-1 knapsack problem (Freville, 2004). Based on the decision taken, some selection of β bins' materials are used, and all the rest are left for the next decision epoch. The materials used are consumed and removed from the system and replaced with new incoming materials. The ones that remain are left for fulfilling future orders, but they also can degrade over time and can become poor quality if left within the system for too long.

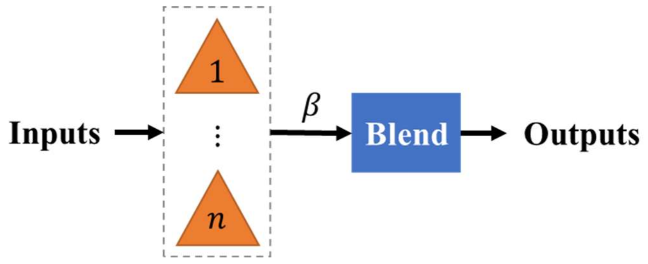


Figure 1.4 Process flow diagram of a standard mix and blend process in a single decision epoch

Figure 1.4 shows a process flow diagram of a standard mix and blend process. Since the end goal for this research objective is to consider a more complex version of this problem, one that is stochastic and optimized across a rolling time horizon, it is anticipated that exact algorithms in stochastic optimization will become incapable of solving this problem by expanding the formulation to account for the nearly infinite number of possible future sequences. We explore how to generalize this approach for all sequential integer linear programs and propose a novel idea called OR-Net. This is a neural network that achieves computational efficiency versus a fully connected (FC) network by focusing on the orthogonal relationships that exist within the single-stage integer linear program's coefficients. This greatly reduces the quantity of weights in the neural network that must be converged on during training. Ideas on how to implement this concept in PyTorch are discussed and one will then be selected for testing. Performances of different solution methods including: the OR-Net, its FC-Net counterpart, an LP solver, and other heuristics will be tested and reviewed for their accumulated reward values and running times.

Ultimately, the research in this thesis is strongly motivated by and focused on problems related to agri-food supply chains. However, we believe these approaches can be generalized to many other supply chain processes, and these techniques may be enhanced or applied to other sequential and stochastic optimization problems.

CHAPTER 2. LITERATURE REVIEW & BACKGROUND

Chapter 2 reviews the relevant literature and contains background sections that cover the basics of important concepts that will be prevalent throughout the following chapters. The notation introduced in this chapter will be used continually throughout the rest of the remaining chapters.

There are five sections in Chapter 2. Section 2.1 covers modeling approaches in agri-food supply chains. Section 2.2 covers data mining in agri-food supply chains including some background information on a few powerful Big Data techniques. Section 2.3 provides an in-depth review of Markov decision processes. Section 2.4 provides details of methods for solving dynamic programming problems such as MDPs, detailing the different classes of methods. Section 2.5 concludes the literature review and background section by covering the ideas behind deep reinforcement learning and how they are actively being researched for their application to problems in optimization and optimal control today.

2.1. Modeling of Agri-Food Supply Chains

Agri-food supply chain problems can be grouped by many methods such as planning horizon, solution and modeling methods, and perishability of the products considered (Ahumada & Villalobos, 2009). The scope of planning horizons can vary from strategic, to tactical, to operational level decisions. Solution and modeling methods can be broken into two major categories, namely deterministic and stochastic. Each of these can include problems that are either dynamic or static. Some also make a distinction of the perishability of the products. In the following paragraphs, we briefly discuss some of these key studies and describe the direction of our research.

The need for holistic modeling frameworks in food supply chains has been recognized, and decisions such as technology investment and capacity planning (Georgiadis, Vlachos, & Iakovou, 2005) have been investigated at a strategic level. Other works have focused specifically on food distribution approaches (Akkerman, Farahani, & Grunow, 2010), but most of these tools however are limited in terms of their generality or scope. The desire to consider more than one stage of the supply chain has been well described. Some have captured the complexity required to consider both production and logistic planning decisions for whole families of products (Kopanos, Puigjaner, & Georgiadis, 2012). Similarly, tactical and operational planning tools have offered more generalization of common farm level decisions. Examples of these include decisions for production and distribution of fresh produce (Ahumada & Villalobos, 2011) and harvest and distribution of perishable agricultural products (Ahumada & Villalobos, 2011). These models, however, do not capture decisions in later possible processing stages of the supply chain.

Gigler et al. (2002) developed a generalized dynamic programming model that considered optimization across all steps of the supply chain. Optimal routes were defined as minimizing total integral costs. This was implemented using a brute force search on an example considering willow biomass as fuel for an energy plant. However, each of these aforementioned studies took a common deterministic approach. Borodin et al. stated, "An agricultural production supply chain is moreover a process characterized by dynamic linkages and evolution often taking place in a stochastic operating environment." (Borodin, Bourtembourg, Hnaien, & Labadie, 2016). Other works have taken a stochastic modeling approach but were limited in either planning horizon, scope (amount of the supply chain considered), or generality and flexibility to be applied to any food supply chain (fresh, perishable, multiple processing stages). For example, some investigated beef herd management decisions (Stygar, Kristensen, & Makulska, 2014). Others explored the wheat quality control problem but focused on the harvest scheduling decisions (Borodin, Bourtembourg, Hnaien, & Labadie, 2015). Some considered the uncertainties introduced by weather and demand fluctuations and utilized two-stage stochastic programming (Ahumada, Villalobos, & Mason, 2012), but these were restricted to fresh products. These studies did not consider randomness introduced by the reliability of internal processes, and they did not see further into the supply chain. Many longer-term planning tools have been developed but were only designed to address specific questions, such as considering the effect of uncertain precipitation on production models to evaluate the investment in irrigation systems (Heumesser, Fuss, Szolgayová, Strauss, & Schmid, 2012), or developing water allocation and irrigation schedules (Sumanatra & Ramirez, 1997).

Table 2.1 Summary of the most relevant agri-food supply chain modeling related literature reviewed, and the opportunity explored in this thesis.

Supply Chain Steps Considered	Decisions Considered	Deterministic Optimization	Stochastic Optimization
Specific	Specific	Ahumada & Villalobos, 2011	Sumanatra & Ramirez, 1997 Heumesser et al., 2012
	Flexible	Kopanos et al., 2012	Ahumada et al., 2012 Borodin et al., 2015
All or Flexible	Specific	Ahumada & Villalobos, 2011	Georgiadis et al., 2005
	Flexible	Gigler et al., 2002	▨

To the best of our knowledge, there are no existing optimization models that capture the desired generality and flexibility to be applied to most any food supply chain problem. This includes the planning horizon or scope along with the consideration of the wide variety of decisions and uncertainties present in food supply chains. See Table 2.1 for a summary of the agri-food supply chain modeling related literature that is most relevant to the opportunity that is covered in this thesis. With the integration of data mining techniques, we can build and solve these more complex models that are driven by data.

2.2. Data Mining in Agri-Food Supply Chains

The desire for integration of modern data analytics into food supply chains did not go unnoticed especially in the areas of temperature sensitive food products (Olsson, 2004). Recent studies have even introduced technological concepts such as the Internet of Things for real-time tracking and tracing of prepackaged food (Li, Liu, Liu, Lai, & Xu, 2017) and improved multi-level supply chain coordination (Yan, Wu, & Zhang, 2017). Another team (Gaukler, Ketzenberg, & Salin, 2017) used radio-frequency identification and sensor technology to dynamically establish expiration dates. Others recognize that trustworthiness in agri-food supply chains will be a future concern and that Big Data architectures can help (Tao, et al., 2018). These ideas were simply not possible to implement a decade ago.

In the following subsections, a few common Big Data techniques will be reviewed as they will be important in the chapters that follow.

2.2.1. Association Rule Mining

Association rule mining is a method in machine learning that aims to discover relationships between variables of interest. It does this by exploring data and uncovering strong rules between these variables based on the frequency of their common existence in a set of items, called a transaction. These strong rules, called association rules (AR), are directional and can be used to discover the likelihood that one item will exist in a random transaction given another item of interest is in the transaction (Agrawal, Imielinski, & Swami, 1993). These rules are commonly extracted from point-of-sale systems, such as those in grocery stores, supermarkets, and retail.

For example, the rule for a situation when a consumer who purchases item W will also buy item U is given by the rule $(W \Rightarrow U)$. The strength of this rule would indicate the likelihood that if a customer buys W they also buy U, and the insights gained from these rules could be used to help improve marketing campaigns or store layout decisions, such as the placement of a premium brand of salsa next to a popular bag of tortilla chips.

In this example, W and U belong to the set of all items I that are represented in the set of transactions J. We use support and confidence to test the performance of the association rules. It is also common to only consider rules that meet a certain threshold of significance or confidence. We define confidence in this rule using the following elements.

Transactions: The dataset is defined as a set of transactions J where a random transaction $j \in J$ is a set of items that were purchased together.

Items: An item of interest W belongs to the total set of items I represented in all our data of transactions such that $W \in I$ and $j \subseteq I$.

Rules: A given rule is defined as the directional relationship (implication) from item or itemset W to the item U as $Rule = (W \Rightarrow U)$.

Support: The support of an item or itemset W is defined as the expectation that a given transaction j will include the item W and is determined by its frequency of occurrence in the total set of transactions J versus the number of transactions. The resulting value is the probability that item W will be included in a randomly selected transaction j, see Equation (2.1).

$$\text{supp}(\{W\}) = \frac{|\{W\} \in j \ \forall j \in J|}{|J|} = \text{Prob}(\{W\} \in j) \tag{2.1}$$

Confidence: A rule's confidence is defined as the expectation of whether a rule will be found to be true, given how often the rule was found to be true in the dataset. This is determined in Equation (2.2) by comparing the support of the item set including items from both $\{W\} \cup \{U\} = \{W, U\}$ versus the support of the item set only containing $\{W\}$.

$$\text{conf}(Rule) = \frac{\text{supp}(\{W\} \cup \{U\})}{\text{supp}(\{W\})} = \frac{\text{supp}(\{W, U\})}{\text{supp}(\{W\})} \tag{2.2}$$

Therefore, the likelihood that someone who buys item W will also buy item U can be described as the confidence of $Rule = (W \Rightarrow U)$, which is the ratio of the dataset's support for both W and U versus its support for just W.

Associate rules are used in many other applications today ranging from biology to understanding internet traffic between sites. Many tools have been created help mine these rules such as the arules package in R which includes implementations of the popular algorithms Apriori and Eclat (Hahsler, Grun, Hornik, & Buchta, 2005). PySpark is Big Data tool that offers parallel implementation of another algorithm that mines for these frequent patterns called FP-Growth.

2.2.2. FP-Growth Algorithm

The FP-Growth algorithm (Han, Pei, & Yin, 2000) computes a list of frequent items sorted by frequency and then compresses this data into a FP-tree. This tree is then mined recursively by the algorithm to identify items whose support passes a given threshold. Nodes on the FP-tree represent items and paths represent transactions or itemsets that share a common prefix. Subsequent patterns can be mined quickly after the tree has been formed, but the bottleneck of FP-Growth is that the potential combinatorial number of candidate patterns is not reduced despite the compact representation of these candidates in the FP-tree. Many efforts have been made to improve FP-Growth. Spark's implementation uses a parallelized version of the FP-Growth algorithm (Anonymous, n.d.) called PFP (Li, Wang, Zhang, Zhang, & Chang, 2008). A short description of the steps in PFP are as follows.

Step 1. *Sharding*

Divides data base into successive parts (shards) and stores on different computers.

Step 2. *Parallel Counting*

Counts the support of all items that appear in database using MapReduce on each shard and stores results in the F-list.

Step 3. *Grouping Items*

Divides the items on the F-list into groups with unique group-ids in the G-list.

Step 4. *Parallel FP-Growth*

Performs a MapReduce pass where the *Mapper* stage generates group-dependent transactions, and the *Reducer* stage runs the FP-Growth algorithm on group-dependent shards.

Step 5. *Aggregating*

Aggregates the results from step 4 into the final result.

This concludes the background section and literature review covering data mining.

2.3. Markov Decision Processes

Agri-food supply chains can be descried by dynamic linkages of sequential decisions that are subject to multiple sources of uncertainty. These include temperature, humidity, technology, storage conditions, etc., that can affect the outcomes and therefore impact the optimal control decisions in these supply chains. Given the level of generality and flexibility desired, this suggests the use of a Markov decision process (MDP). MDPs are discrete stochastic process that provide a mathematical framework for modeling decision making in situations where outcomes are probabilistic (Puterman, 2005). They are comprised of the following elements.

Decision Epochs: It is assumed that decisions occur at every change in the state.

State Space, S: Let $s \in S$, be the state of the system that is defined as a vector comprising of the relevant information needed to make decisions and quantify outcomes. An individual state can be theoretically described as a location within our state-space S. This assumes that the process is not fully history dependent.

Action Space, X: Let $\mathbf{x}_s \in X$, be the actions selected at state s. This can be represented as a vector of actions $\mathbf{x}_s = [x_s^1, x_s^2, \ldots]$. These are selected from the set of available actions X. This known as the action space, which includes all the potential unique actions at any given state.

Transition Model, P: $p(s'|s, \mathbf{x}_s)$ is defined as the transition probability distribution from the state s given action $\mathbf{x}_s$ to a new state s'. The transition probability matrix (or transition model) includes all the transition probabilities within the state space S. This is Markovian, meaning that only the most recent state s and action $\mathbf{x}_s$ are needed to know the distribution of s'.

Rewards: A reward $R(s, \mathbf{x}_s)$ is the expected net benefit received for being in state s and taking action $\mathbf{x}_s$. The total set of rewards can be defined using a function or a matrix.

Policy: A policy π is defined as a mapping of actions to take given the state. This is represented as $\pi = \{\mathbf{x}_{(s)} \text{ for all } s \in S\}$.

Value Function: The value function for a policy π is V_π, which can be described recursively as in Equation (2.3) using Bellman's equations, where $\pi(s)$ is the action to take as determined by the policy π and γ is the discount factor that prevents the value function from going to infinity. The optimal policy π^* is defined as the decision policy which maximizes the value function, see Equation (2.4).

$$V_\pi(s) = R\big(s, \pi(s)\big) + \gamma \sum_{s' \in S} p\big(s'|s, \pi(s)\big) V_\pi(s') \tag{2.3}$$

$$\pi^* = \arg\max_{\pi'}(V_{\pi'}) \tag{2.4}$$

2.4. Optimal Control Methods

The goal of dynamic programming (including MDPs) is to find the optimal policy π^* that maximizes this value function $V_\pi(s)$ at every state $s \in S$. This section describes different approaches of finding optimal policies for MDPs: model-based methods and model-free methods which both have their own strengths and weaknesses.

2.4.1. Model Based Methods

Model based methods require knowledge or an assumption of the transition model which identifies the possible next states the model is likely to reach after taking the decision. Under this knowledge, exact methods exist that guarantee optimality of the proposed optimal policy. One of the most popular ways to solve for the optimal policy is by implementing the Policy Iteration algorithm (Bellman, 1955).

Policy Iteration Algorithm: This algorithm works under a simple premise. Firstly, select an initial policy and then determine the value of each state under the current policy. Next, consider whether the value could be improved by selecting a different action. If it can, change the policy at that state to take this new action. This step-by-step process gradually improves the performance of the policy and when no improvements are possible, then the policy is guaranteed to be optimal.

The general algorithm for Policy Iteration is:

Step 1. Initialization

Set iteration counter to $i = 0$, and select an initial policy $\pi^i = \pi^0$.

Step 2. Policy Evaluation

Given a policy π^j, compute the value function V_π^j by solving the following set of linear equations (2.5).

$$V_\pi^i(s) = R\left(s, \pi^i(s)\right) + \gamma \sum_{s' \in S} p\left(s' | s, \pi^i(s)\right) V_\pi^i(s') \tag{2.5}$$

Step 3. Policy Improvement

Use the computed value function V_π^i to find an improved policy π^{i+1}.

$$\pi^{i+1}(s) = \arg\max_{\mathbf{x} \in X} \left\{ R(s, \mathbf{x}) + \gamma \sum_{s' \in S} p(s' | s, \mathbf{x}) V_\pi^i(s') \right\} \tag{2.6}$$

Step 4. Optimal Policy Check

If $\pi^{i+1} = \pi^i$, then $\pi^* = \pi^{i+1}$ is the optimal policy. Otherwise, increase $i = i + 1$ and repeat from Step 2.

Policy iteration is popular for infinite horizon problems because of how easily the value of a policy can be determined. Another popular model-based solution method is accomplished by formulating the MDP problem as a Linear Program (Manne, 1960).

MDP Linear Programming Formulation: This technique has become popular because of major improvements in the area of commercial LP solvers. See below for the formulation of an MDP problem as a LP. This requires knowledge (or an assumption) of the initial state or its probability distribution μ_0 over S, with $\mu_0 > 0$ for all $s \in S$.

$$\begin{aligned} &\max\ \mu_0(s)V(s) \\ &\text{s.t.}\quad V(s) \geq R(s, \mathbf{x}) + \gamma \sum_{s' \in S} p(s' | s, a) V(s') \quad \forall\, s \in S, \mathbf{x} \in X \end{aligned} \tag{2.7}$$

The best specialized MDP algorithms like modified policy iteration are typically faster than general LP algorithms. However, the LP formulation has been the foundation for work in approximate large-scale MDP solutions (Farias & Roy, 2003).

2.4.2. Model Free Methods - Reinforcement Learning

There are several challenges with solving large-scale MDPs, especially in practice. The first of these is that the information desired to be captured by the states and actions will grow (often exponentially) based on the problem's complexity. This is referred to as the "Curse of Dimensionality" and is why dynamic programming is sometimes avoided for large problems (Powell, 2011). The next major challenge is that the model may not be well understood. This can include knowledge about the transition model and the reward function, as knowledge of both are required to find an optimal policy using model-based methods.

In this section, model-free methods will be discussed as these techniques try to address these concerns. These methods come from an area of research that has many names including Approximate Dynamic Programming (ADP) (Powell, 2011), Reinforcement Learning (RL) (Sutton & Barto, 2018), and Neuro-Dynamic Programming (Bertsekas & Tsitsiklis, 1996). These are all very similar fields and mostly differ in the specific applications they are interested in addressing. For example, RL is more common in the Artificial Intelligence community due to its applications in robotics, and ADP is becoming more popular in the Operations Research domain due to its focus on large state spaces that can sometimes be well captured with transition functions instead of matrices. The most common framework for model-free solutions is using a learning agent that interacts with its environment through a feedback loop and updates its decision policies through experiences (i.e., data), see Figure 2.1. The most popular model-free technique in these fields was introduced by Watkins and it is called Q-learning (Watkins & Dayan, 1992).

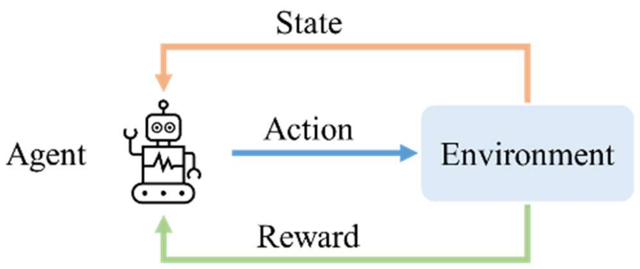


Figure 2.1 Agent interacting with its environment through a feedback loop

Q-learning: This well-known method is named after the values it aims to estimate. Q-values are estimates of the value of being in state s and taking the action $\mathbf{x}$, see Equation (2.8). This can be rewritten entirely in terms of the Q-value functions as shown in Equation (2.9).

$$Q^*(s,\mathbf{x}) = R(s,\mathbf{x}) + \gamma \sum_{s' \in S} p(s'|s,\mathbf{x}) V^*(s) \tag{2.8}$$

$$Q^*(s,\mathbf{x}) = R(s,\mathbf{x}) + \gamma \sum_{s' \in S} p(s'|s,\mathbf{x}) \max_{\mathbf{x}'} Q^*(s',\mathbf{x}') \tag{2.9}$$

The properties of the MDP solution that are of interest can also be described by the Q-values. These include the optimal value function (2.10) and the optimal policy (2.11).

$$V^*(s) = \max_{\mathbf{x}} Q^*(s,\mathbf{x}) \tag{2.10}$$

$$\pi^*(s) = \arg \underset{\mathbf{x}}{max}\, Q^*(s,\mathbf{x}) \tag{2.11}$$

How this method becomes model free is when knowledge of the outcome state s' is also known. Then these values can be approximated using equation (2.12). Further, if the reward function $R(s,\mathbf{x})$ is not well known, this can also be approximated with enough data using equation (2.13) if the reward received r after each experience is known.

$$Q^*(s,\mathbf{x}) \approx R(s,\mathbf{x}) + \gamma \max_{\mathbf{x}'} Q^*(s',\mathbf{x}') \tag{2.12}$$

$$Q^*(s,\mathbf{x}) \approx r + \gamma \max_{\mathbf{x}'} Q^*(s',\mathbf{x}) \tag{2.13}$$

The procedure for implementing Q-learning is:

Step 1. Initialization

Set iteration counter to $i = 0$, and select initial Q-value estimates $\hat{Q}_i = \hat{Q}_0$.

Step 2. Action Selection

Given the input state s_i, select an action $\mathbf{x}_i$.

Step 3. Update Q-values

Given a new data element (experience) $d_i = (s_i, \mathbf{x}_i, r_i, s_{i+1})$ and existing Q-values $\hat{Q}_i$, update the Q-value estimate for s_i and $\mathbf{x}_i$ using Equation (2.14). The learning rate η is a step-size in the direction of the newest available information, where $0 < \eta \leq 1$.

$$\hat{Q}_{i+1}(s_i, \mathbf{x}_i) = (1 - \eta)\hat{Q}_i(s_i, \mathbf{x}_i) + \eta \left(r_i + \gamma \max_{\mathbf{x}'} \hat{Q}_i(s_{i+1}, \mathbf{x}') \right) \tag{2.14}$$

Step 4. Convergence Check

If convergence criteria have be met, set $Q^* = \hat{Q}_{i+1}$. Otherwise, increase counter $i = i + 1$, go to Step 2 and repeat.

The strengths of Q-learning lies with the ease of computation, and when given an infinite exploration of all state-action pairs $(s, \mathbf{x}) \in (S, X)$, due to Banach fixed-point theorem, this algorithm will converge on the true Q-values Q^* and find the optimal policy π^* (de Farias & Van Roy, 2000). In the simulation space, the greatest challenge of model-free learning is the *exploration vs exploitation* problem, where the agent must decide if it is more important to choose a new action (possibly at random) or to use its knowledge and select optimally. Therefore, step 1 is critical for the time to convergence in training steps and the stability of performance in applications of online learning with changing environments.

A major limitation of traditional tabular Q-learning is that both the state and action spaces are required to be discrete. Another limitation of Q-learning is that it cannot infer the Q-values of states or actions it has never visited from other experiences. Newer methods in the area of function approximation allow for these two assumptions to be dropped. One of the most profound and active areas of research in this domain is called Deep Reinforcement Learning (DRL).

2.5. Deep Reinforcement Learning for Optimization Problems

Combinatorial optimization problems (COPs) are mathematical optimization problems that involve finding the optimal object from a finite set of objects. Many consider both integer programs (IPs) and COPs to be branches from the field of discrete optimization, so they are very closely related in the research literature. In general, for large instances of these problems exhaustive search is not a reasonable option, which is why this research field for finding efficient solutions to these problems is consider highly valuable. Much work has been done to improve the solution time within the IP framework for exact solution methods. This includes algorithms and methods such as: Branch & Bound (Land & Doig, 1960), Branch & Cut (Crowder, Johnson, & Padberg, 1983), and Chvátal-Gomory Cutting Planes (Gomory, 1658; Chvátal, 1973). These are important breakthroughs that when paired with heuristics, help drive modern day commercial solvers like CPLEX and Gurobi. However, the following discussion will focus on work in recent years that has come more specifically from the fields of DRL.

DRL became instantly famous when DeepMind researchers successfully trained agents to play games in Atari at levels comparable to professional video games testers (Mnih, et al., 2015). DeepMind eventually developed AlphaGo to beat world champion Go player (Silver, et al., 2016). This knowledge was then generalized further for AlphaZero (Silver, et al., 2018), which mastered the games of Go, chess, and Shogi starting from random play and given no domain knowledge except the game rules.

The most popular method in DRL is called Deep Q-networks (DQN). The basic idea behind DQN is designing a neural network that performs function approximation to estimate the Q-values like Q-learning (Watkins & Dayan, 1992). The DQN is notated by Q_θ where θ describes the weights of the arcs in the network. Instead of updating Q-values, the weights θ of the network are updated. The input states and actions can be any combination of discrete or continuous as they are simply input features to the neural network. DQNs are relatively new in their development and commonly experience issues. Some of these issues are because they are no longer converging to the Q-values themselves (a fixed-point in standard Q-learning), but rather a projection of the Q-values. This leads to problems with stability and convergence. Some of the modifications to DQN that have shown good results for dealing with this and other issues include experience replay, target networks, Double Deep Q-networks, and Dueling Q-networks. An experience replay buffer allows agents to remember and reuse experiences from their past (Lin, 1992), making more efficient use of the data and avoiding biases to recent data. Prioritized experience replay improved on this idea by identifying important transitions and replaying them more often, so the agent learns more efficiently (Schaul, Quan, Antonoglou, & Silver, 2016). To deal with the issue of what is called moving Q-targets, DeepMind researchers developed target networks (or fixed Q-targets) (Mnih, et al., 2015). The target network is a copy of the main DQN, but it updates less frequently to prevent the DQN from chasing a moving target (itself).

Another issue that DQN researchers found was that the Q-values were commonly overestimated, which is called maximization bias. It was not well understood if this had much of an effect on the actual performance of the DQN until Double DQN reduced overestimations of the

Q-values (Hasselt, Guez, & Silver, 2016). This improved performance in games and simultaneously solved the moving Q-target issue by creating two independent DQN, one for selecting the next action and another to evaluate that action. The next improvement was found from Dueling Q-networks (Wang, et al., 2016) which decomposed the Q-value functions into two parts: state value functions $V(s)$ and advantage value functions $\mathcal{A}(s, \mathbf{x})$. This idea was based on how deep neural networks work in their hidden layers. They identify features that contribute to understanding something more complex, such as the presence of vertical lines in object detection. The researchers for Dueling Q-networks believed that the features for identifying the value of states and for evaluating actions may not be the same, so they split the network for each separate function and then combined them at the end. Despite these issues, DQN have shown great promise and often exceed the performance of domain specific experts (both human and artificial). Some of the unique strengths of deep learning as a field include complex function approximation and transferable learning of features. These ideas are actively being researched for their applications in DRL and may lead to new advances in optimal stochastic control.

Due to the significant successes that DRL has experienced in recent years, many have started to wonder if DRL could be used to solve other more impactful problems in society. Some have even started to apply these tools to attempt to solve COPs. In 2015, Vinyals et al. created pointer networks (Vinyals, Fortunato, & Jaitly, 2015), which are neural networks that learn conditional probabilities to connect sequences. They showed this could be used to help solve different COPs like convex hulls and traveling salesman problems (TSP). As COPs and IPs are often modeled as graphs which have common structures, others such as Dai et al. (Dai, Khalil, Zhang, Dilkina, & Song, 2018) levered reinforcement learning with graph embedding to try and learn the graph algorithms through exploiting these common structures. Their approach called stucture2vec trained a graph neural network using deep Q-learning and demonstrated the scalability of these tools. Next, researchers began to apply transformer architectures like those that had been proven highly successful in problems with sequenced data like those in natural language processing. One example introduced graph attention networks (Veličković, et al., 2018) as NN architectures that leverage stacked attention layers to avoid costly matrix operations or requiring knowledge of the graph structure upfront. Another paper (Kool, Hoof, & Welling, 2019) specifically addressed routing problems like TSP and others by training an attention network model using REINFORCE (Williams, 1992) to deliver near optimal results. More recently, Manchanda et al. (Manchanda, et al., 2020) shifted their focus on scalable approaches to finding heuristics over large graphs via DRL. They train a graph convolutional network using their proposed framework GCOMB that tackled large COPs much more efficiently and with similar quality as other state-of-the-art methods. GCOMB showed the value in finding lightweight architectures to solve these graph problems that often suffer with their extremely complex nature and scale.

This concludes the literature review and background sections contained in Chapter 2.

CHAPTER 3. STOCHASTIC MODEL OF AGRI-FOOD SUPPLY CHAINS

Chapter 3 is based on two manuscripts. The first is “Optimal Control in Dynamic Food Supply Chain under Quality Constraints” published in the Proceedings of the 2019 IISE Annual Conference (Kappelman & Sinha, 2019). This chapter also includes the modeling related excerpts based on the manuscript “Optimal Control in Dynamic Food Supply Chains using Big Data” published in Computers & Operations Research (Kappelman & Sinha, 2021).

3.1. Introduction

Performance in supply chains can be difficult to predict and optimize. This is especially true in food supply chains because of some of their unique characteristics, which can lead to large inefficiencies and waste. Approximately 25% of produced food in the U.S. is lost or wasted in the supply chain before reaching the consumer (Dou, et al., 2016). Food products are time sensitive and perishable, and they are often sourced from remote or difficult to reach locations. Their production is affected by environmental factors and is subject to strict quality demands and regulations. Also, no two products’ or organizations’ supply chains are the same (Georgiadis, Vlachos, & Iakovou, 2005). The decisions that deal with these factors and influence product quality are vast. These include the selection of suppliers and the settings for their process parameters at every step of the supply chain.

Many works argue the complexity of the model and then use approximate methods to analyze the performance of their systems. In contrast, the main contribution of this chapter is to model the food supply chain dynamically and enable data-driven decision making. The major advantage of data-driven decision models in supply chains is their ability to statistically predict outcome probabilities based on input decisions and information states, which helps to capture the uncertainty introduced by different sources. It also enables the decision maker to understand how these outcomes can contribute to product quality across the entire supply chain. This is accomplished by integrating data mining techniques with an MDP model to determine the optimal suppliers and their process control parameters at each level of the supply chain. With the consideration of potentially perishable products, we maximize the quality of the final product. Further, numerically we explore the impact of costs and quality constraints on the optimal decisions.

Chapter 3 is comprised of four sections. Section 3.2 describes the proposed food supply chain and mathematical model. Section 3.3 covers some experiment results. Section 3.4 concludes our work and highlights the contributions of this chapter.

3.2. Generalized Stochastic MDP Model for Agri-Food Supply Chains

3.2.1. Background and description of our model

We consider a dynamic food supply chain and model it as a process consisting of a series of sequential steps. Common steps for any problem could include producer, one or more processors, wholesaler, and retailer. Steps for an operational problem might describe the individual stages of development within the decision maker's control. From herein we shall follow more specifically the scope an entire supply chain and described above, but please note that this modeling approach is just as applicable to operational level problems. Each step could have multiple options for the supplier selected, who each implement the system parameters to produce the end-product (which could consist of fresh produce, livestock, grain, etc.). The resulting quality level at every step for this product is stochastic. We also consider that the quality level at each supply chain step has a minimum accepted level for the materials to be useful in the final products.

Material is rejected if its quality level does not meet the minimum accepted level. A variety of factors can be used to determine the quality level. These include cosmetic conditions, consumer preference, nutritional value, and factors related to the needs of the following process steps such as particle size or moisture content as they may relate to manufacturing and processing related concerns. Figure 3.1 shows some of the variety of decisions to be made that could affect the product quality and supply chain performance at each step. These decision options could include the utilization of different raw materials, innovative technologies, processing equipment, transportation modes, and storage options available that are often unique to each specific step.

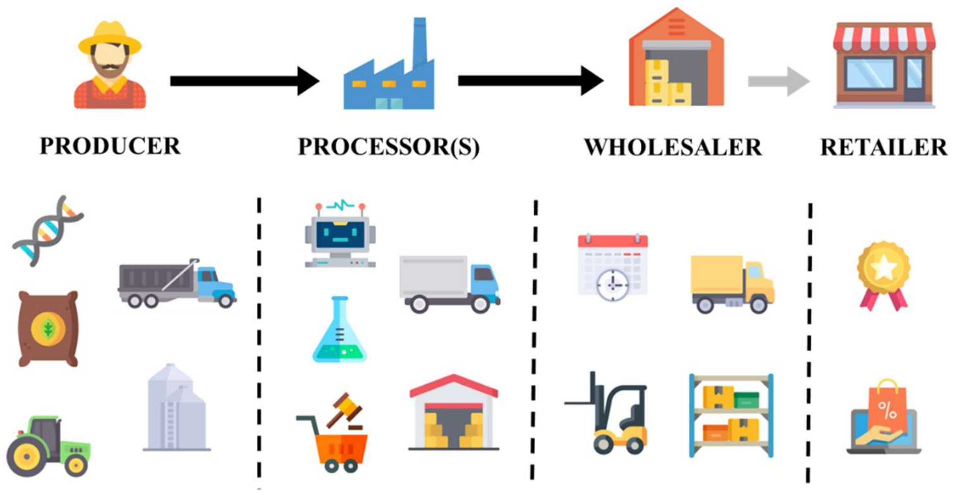


Figure 3.1 Example of steps and variety of decisions involved in agri-food supply chains.

At the production step the decision maker could consider the use of different biotechnology, seeds, or chemical applications; and whether to use manual labor for harvest or specialized equipment. Also, the timing of the harvest could impact the choice to store the material for a time-period or ship immediately. At the processing step(s), different level of process controls and automation might be available such as online monitoring in place of traditional sampling techniques. The process might also operate in batch or continuous modes and key steps accomplished via either chemical or mechanical means. At the wholesaler operations, the storage conditions might vary by supplier such as temperature and humidity control. There may also be different means of handling particularly sensitive items like fresh produce. Lastly, the retailer operations might also have an impact like delivery options, specific labeling, or storage and display options (e.g., refrigerated or room temp for products that can be subjected to both). Any one of these decisions can affect the supply chain's profitably and the end-product's perceived quality.

In this supply chain, we track quality because this property highly influences when the materials in the current steps are available for use in different potential final products and the final product's acceptability by the consumer. These considerations make quality the ideal tracking mechanism for optimizing the supply chains' profit and marketability while reducing food loss (Nagurney, Besik, & Yu, 2018).

3.2.2. Generalized MDP mathematical model

In this section we develop a generalized Markov decision process (MDP) model to describe the step-by-step sequential decision-making processes within food supply chains. MDPs are discrete stochastic processes that provide a mathematical framework for modeling decision making problems in situations where outcomes are probabilistic. This means that a MDP is an ideal mechanism for capturing the uncertainties and vast array of complex situations and actions that decision makers in agri-food supply chains face.

We consider a food supply chain with N steps. For each step $k \in K, K = 1, \dots, N,$ let $y_k \in Y$ denote the quality and the generalized total set of all quality levels, $Y = \{1,2, \dots, L_k, \dots, L\}$ where L_k represents the minimum quality level threshold that needs to be satisfied at the supply chain step k for a product to be moved to step $k + 1$. Note that the quality threshold, L_k, is dependent on the step because the quality requirements may be different depending on the product's current step within the supply chain. Products with quality levels y_k less than L_k are rejected. Note that in our model, the quality level of products at each level is stochastic with the aim to determine the optimal mix of suppliers and their parameters to achieve the final product at a highest quality level and minimize the costs. The MDP model that describes our supply chain contains the following elements (Puterman, 2005).

Decision epoch: We assume that the decisions are taken in continuous time and occurs at every change in the state i.e., at each quality level and supply chain step $k, k \in K$.

State Space, S: Let $s = (k, y_k), s \in S$, be the state of the system that is defined as the tuple comprising of the supply chain step and quality level. This assumes that the process is not fully history dependent. Figure 3.2 visualizes this two-dimensional state space.

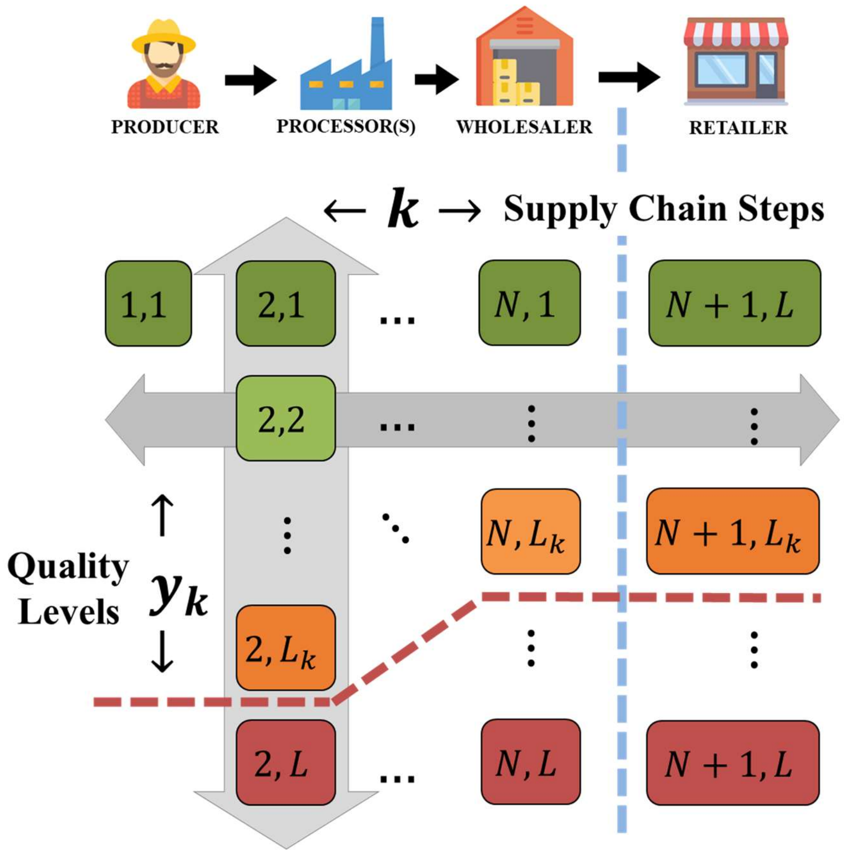
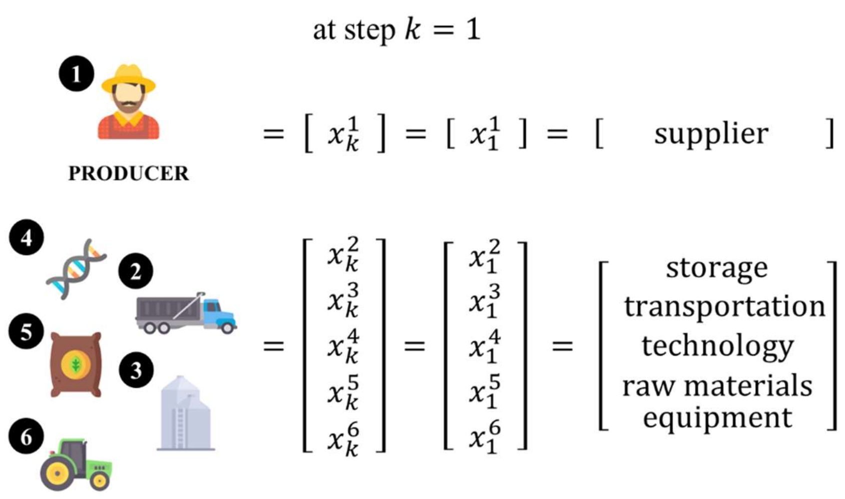


Figure 3.2 State space diagram of our supply chain.

Action Space, X: We represent actions selected in step k as a vector of actions $\mathbf{x}_k = [x_k^1, x_k^2, \dots, x_k^n]$. The first element x_k^1 is the supplier selected at step k. The remaining elements $x_k^2, \dots, x_k^n$ are the remaining process parameters that were selected. We will refer to all x_k^i simply as process parameters (including the specific supplier selected). The vector length n is the maximum number of parameters that need to be selected in any given step. These are selected from the set of available actions X_k such that $\mathbf{x}_k \in X_k, X_k = \{1, \dots, h\}$ for all i and k. From this information, we can determine that our action space, which includes all the potential unique actions at any given step is size h^n. This means that our action space will grow exponentially as we try to consider more information and quantify the effect of each possible action.



at step $k = 1$
1
PRODUCER
$= [\ x_k^1\ ] = [\ x_1^1\ ] = [\ \text{supplier}\ ]$
4
2
5
3
6
$= \begin{bmatrix} x_k^2 \\ x_k^3 \\ x_k^4 \\ x_k^5 \\ x_k^6 \end{bmatrix} = \begin{bmatrix} x_1^2 \\ x_1^3 \\ x_1^4 \\ x_1^5 \\ x_1^6 \end{bmatrix} = \begin{bmatrix} \text{storage} \\ \text{transportation} \\ \text{technology} \\ \text{raw materials} \\ \text{equipment} \end{bmatrix}$

Figure 3.3 Action vector $\mathbf{x}$ description example for supply chain step $k = 1$.

Note that these actions are problem, product, and step specific. Our model considers supplier selection as a primary decision because it is believed to be the most controllable by the decision maker. Many process parameters related decisions may be unique to both supplier and step. Lastly, we recognize that storage, transportation, and handling options are actions that need to be selected at potentially all steps.

Transition probabilities: $p(s'|s, \mathbf{x}_k)$ is defined as the transition probability distribution from the state $s = (k, y_k)$ given action $\mathbf{x}_k$ to state $s' = (k+1, y_{k+1})$. We notate our transition probability matrix as P, which includes all the transition probabilities within the state space S. Figure 3.4 presents the simplified transition diagram of our supply chain.

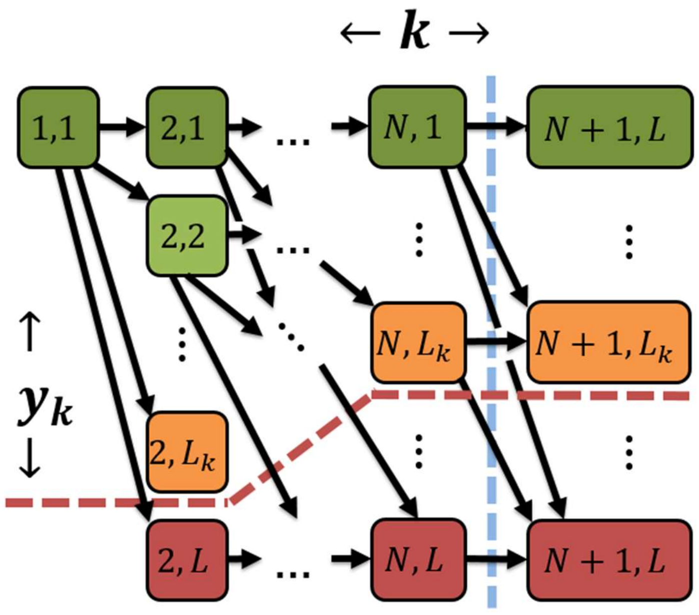


Figure 3.4 Supply chain states $s = (k, y)$ connected by transition probabilities $p(s'|s, \mathbf{x})$.

Since the model must meet specification at each step, product at any quality level worse than L_k (where $y_k > L_k$) would be rejected. We model thresholds on quality levels by using our worst quality state L as an absorbing state for all rejected products. The transition probabilities for our model are derived from available data using a Big Data technique described in the next section.

Reward function: A reward $R(s, \mathbf{x}_k)$ is the expected net benefit received for being in state $s = (k, y_k)$ and choosing to take action $\mathbf{x}_k$. We will define reward using a function in our examples. The general form of this function is $R(s, \mathbf{x}_k) = B(s) - C(\mathbf{x}_k)$, where $B(s)$ is the benefit achieved (a function of the state) and $C(\mathbf{x}_k)$ is the cost to get there (a function of the action taken). These are both linear functions and their general forms can be seen in Equations (3.1) and (3.2).

$$B(s) = B(k, y_k) = b_0 + b_k y_k \tag{3.1}$$

$$C(\mathbf{x}_k) = c_0 + \mathbf{c}_k \mathbf{x}_k = c_0 + \sum_{i=1}^{n} c_k^i x_k^i \tag{3.2}$$

A policy is defined as a set of actions to take given all possible states, represented as $\pi = \{\mathbf{x}_{(s)} \text{ for all } s \in S\}$. Our value function for a policy π is V_π, which can be described recursively as in Equation (3.3) using Bellman's equations, where $\pi(s)$ is the action to take as determined by the policy π and γ is the discount factor. The optimal policy is determined by implementing a simple Policy Iteration algorithm (Bellman, 1955) to select an optimal decision policy π^* that maximizes our value function, see Equation (3.4).

$$V_\pi(s) = R\big(s, \pi(s)\big) + \gamma \sum_{s' \in S} p\big(s' | s, \pi(s)\big) V_\pi(s') \tag{3.3}$$

$$\pi^* = \text{argmax}_{\pi'}(V_{\pi'}) \tag{3.3}$$

3.2.3. Monotonicity in the Optimal Value Function

In this section we show monotonicity in the optimal valuation function $V_\pi^*(s)$ using Proposition 1. Proving monotonicity ensures faster convergence to the optimal policy π^* (Puterman, 2005) when implementing the Policy Iteration algorithm.

Proposition 1. *The optimal value function* $V_\pi^*(s)$ *in Equation* (3) *is non-decreasing in state* s, for all $s \in S$. To prove this proposition we must show the following conditions:

1. $R(s, \mathbf{x}_k)$ is non-decreasing in s for all $\mathbf{x} \in X$ and $k \in K$.
2. $q(j | s, \mathbf{x}_k) = \sum_{y'=1}^{j} p(s' = (k+1, y') | s, \mathbf{x}_k)$ is non-decreasing in s for all $j \in S$, $\mathbf{x} \in X$ and $k \in K$.
3. $R_N(s)$ is non-decreasing in s.

Conditions 1 and 3 are met because of the linear nature of the reward function $r(s,\mathbf{x}_k) = B(s) - C(\mathbf{x}_k)$ where $B(s)$ and $C(\mathbf{x}_k)$ are described in Equations (3.1) and (3.2). Next, we show Condition 2. With the assumption of perishability with our supply chain, the quality y' in step $k' = k + 1$ will always have at least the same values as in step k. We state that all products (data points) are initialized at state $s = (k, y_k) = (1,1)$. The supply chain step is then incremented such that $k' = k + 1$ and the quality level can only be maintained or depreciated such that $y' = y_{k+1} \geq y_k$ (higher values represents bad quality). This depreciation is assumed random but is limited based on the quality of prior actions, which we describe as the total investment in actions at a particular step $C(\mathbf{x}_k)$. We define a discrete probability function f, that describes the distribution of quality level outcomes at the next step y' based on the current quality level $y'_{\text{best}} = y_k$ and the worst possible level y'_{worst}. This worst possible level is non-increasing based on the current quality level and investment in actions. Therefore, $y' \sim f(y'_{\text{best}}, y'_{\text{worst}})$ can be reduced to a non-increasing function based on the current quality level and investment in actions $y_{k+1} \sim f(y_k, \mathbf{x}_k)$.

Suppose two different actions δ_1 and δ_2 are taken such that $C(\delta_1) \geq C(\delta_2)$. The more expensive action δ_1 will provide a more favorable distribution than the cheaper action δ_2 given a current state s. It is known that the expectation of outcomes $\mathbb{E}[f(y,\delta_1)] \leq \mathbb{E}[f(y,\delta_2)]$, where $\mathbb{E}[f(y,\delta_1)]$ is desirable to be close to 1. Therefore, the likelihood of achieving a certain desired threshold state $s' = j$ in the next step is at least as good for the higher costing action δ_1 than for δ_2 such that $q(j|s,\delta_1) \geq q(j|s_2,\delta_2)$.

Also, suppose two possible states $s_1 = (k,\mu_1)$ and $s_2 = (k,\mu_2)$ are taken at the same supply chain step k. We define $\mu_1 \leq \mu_2$ to indicate that product at s_1 has at least as good of a current quality level (closer to $y_k = 1$) as product at s_2. Since better current quality levels provide a more favorable distribution than worse quality levels for a given action on the output state, it can be seen that the expectation of outcomes $\mathbb{E}[f(\mu_1, \mathbf{x}_k)] \le \mathbb{E}[f(\mu_2, \mathbf{x}_k)]$ and also that $q(j|s_1, \mathbf{x}_k) \ge q(j|s_2, \mathbf{x}_k)$.

Therefore, we conclude this proof by showing all three conditions, and the optimal value function $V_\pi^*(s)$ is non-decreasing and monotone. ■

3.3. Experimental Results

This section presents the characteristics of the optimal policy using different scenarios. We assume a case study with three supply chain steps (1 = producer, 2 = processor, 3 = wholesaler) and four quality levels (1 = great, 2 = good, 3 = fair, 4 = poor). We consider that at each step there are three parameters (including supplier), each with three possible settings resulting in an action space of $3^3 = 27$ possible actions. Note that the action space grows exponentially with system parameters, including the number of suppliers.

3.3.1. Data generation

To ensure each action has a high possibility of being explored, 2700 original data points were generated, which will be maintained for each step. Each data point begins with a great initial quality level at step 1 meaning state $s = (k, y) = (1,1)$ for all original data points. Therefore, states $s = (1,2)$, $(1,3)$, and $(1,4)$ are not used.

We simulated our data points at each stage using a uniform random number generator and then determined our values from these numbers, which can be a good depiction of reality. For example, to determine the quality level for a data point, the previous quality level score y_k was compared to a new random number u and the maximum (worse) was taken, $y_{k+1} = \max\{u, y_k\}$. All unique data points were initialized at great quality ($y_1 = 1$).

3.3.2. Sensitivity analysis with respect to quality requirements

For case study, two reward functions were tested, and eight trials run for each. The two reward functions are described specifically in the header of Table 1, but there is no reward $R = 0$ when the product was rejected ($y_k > L_k$). Note that $k' = k + 1$ and $y' = y_{k+1}$ in the reward function definitions. A discount factor of $\gamma = 0.75$ was used for all runs. For each trial, a new data set was generated and then solved for the optimal policy using Policy Iteration at each potential L_k setting. Final policies were compared, and it was recorded how often the policy for a data set changed as the quality threshold went from fully unrestricted ($L_k = 4$) to more restrictive until only the best quality was accepted ($L_k = 1$). This was done by noting where a specific optimal policy element (action to take at a given state) was first suggested starting from the least restrictive trial run. These run results can be seen in Table 1.

Table 3.1 Run results indicating the percent of policy origination coming from which trial run

Reward function	**#1: $r = 25 + k' - 2y' - 0.1\sum x_k$**				**#2: $r = 25 + k' - 2(y' - L_k) - 0.1\sum x_k$**			
L_k setting for trial run	4	3	2	1	4	3	2	1
% from $L_k = 4$ policy	**100%**	67.5%	27.5%	10%	**100%**	77.5%	27.5%	5%
% from $L_k = 3$ policy		32.5%	0%	5%		22.5%	10%	0%
% from $L_k = 2$ policy			72.5%	10%			62.5%	12.5%
% from $L_k = 1$ policy				35%				42.5%
% N/A*				40%				40%

Note that in cases where a state started in the rejection zone, this was recorded as N/A. Other observations included that the default action (the cheapest) was selected for all states where the current quality was $y_k = 3$. This suggests that when the current quality level was poor enough and adjacent to the rejection zone, the tradeoffs in the reward function worked out in a way where there was not a sufficiently good action to take for the product to have a high expectation on average for avoiding from being rejected anyways. We specifically say "poor enough" because this was not always the case for situations where the current quality level was adjacent to L_k, although it did happen more frequently the further away from $y_k = 1$ the state being evaluated was.

Outcomes in Table 3.1 showed that once the system adjusted to reject anything worse than good ($L_k > 2$) the decision rules shifted from heavily favoring the original rules to now having a favoritism to the most recent rules. The in-between zones' rules became more and more scarce. Only the most dominant decision rules survived from the beginning, and their dominance was tested as the system become less and less flexible. This is an effect of the supply chain responding the quality constraints, which are causing it to now earn zero value from any previous middle ground earnings. We believe this is representative of a strategic shift from the flexible system's best weighted average strategy to a restricted system's more aspirational strategy that maximizes the chance for achieving only the top end results.

3.4. Conclusions

In this chapter, we expanded the generalized model for optimizing dynamic food supply chains to account for the stochastic behavior of the outcomes given decisions made and actions taken. This model considers the effect these individual decisions (including supplier selection) may have on the product's final quality. We model a basic food supply chain dynamically with the decisions and actions at each step as a Markov decision process. Sensitivity analysis was then performed to study the impact of threshold values and reward function on the optimal policy selection. This reveals a strategy shift for the optimal policy from selecting a best weighted average with lower restrictions towards becoming aspirational as the system became more restricted by the quality constraints.

As noted previously, the action space grows exponentially. Therefore, our final solution technique may need to be modified as larger problems that are more complex may not be able to converge to a final solution in an efficient timeframe using Policy Iteration. We propose to use Reinforcement Learning to solve these more complex and general problems.

CHAPTER 4. DETERMINING SUPPLY CHAIN DYNAMICS FROM DATA

In Chapter 3, we created a generalized stochastic model as an MDP for agri-food supply chains. Chapter 4 integrates data mining techniques to convert this MDP model into a data-driven model, and it is based on the data mining techniques introduced in two manuscripts. The first is "Optimal Control in Dynamic Food Supply Chain under Quality Constraints" published in the Proceedings of the 2019 IISE Annual Conference (Kappelman & Sinha, 2019) where Bayesian networks are integrated into our MDP model. Chapter 4 also includes a novel Big Data mining approach based on the manuscript "Optimal Control in Dynamic Food Supply Chains using Big Data" published in Computers & Operations Research (Kappelman & Sinha, 2021).

4.1. Introduction

Many approaches to modeling and solving problems in agri-food supply chains argue the complexity of the models and then use approximate methods to analyze the performance of their systems. In contrast, the main contribution of this chapter is to enhance the generalized model for food supply chain by enabling data-driven decision making. The major advantage of data-driven decision models in supply chains is their ability to statistically predict outcome probabilities based on input decisions and information states, which helps to capture the uncertainty introduced by different sources. It also enables the decision maker to understand how these outcomes can contribute to product quality across the entire supply chain. This is accomplished by integrating data mining techniques with the MDP model developed in Chapter 3 to determine the optimal suppliers and their process control parameters at each level of the supply chain. With the consideration of perishable products, we maximize the quality of the final product. Further, numerically the impact of costs and quality constraints on the optimal decisions is explored.

Chapter 4 is comprised of six sections. Section 4.2 integrates a Bayesian network to determine the transition probabilities for the MDP model and explains the limitations to this approach. Section 4.3 provides some detail behind the motivation for Big Data mining and details for a novel ideal on how to apply of these techniques to determine the transition model. Section 4.4 describes our experimental results including some performance studies of the new technique and sensitivity analysis of the optimal policy vs. stricter quality requirements. Section 4.5 concludes our work and highlights the contributions of this chapter.

4.2. Integration with Bayesian network

It is recognized that a common weakness with using MDP as a modeling framework is that most related problems in the literature assume a transition matrix is provided. To determine the transition matrix from historical data, this section integrates Bayesian networks to work with the MDP model established in Chapter 3.

The transition probabilities for our MDP model are defined using a Bayesian network (or Bayes net), which is a graphical structure (often referred to as a belief network) that can be custom designed to represent set of probabilistic relationships between variables of interest (Neapolitan, 2004). This allows us to easily design and select a network hierarchy that learns the transition probabilities appropriate for our MDP model from available data.

The goal task of a Bayes net is to learn a classification function, which represents a mapping of our data inputs (supply chain step, quality level, supplier selection, and settings for their process parameters) to our data outputs (next supply chain step and new quality level). To describe how this is accomplished, we will begin by outlining the notation we used for our data.

Let w_j be the input parameter where $j \in \{1,2,\ldots,n+2\}$. Our input (feature) vector $\mathbf{w}$ is the collection of all our input parameters, see Equation (4.1). This vector includes the input state

s and the action selected $\mathbf{x}$. We are assuming that this problem is not history dependent, and therefore the actions taken in earlier steps are not a necessary input. The $n+2$ notation is because n is the length of our action vector $\mathbf{x}$, and the state s is two-dimensional containing both the current supply chain step k and achievable quality level y_k.

$$\mathbf{w} = [w_1, \dots, w, \dots, w_{n+2}] = [k, y_k, x_k^1, \dots, x_k^n] = [s, \mathbf{x}_k] \tag{4.1}$$

Data outputs are represented by our output (label) vector $\mathbf{u}$, see Equation (4.2), which represents the resulting state s'.

$$\mathbf{u} = [k', y_k'] = [k+1, y_{k+1}] = [s'] \tag{4.2}$$

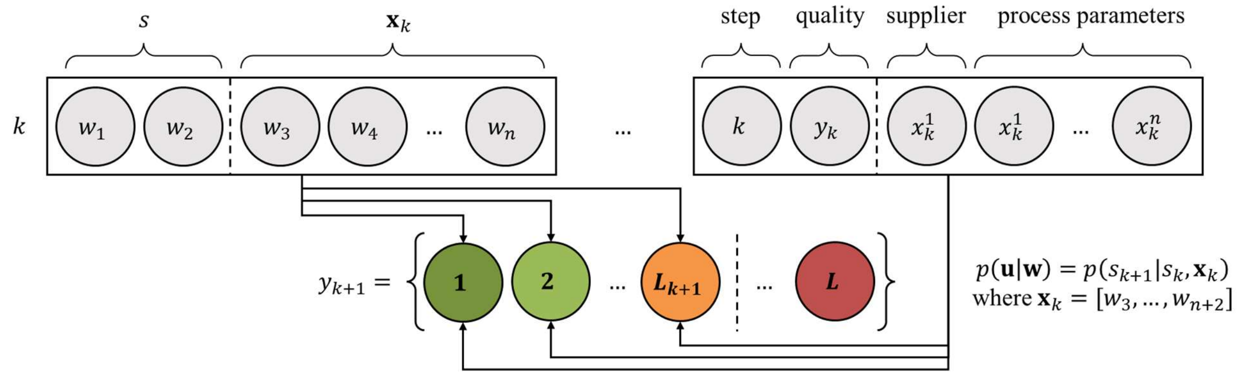


Figure 4.1 Bayes net which represents a mapping of our input state (including the supply chain step, quality level, and actions selected to our output quality level

Let's say, $D = (W, U) = \{(\mathbf{w}^1, \mathbf{u}^1), \dots, (\mathbf{w}^d, \mathbf{u}^d), \dots\}$, be the set of training data where $\mathbf{w}^d \in W$, $\mathbf{u}^d \in U = S'$. The goal task is to learn a classification function: $f: W \rightarrow U$. This is a mapping from our input (feature) vector $\mathbf{w}^d$ to our output (label) vector $\mathbf{u}^d$. The classification function our Bayes net learns is simply our transition probability distribution, see Equation (4.3). Figure 4.1 displays a simplified diagram of our Bayes net.

$$f = p(\mathbf{u}|\mathbf{w}) = p(s'|s, \mathbf{x}_k) = p(k+1, y_{k+1}|k, y_k, \mathbf{x}_k) \tag{4.3}$$

We can estimate $p(\mathbf{u})$ using available data based on the frequency of occurrence for each $\mathbf{u} \in U$. However, it is not likely to have many occurrences of the unique combination for any given $\mathbf{w} = [w_1, w_2, \dots, w_{n+2}]$ when the data is limited. One way to work around this is to take a *naïve* approach and assume that all the features (parameters) in $\mathbf{w}$ are conditionally independent (referred to as a naïve Bayes classifier) (Bishop, 2006). For smaller problems this approach normally does not need to be taken. For larger problems, exact inference may be challenging to compute so either approximation methods (like naïve Bayes) or more efficient approaches should be considered.

4.3. Big Data Mining

4.3.1. Motivation for the use of Big Data mining

One of the primary challenges with MDPs in practice is that the transition model may not be well understood, and for this reason, it is sometimes assumed to be already known in textbooks. However, in practice the P-matrix needs to be determined from prior knowledge or data. For the case where we desire to learn this from data, one method is to model and solve for your transitional P-matrix using a Bayesian Network (as described in section 4.2). The challenge with this method is that computation of exact inference in a Bayes net is $\#P$-complete (Roth, 1996), which asks the question of *how many*. This is at least as hard as the NP-hard equivalent question of *are there any*. Therefore, determining the P-matrix might restrict the complexity of the MDP problem capable of being solved in a reasonable amount of time. This is especially the case for large or complex MDP problems where the state space can grow substantially as more information is considered and the action space grows exponentially as the number of possible control parameters or their potential settings are increased.

Most techniques to improve the speed of this computation require making approximations. Such is the case when using a Naïve Bayes approach, which assumes conditional elements are independent. The Naïve Bayes approach is time efficient and as a technique it is considered very reliable for classification. However, it is not a good method for parameter learning or exact inference because of the underlying assumption of independence (Zhang H. , 2004). The remainder of Chapter 4 aims to perform this computation utilizing a Big Data framework to make this more efficient and potentially avoid having to utilize approaches that can only solve this approximately. It may also allow us to solve larger problems where the amount of the data is simply too large to handle without specialized tools or techniques.

4.3.2. Solution method theory for determining the MDP transition probabilities

In subsection 2.2.1 (associate rule mining) we showed how directional relationships between items could be determined by looking relative occurrence within transactions $j \in J$. From these relationships rules can be formed. Two metrics were formed for these rules and their items called support and confidence. As seen in Equation (4.4), support is the probability that item W will be included in a randomly selected transaction j. We will write this out as $p(W)$ for shorthand.

$$\text{supp}(\{W\}) = \text{Prob}(\{W\} \in j) = p(W) \tag{4.4}$$

Confidence was the directional relationship between the items W and U that was a measure of how often the rule $(W \Rightarrow U)$ was found to be true. After some further refining of this expression, see Equation (4.5), the confidence of the rule is equivalent conditional probability that the new item U will be in a random transaction j given that W is also in the transaction. This relationship can be expressed as $p(U|W)$.

$$\text{conf}(Rule) = \frac{\text{supp}(\{W, U\})}{\text{supp}(\{W\})} = \frac{\text{Prob}(\{W, U\} \in j)}{\text{Prob}(\{W\} \in j)} = \frac{p(W \cap U)}{p(W)} = p(U|W) \tag{4.5}$$

In the next section, we will expand this expression to show how we can use these relationships to determine our conditional transition probabilities $p(s'|s,\mathbf{x})$ that comprise the transition model (P-matrix) of our MDP problem.

4.3.3. Application of AR mining to solve for *P*-matrix

Given a data set D with elements $d \in D$ such that each data element contains the information $d = (s, \mathbf{x}, s')$ where s is the from state, s' is the to state, and $\mathbf{x}$ was the action taken, determine the conditional transition probabilities $p(s'|s,\mathbf{x})$ that describe the probability that taking action $\mathbf{x}$ in state s will lead to state s'. These elements make up the P-matrix which is our transition model for the MDP problem.

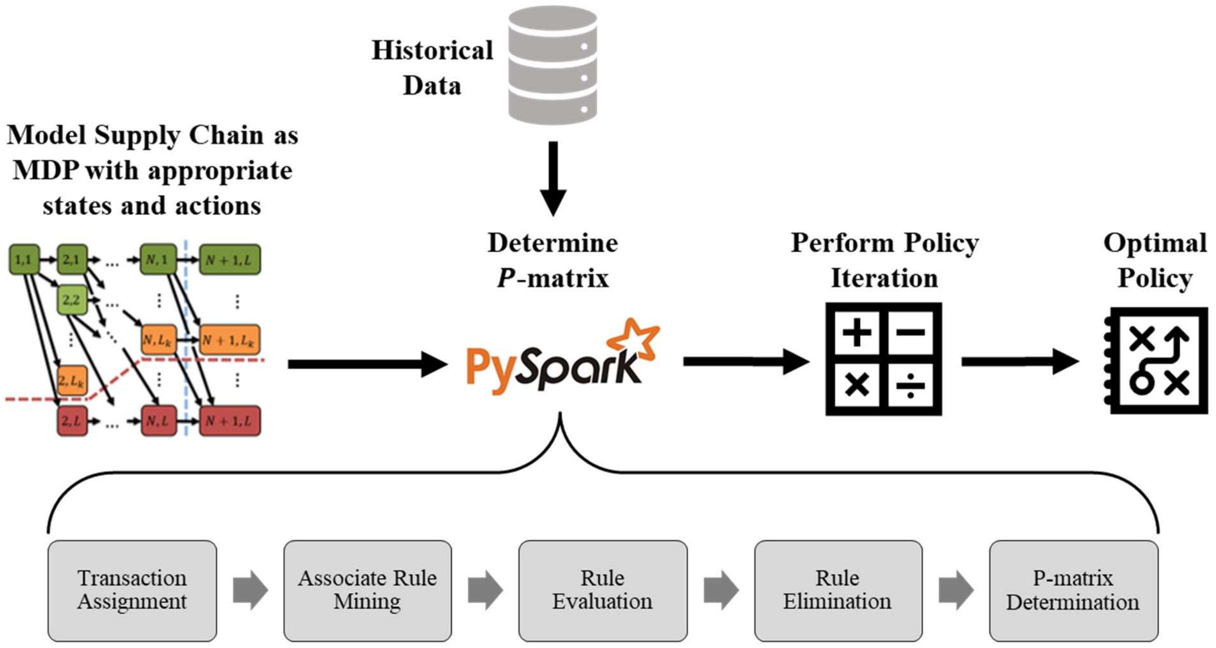


Figure 4.2 Framework figure for data-driven MDP model for a supply chain with P-matrix determined using AR mining

Our proposal is to implement the procedure as in the bottom of Figure 4.2 to determine the P-matrix. The framework figure at the top of Figure 4.2 shows how this technique works with the MDP model for finding an optimal policy for the desired supply chain problem. Here is a detailed description of the procedure steps used to determine the P-matrix using the Big Data tool PySpark to implement AR mining:

Step 1. Transaction Assignment

Assign the data elements $d = (s, \mathbf{x}, s')$ from our MDP problem into transactions $j \leftarrow d$ such that our transactions take the form $j = \{(s, \mathbf{x}), s'\}$.

Step 2. AR Mining

Perform associate rule mining to evaluate the rules that connect potential items in these transactions. Note: the only rules that we are interested in take the form $Rule = \big((s, \mathbf{x}) \Rightarrow s'\big)$. These rules give us the expectation for outcome state s' given input state-action pair $(s, \mathbf{x})$.

Step 3. Rule Evaluation

The confidence for these rules represents the relationship between all possible input state and action pairs, $(s, \mathbf{x})$, and all potential output states, s'. The conditional probabilities for our P-matrix are the confidence for the specific rules we are interested in per Equation (4.6).

$$\mathrm{conf}(Rule) = \mathrm{conf}\big((s, \mathbf{x}) \Rightarrow s'\big) = \frac{\mathrm{supp}(\{(s, \mathbf{x}), s'\})}{\mathrm{supp}(\{s'\})} = \frac{p(s, \mathbf{x}, s')}{p(s, \mathbf{x})} \tag{4.6}$$

$$= p(s'|s, a)$$

Step 4. Rule Elimination

Eliminate any rules that do not take the form $Rule = \big((s, \mathbf{x}) \Rightarrow s'\big)$. This removes any reverse rules that take the form $\big(s' \Rightarrow (s, \mathbf{x})\big)$.

Step 5. P-matrix Determination

Using the remaining rules, assemble the P-matrix for our MDP.

$$P = [p(s'|s,\mathbf{x})] = \left[\text{conf}\big((s,\mathbf{x}) \Rightarrow s'\big)\right] \ \forall\, \mathbf{x} \in X \text{ and } s, s' \in S \tag{4.7}$$

This technique requires selecting appropriate minimum confidence values that do not omit valuable information but avoid creating relationships that have no meaning in the context of the MDP problem. This offers a potential solution to one of the shortcomings of traditional methods as they do not typically consider minimum frequency before establishing a rule. Because of this, we believe that the minimum support rules built inside these Big Data mining tools can help remove outliers that might otherwise strongly influence the optimal policy. Exact tuning for setting good threshold values for these would depend on the problem and a variety of other factors.

Many tools have been created help mine these rules such as the arules package in R which includes implementations of the popular algorithms Apriori and Eclat (Hahsler, Grun, Hornik, & Buchta, 2005). We will utilize a Big Data tool that allows for parallel implementation of another algorithm that mines for frequent patterns called FP-Growth (Han, Pei, & Yin, 2000). Spark's implementation uses a parallelized version of the FP-Growth algorithm (Anonymous, n.d.) called PFP (Li, Wang, Zhang, Zhang, & Chang, 2008).

4.4. Numerical Experiments

In this section, we will describe two numerical experiments that were performed. The first is a set of performance tests that were done to show how the Big Data technique compares in running time to a more standard technique. The second set of experiments provide an example for how this data-driven stochastic modeling technique can be implemented. We also show how the optimal policy changes as we modify the minimum quality threshold for acceptance, L_k to emphasize the influence quality standards can have on food loss and waste.

4.4.1. Big Data technique performance tests

The problem setup for our performance tests included first identifying a state space size $|S|$ and an action space size $|X|$. Ten data points per state-action pair were generated such that $|D| = 10 \times |S| \times |X|$ was the total number of data points created. This data was generated randomly according to these parameters and stored in a CSV file. The two techniques were then compared for 3 runs each for multiple state and action space sizes. The state space size used were 50, 100, 150, and 200. The action space sizes used were $5^3 = 125$, $5^4 = 625$, $4^5 = 1024$, and $4^6 = 4096$ where $|X| = h^n$ and $n =$ number of action parameters and $h =$ number of parameter settings. For the traditional technique to compare to our proposed Big Data technique we implemented a dictionary method to count the frequency of unique events for state s, action a, and resulting state s'. For all the experiments we used a Core i5 3.6GHz machine with 12 cores and 32GB RAM. The full run results can be seen in Table 4.1.

The traditional (dictionary) method proved highly efficient for smaller problems, which suggests that a Big Data technique is not warranted for these computations. However, as either the state space or the action space grew, the problems quickly became too much for the traditional technique to manage. Only for the smallest state space $|S| = 50$ was this method faster for all action space sizes tested, but it does appear that eventually the two would likely cross, given lager experiments. The Big Data technique showed to be the more efficient method for any of the larger combinations.

Table 4.1 Average running time (s) to solve for P-matrix using Big Data and traditional methods

State space size $|S|$	Action space size $|X|$	Big Data technique	Traditional method
50	125	5.77	0.38
	625	9.11	2.23
	1024	11.51	3.61
	4096	22.06	16.87
100	125	6.60	1.45
	625	12.65	8.51
	1024	15.38	13.51
	4096	41.51	66.07
150	125	7.87	3.56
	625	16.05	19.34
	1024	20.72	34.14
	4096	65.96	188.21
200	125	9.16	5.71
	625	19.38	35.04
	1024	26.31	56.34
	4096	84.42	346.21

For the most extreme case where $|S| = 200$ and $|X| = 4096$, the traditional method was unreliable and crashed on multiple occasions. Five runs were attempted in order to collect results for our desired three. The greatest time commitment for the Big Data technique for larger problems was the rule collection (steps 4 and 5 in the procedure). This is because these results were being collected from a single machine and not several independent machines. Also, the transaction assignment step was not optimized for parallel computation, and it grew linearly according to the amount of data $|D|$. Figure 4.3 shows their relative performances by plotting the ratio of running times for the traditional (dictionary) method vs the Big Data technique as the state and action spaces are increased.

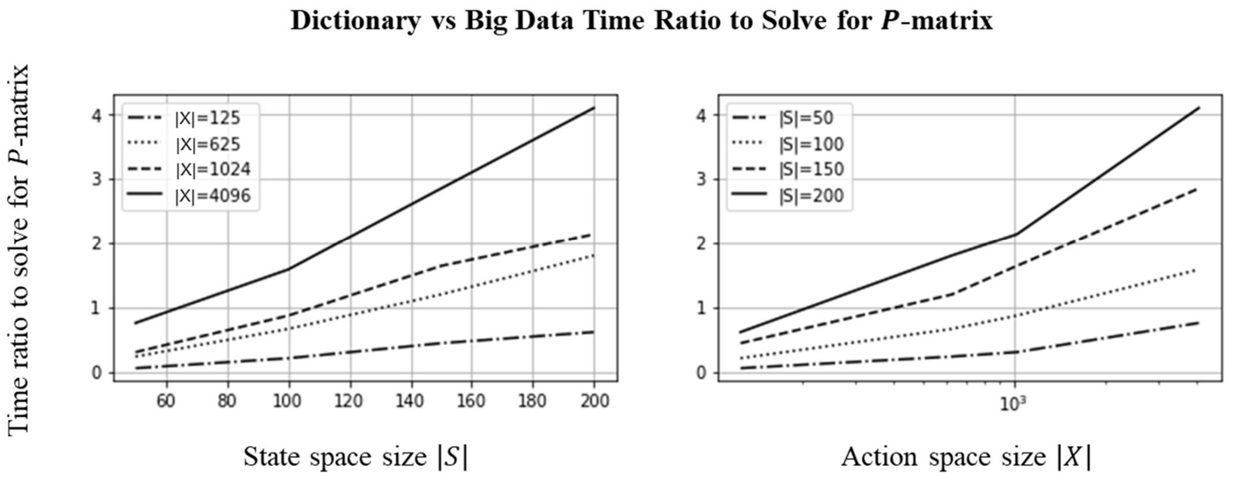


Figure 4.3 Ratio of running times (Dictionary vs Big Data) to determine P-matrix as state space size $|S|$ and action space size $|X|$ grow.

These results show that the proposed Big Data technique is a reliable and time efficient technique to solve for the P-matrix as an MDP problem's complexity requires further expansion of the state and/or action spaces. For smaller problems, the Big Data technique is sufficient but not necessary.

4.4.2. Example Implementation and Sensitivity results for Optimal Policy

The following experiment provides an example for how this data-driven stochastic modeling technique can be implemented. It includes multiple runs for sensitivity analysis to evaluate how the optimal policy changes as the minimum quality threshold for acceptance L_k becomes more restrictive, as this can affect food loss and waste. These experiments are performed via simulation for two different distributions (uniform and normal) of outcomes for the quality level achieved in the subsequent step, given the actions selected. In an application study, the appropriate distributions would come from the data as this technique should work for all distributions. However, if domain knowledge about these distributions has already been established for the specific application, then it is advised that a simulation study (similar to this) be conducted to create a theoretical benchmark to compare outcomes.

The problem setup we consider includes a total of 5 decision epochs prior to the final stage where a benefit is finally realized, so $k \in \{1, \ldots, 6\}$. All products begin at an initial quality level of $y = 1 =$ best and in later can potentially reach any of the possible levels $y' \in \{1, \ldots, 10\}$ where $10 =$ worst. At each of these decision epochs, there are a total of $n = 5$ parameters to be selected from $h = 3$ possible settings each. This results in an action space of $|A| = 3^5 = 243$ possible actions. To ensure each action has a high possibility of being explored, we generated a total of 729,000 original data points, or 50 per possible state-action pair. These all start at $s = (k, y_k) = (1,1)$, but from there the step is incremented by 1 ($k' = k + 1$) and the quality can only be maintained or depreciate ($y' = y_{k+1} \geq y_k$). This depreciation is random but based on the investment (cost) in the prior actions $C(\mathbf{x}_k)$. Note that quality depreciation is with respect to the end-product. For products that can experience valued added processes, any action and outcome that does not add the most value would be described as a decrease in the value of the end-product. Two distributions were investigated to describe the depreciation distribution, Normal (using $\mu \pm 2\sigma$ for extreme cases) and Uniform. Both distributions were based on the current level $y'_{\text{best}} = y$ and the worst possible level y'_{worst}, which was a linear function based on the investment in actions. More expensive actions and better initial quality levels provided a more favorable distribution and the alternatives resulted in a less favorable distribution. This is shown in in Figure 4.4. In a specific application study, the appropriate cost function should be used for the product and supply chain. This could include fixed costs and other factors resulting in a non-linear cost function. However, for simplicity in this simulation study, a linear relationship between action investment and worst quality outcome was used.

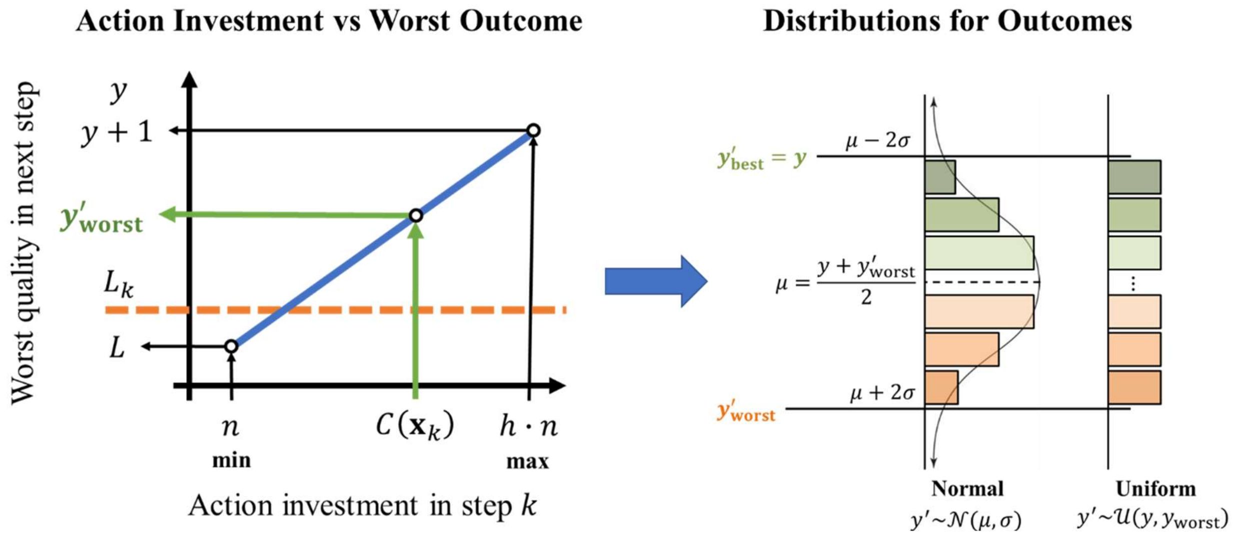

Action Investment vs Worst Outcome
Worst quality in next step
y
$y+1$
y'_{worst}
L_k
L
n
min
$C(\mathbf{x}_k)$
$h \cdot n$
max
Action investment in step k
Distributions for Outcomes
$y'_{\text{best}} = y$
$\mu - 2\sigma$
$\mu = \frac{y + y'_{\text{worst}}}{2}$
$\mu + 2\sigma$
y'_{worst}
Normal
$y' \sim \mathcal{N}(\mu, \sigma)$
Uniform
$y' \sim \mathcal{U}(y, y_{\text{worst}})$

Figure 4.4 Graphics showing the relationships between action investment $C(\mathbf{x})$ and the distributions of possible outcomes for y' that were tested.

The benefits were zero in all steps except the last, $k = 6$ where a linear relationship was scaled from $B(6,1) = 100$ as the highest final benefit and decreasing to $B(6,10) = 60$ was the lowest benefit, as shown in Equation (4.8). For cases where $y_6 > L_k$, this benefit was zero as the product was rejected. The cost equation utilized was a simple summation of the action values as can be seen in Equation (4.9).

$$B(k = 6, y_k) \in [100,60] \quad \leftrightarrow \quad y_k \in [1,10] \tag{4.8}$$

$$C(\mathbf{x}_k) = \sum_{i=1}^{n} x_i \;\text{ where }\; x_i \in \{1, \dots, m\} \tag{4.9}$$

Using this framework, we simulated the data for 10 experiments using each distribution at three different L_k settings. These settings were $L_k = 10 =$ unrestricted, $8 =$ partially restricted, and $5 =$ highly restricted. The final results for the amount of investment in actions $C(\mathbf{x}_k)$ as suggested by the optimal policy versus the L_k setting can be seen in Figures 4.5 and 4.6 for normal distribution and uniform distribution.

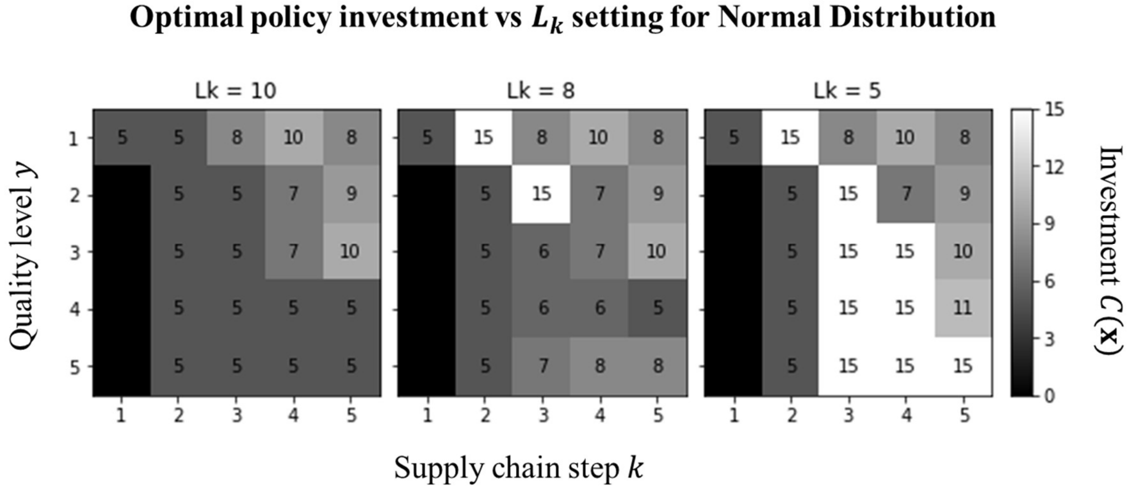


Figure 4.5 Results for optimal policy investment $C(\mathbf{x})$ vs the current quality level L when the outcome quality level y' follows a normal distribution

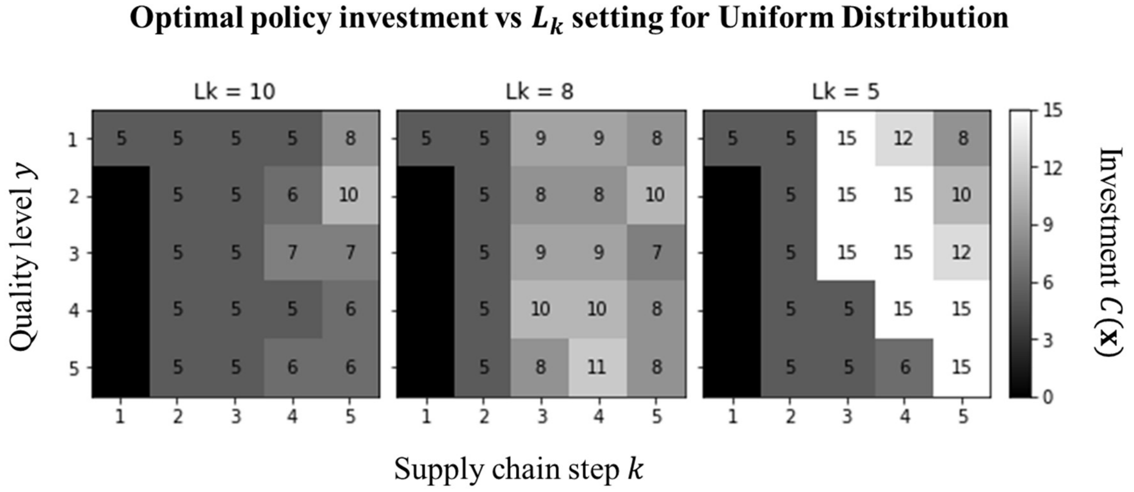


Figure 4.6 Results for optimal policy investment $C(\mathbf{x})$ vs the current quality level L when the outcome quality level y' follows a normal distribution

These results show that the general trend is to invest more in high quality actions as the quality threshold becomes more restrictive. It is observed that at the initial stages high investments are not warranted, which is most likely based on the margins between different of outcomes for this specific problem. If this supply chain had more steps with similar distributions of outcomes, those higher investments might creep backward to some of the earlier steps as well.

The optimal policy investment when the quality levels are higher (upper right and lower left) were relatively stable, and the increase in investment triggered by quality restrictions is typically made at the interior states. For the upper right states, a high-quality outcome had already been secured with a small or marginal investment so further investments were not needed. At the lower left states, the lowest investment was always made because higher investments do not create enough change in the probability of outcomes to sacrifice the product. The decision maker should probably seek earlier salvage opportunities for those products in the lower left states that have a high enough probability of rejection in the later states. This information would be extremely useful for the decision maker so they could avoid unnecessary investments by making better decisions and improve their supply chain's profitability.

4.5. Conclusions

This chapter addresses a few complexity issues with modeling dynamic food supply chains to account for uncertainties present and the vast array of potential decisions and possible quality states. We introduce a general mathematical model for a food supply chain as a Markov decision that considers the stochastic effect these decisions have on a product's quality. Since the transition model is typically unknown in practice, we determine these probabilities from historical data. It is recognized that as a problem's complexity increases, the states and potential actions that need to be considered grow exponentially so we take an integrated approach that involves Big Data mining techniques. We formalize a procedure to determine values for our transition matrix using this new method. Performance tests show how the running time improves when using this Big Data technique versus standard methods for complex problems. Lastly, we conduct experiments showing how the optimal policy changes as the quality threshold for rejection becomes more restrictive which identifies when it is either valuable or unnecessary to invest in higher quality actions.

CHAPTER 5. DEEP RL FOR OPTIMAL CONTROL IN STOCHASTIC IPS

In previous chapters we established modeling dynamic agri-food supply chains and how to make them data-driven. Chapter 5 explores the idea of leveraging deep reinforcement learning to solve for stochastic sequential integer linear programs such as the mix and blend problem, which is commonly found in agri-food supply chains. This chapter is based on the manuscript “OR-Net: An Efficient Neural Network for Optimal Control in Stochastic Integer Programs” which has been submitted for journal publication to International Journal of Modelling and Simulation.

5.1. Introduction

Grains, oilseeds, and other granular products make up a very large percentage of agricultural exports (Kenner, Jiang, & Russell, 2022). Their quality characteristics can be customized to meet specific orders and are often controlled through mixing and blending processes. These processes commonly are executed repetitively, forming a series of sequential decisions that each aim to meet quality constraints while minimizing costs. Within a single decision frame, the mix and blend problem is a specialized version of the knapsack problem, which is a binary integer program. Integer programming (IP) is an extremely important mathematical optimization problem that comes up in many industries. Problems related to scheduling, staffing, resource allocation, equipment/machine settings, healthcare and biology, advertising, portfolio selection, logistics and vehicle routing all depend on IP models to deliver optimal or near optimal results. IP is one of Karp’s original 21 NP-Complete problems (Karp, 1972). Because of its inherent difficulty, approximation methods are commonly explored, especially for larger problems and dynamic rolling time-horizon or stochastic variants of these problems.

The field of deep learning (DL) has made several advancements in recent years, solving problems with neural networks (NN) to performance levels beyond traditional machine learning (ML) that were previously believed to only be possible by humans. This is in part due to the strong capabilities NN have with complex function approximation. Efficient network architectures have also been introduced, such as convolutional neural networks for image tasks (Krizhevsky, Sutskever, & Hinton, 2017), that greatly improve the use of computational resources. Deep reinforcement learning (DRL) is a field that has shown tremendous performance in sequential decision-making problems, usually proven in controlled gaming environments. More recently DRL has also been able to perform optimization related tasks like inventory control (Oroojlooyjadid, Nazari, Snyder, & Takáč, 2017) even better than traditional techniques, especially when operating in a stochastic environment. It is important to note that ML or DL approaches to COPs are best applied to problems where either an exact solution cannot be found in reasonable amount of time or there are many sources of uncertainty surrounding the problem that make it difficult to contract into a simple formulation.

This chapter is organized into seven sections. Section 5.2 outlines methods used to model sequential IPs using a MDP framework and solve using DRL. Section 5.3 introduces the OR-Net including the motivation, problem statement, and technical approach. Section 5.4 covers the experimental setup and preliminary results. Section 5.5 provides a conclusion to this chapter.

5.2. Modeling Sequential ILPs as an MDP

5.2.1. Integer Programming

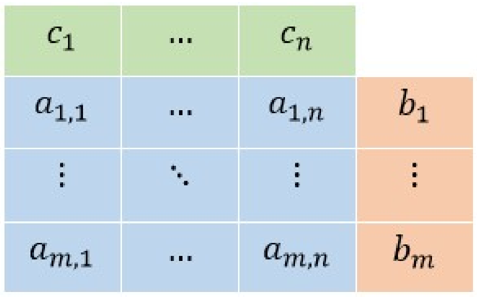
Integer programs (IP) are concerned with finding optimal solutions to problems where the objective function (goal) and constraints (requirements or restrictions) are mathematical equations, and the decision variables are integer. The most popular of these are integer-linear programs (ILP), where the objective function and constraints are linear. The standard form for an ILP is in Equation (5.1), where **x** is the vector of decision variables, **A** is a matrix of attributes, **b** is a vector of bounds, and **c** is a vector of costs.

$$\begin{aligned} \max \;\; & z = \mathbf{c}^\top \mathbf{x} \\ \text{s.t.} \;\; & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & x_j \geq 0 \\ & x_j \in \mathbb{Z}^n \end{aligned} \tag{5.1}$$

Decision variables, **x**: The decision variables, a vector of length n such that $\mathbf{x} = [x_1, \dots, x_n]^\top$, are the choices that the decision maker can control. Binary integer programs (BIP) are such that $x_j \in \{0,1\}$.

Objective function, z: The objective function, $z = \mathbf{c}^\top \mathbf{x}$, is key metric that should be optimized. The goal of integer programming is to determine the decision variables that provide the optimal solution, $\mathbf{x}^* = \arg\max_{\mathbf{x}} z(\mathbf{x})$ and $z^* = z(\mathbf{x}^*)$.

The coefficients that help describe a standard ILP problem are attributes, bounds, and costs. Figure 5.1 shows how these are potentially related to each other as they influence the selection of the decision variables to make up the optimal solution.



c_1	$\cdots$	c_n	
$a_{1,1}$	$\cdots$	$a_{1,n}$	b_1
$\vdots$	$\ddots$	$\vdots$	$\vdots$
$a_{m,1}$	$\cdots$	$a_{m,n}$	b_m

Figure 5.1 Coefficients in a standard ILP problem

Attributes, **A**: The attributes, a matrix of size $m \times n$, help to describe the relationship between the decision variables and the constraints. A single element $a_{i,j}$ represents the attribute value of the decision x_j as it relates to the constraint bounded by b_i.

Bounds, **b**: The constraint bounds, a vector of length m such that $\mathbf{b} = [b_1, \dots, b_m]^\top$, create limits that describe the constraints of the problem.

Costs, **c**: The cost metrics, a vector of length n such that $\mathbf{c} = [c_1, \dots, c_n]^\top$, describe the value obtained or expense incurred c_i for each decision variable .

One of the limits with modeling meaningful optimization problems with IP, is that they are deterministic, meaning they do not capture the uncertainty that is often encountered in real world problems. In the following section we will outline a method for modeling a rolling time horizon sequential ILP using a MDP framework.

5.2.2. Markov Decision Processes Models for Integer Programs

Markov decision processes (MDPs) are discrete stochastic processes that provide a mathematical framework for modeling decision making in situations where outcomes are probabilistic (Puterman, 2005). They are comprised of the following elements.

Decision Epochs: Epochs $t = 1, \dots, T$ are defined as the periods when decisions are made, rewards are collected, and state transformations are observed.

State Space, S: Let $s_t \in S$, be the state of the system that is defined as a vector comprising of the relevant information needed to make decisions and quantify outcomes at epoch t. The state for our ILP contains the ILP coefficients **A**, **b**, and **c**. These can be flattened into a vector $s_t = [\mathbf{A}|\mathbf{b}|\mathbf{c}]$ with a length of $nm + m + n$. Note that since our ILP coefficients are continuous, the corresponding state space of our MDP is infinite.

Action Space, X: Let $\mathbf{x}_t \in X$, be the actions selected at the decision epoch t. The actions in our MDP model are the same as decision variables in the ILP specific to the current epoch t.

Rewards: A reward $r_t \sim R(s_t, \mathbf{x}_t)$ is the expected net benefit received for being in state s_t and taking action $\mathbf{x}_t$. The total set of rewards can normally be defined using a function or a matrix, but since the ILP has an infinite state space we will need to use a function. This function will be the unconstrained objective function, see equation (5.2). A penalty λ is applied for not complying with the constraints. Operating in an unconstrained environment allows our decision maker an environment to both maximize the objective function and comply with the constraints with a single function.

$$R(s_t, \mathbf{x}_t) = \mathbf{c}^\top \mathbf{x} - \lambda^\top \max(\mathbf{A}\mathbf{x} - \mathbf{b}, 0) \tag{5.2}$$

Transition Model, P: A transition probability $p(s_{t+1}|s_t, \mathbf{x}_t)$ is defined as the probability distribution from state s_t given action $\mathbf{x}_t$ to state s_{t+1}. These probabilities are Markovian, meaning that only the most recent state s_t and action $\mathbf{x}_t$ are needed to know the distribution of next possible outcome states s_{t+1}, not the entire history of states s_0 to s_t. The transition model includes all the transition probabilities within the state space S, which is infinite for our ILP so we will have to consider this when determining our solution approach.

Policy: A policy π is defined as a mapping of actions to take given the state. This is represented as $\pi = \{\mathbf{x}_{(s)} \text{ for all } s \in S\}$ such that $\mathbf{x} = \pi(s)$ is the action suggested to be taken by policy π when at state s. A stochastic policy $\pi(\mathbf{x}|s)$ is when the outcome mapping of the policy is the probability that the action $\mathbf{x}$ will be selected given the state s. Note that the policy determination will face the same challenges due to having an infinite state space.

Value Function: The value function for a policy π is V_π, which can be described recursively as in Equation (5.3) using Bellman's equations, where $\pi(s)$ is the action to take as determined by the policy π and γ is the discount factor that prevents the value function from going to infinity. The optimal policy π^* is defined as the decision policy which maximizes the value function, see Equation (5.4).

$$V_\pi(s) = R\big(s, \pi(s)\big) + \gamma \sum_{s' \in S} p\big(s' | s, \pi(s)\big) V_\pi(s') \tag{5.3}$$

$$\pi^* = \underset{\pi'}{\operatorname{argmax}} V_{\pi'} \tag{5.4}$$

The goal of dynamic programming is to find the optimal policy π^* that maximizes this value function at every state $s \in S$. There are different approaches of finding optimal policies for MDPs, but they can mostly be described as *model-based* methods and *model-free* methods. Model based methods require knowledge or an assumption of the transition model P. Under this knowledge, exact methods such as Policy Iteration algorithm (Bellman, 1955) can guarantee optimality of the proposed optimal policy. Unfortunately, since our ILP has an infinite state space this problem deals with what is referred to as the “Curse of Dimensionality” (Powell, 2011). The goal of this MDP model was to solve rolling time horizon sequential ILP problems that deal with sources of uncertainty, so model-free methods may need to be considered.

5.2.3. Reinforcement Learning

In this section, popular model-free methods will be discussed as these techniques try to address these concerns. These methods come from an area of research with many names including Approximate Dynamic Programming (ADP) (Powell, 2011), Reinforcement Learning (RL) (Sutton & Barto, 2018), and Neuro-Dynamic Programming (Bertsekas & Tsitsiklis, 1996). These are all very similar fields and mostly differ in the specific applications they are interested in addressing, but in this chapter, we use RL to refer to this field. The most common framework for model-free solutions is by using a learning agent that interacts with its environment through a feedback loop and updates its decision policies through experiences, see Figure 5.2.

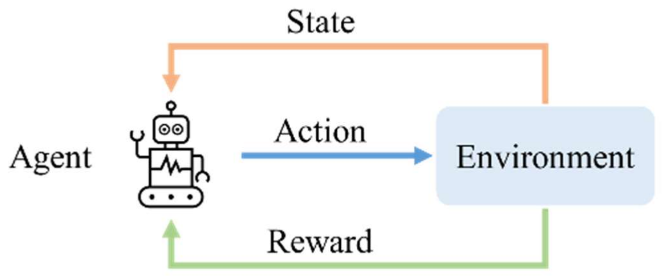


Figure 5.2 RL agent interacting with its environment through a feedback loop

The most popular model-free technique in these fields was introduced by Watkins and it is called Q-learning (Watkins & Dayan, 1992). This well-known method is named after the values it aims to estimate. Q-values are estimates of the value of being in state s and taking action $\mathbf{x}$, see Equation (5.5). This method becomes model free is when knowledge of the outcome state s' and reward r_s is available to approximate $R(s, \mathbf{x})$.

$$\begin{aligned} Q^*(s, \mathbf{x}) &= R(s, \mathbf{x}) + \gamma \sum_{s' \in S} p(s'|s, \mathbf{x}) V^*(s) \\ &\approx r_s + \gamma \max_{\mathbf{x}'} Q^*(s', \mathbf{x}') \end{aligned} \tag{5.5}$$

The strengths of Q-learning lie with its ease of computation and when given an infinite exploration of all state action pairs $(s, \mathbf{x}) \in (S, X)$, due to Banach fixed-point theorem, this algorithm will converge on the true Q-values Q^* and find the optimal policy π^* (de Farias & Van Roy, 2000). However, a major limitation of traditional tabular Q-learning is that both the state and action spaces are required to be discrete, which they are not in our ILP problem. Another limitation of Q-learning is that it cannot infer the Q-values of states or actions it has never visited from other experiences. Newer methods in the area of function approximation allow for these two assumptions to be dropped. One of the most profound and active areas of research in this domain is called Deep Reinforcement Learning (DRL). This field includes fresh concepts such as DQN and policy gradients.

The central idea behind policy gradients was that most other DRL methods including DQN were trying to solve for the value of the state $V(s)$ or the value of the state and action $Q(s, \mathbf{x})$. However, if the end goal is to use these values to determine the optimal policy, then why not just solve directly for the policy?

The policy gradient is defined in Equation (5.7) as the gradient that minimizes the loss function, Equation (5.6), of the log probability of the action taken based on the policy. The goal is to increase the probability of actions that deliver good rewards and minimize the probability of actions that deliver bad rewards. The policy gradient does this by defining the direction ∇J to change our policy network's parameters θ such that it improves the policy to collect better rewards.

$$\mathcal{L} = -Q_\theta(s, \mathbf{x}) \log \pi(\mathbf{x}|s) \tag{5.6}$$

$$\nabla J \approx \mathbb{E}[Q_\theta(s, \mathbf{x}) \nabla \log \pi(\mathbf{x}|s)] \tag{5.7}$$

One of the most popular policy gradient methods is the REINFORCE algorithm (Williams, 1992). Its steps are as follows:

Step 1. Initialization

Initialize the network weights θ with random values. Start at episode $j = 1$ and epoch $t = 1$ with a random starting state $s_{t=1}^{j=1}$

Step 2. Trajectory rollout

Play J full episodes consisting of T epochs each, saving the data elements $d_t^j = (s_t, \mathbf{x}_t, r_t, s_{t+1})^j$ for all $t = 1, \dots, T$ and $j = 1, \dots, J$.

Step 3. Calculate discounted total rewards

For every epoch t of every episode j, calculate the discounted total reward for subsequent steps:

$$Q_t^j = \sum_{i=0}^{t} \gamma^i r_i$$

Step 4. Calculate the loss function for all transitions

$$\mathcal{L} = \sum_{j,t} Q_t^j \log \pi_\theta\left(\mathbf{x}_t^j | s_t^j\right)$$

Step 5. Update weights using SGD

Perform a stochastic gradient descent (SGD) update of weights θ to minimize the loss function $\mathcal{L}$.

Step 6. Repeat until converged

If convergence criteria are met, set $\pi^* = \pi_\theta$. Otherwise, go to step 2 and repeat, resetting at episode $j = 1$ and epoch $t = 1$ with a random starting state $s_{t=1}^{j=1}$.

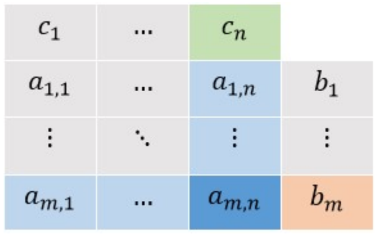
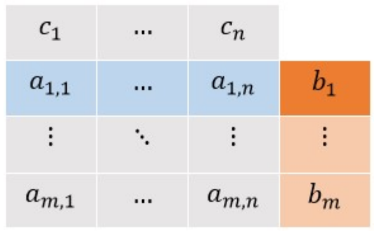
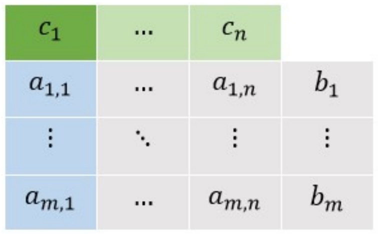
A few of the benefits from the REINFORCE algorithm over Q-learning include that explicit exploration is not needed. Using a stochastic policy $\pi_\theta(\mathbf{x}|s)$ means that exploration happens automatically. By initializing the network with random weights θ, our policy $\pi_\theta(\mathbf{x}|s)$ delivers a nearly uniform probability distribution which generates random agent behavior in the early stages of the algorithm. Also, this approach does not require the use of a replay buffer or a target network. This is because REINFORCE agents would be considered on-policy networks, meaning current policy interacts directly with the environment.

5.3. Orthogonal Relationship Networks

5.3.1. Motivation for an OR-Net

Revisiting Figure 5.1, it makes sense that relationships between the values c_i and $a_{i,j}$ would be of interest. Similarly, the relationships between c_i and c_j or b_i and b_j may also be of interest. However, there does not appear to be a reason to believe that the relationship between c_i and $a_{j,k}$ would be of interest. These relationships that are of interest are between coefficients that are orthogonal to each other.

Table 5.1 Orthogonal Relationships Between ILP Coefficients

Graphic of Orthogonal Relationships	**Relationships of Interest**	**Connections in OR-layer**
	$a_{i,j} \rightarrow \begin{bmatrix} b_i \\ c_j \\ a_{i,k} \\ a_{l,j} \end{bmatrix} \forall k,l$	$nm \cdot \begin{bmatrix} 1 \\ 1 \\ m \\ n-1 \end{bmatrix}$
	$b_i \rightarrow \begin{bmatrix} b_j \\ a_{i,k} \end{bmatrix} \forall j,k$	$m \cdot \begin{bmatrix} m \\ n \end{bmatrix}$
	$c_i \rightarrow \begin{bmatrix} c_j \\ a_{k,i} \end{bmatrix} \forall j,k$	$n \cdot \begin{bmatrix} n \\ m \end{bmatrix}$

Therefore, the idea behind an OR-Net is building a NN that includes an orthogonal relationships layer. This OR-layer reduces the overall connectivity of the network by focusing on the connections (the coefficient relationships) that are of interest, since they are the ones used to solve integer-linear programs in other solution methods. The details of these relationships are described and visualized in in Table 5.1.

5.3.2. Technical Approach

To solve an ILP problem using DRL, the ILP coefficients **A**, **b**, and **c** are used as input features to a NN and can be done so by flattening them into a single vector, such as in Figure 5.3.

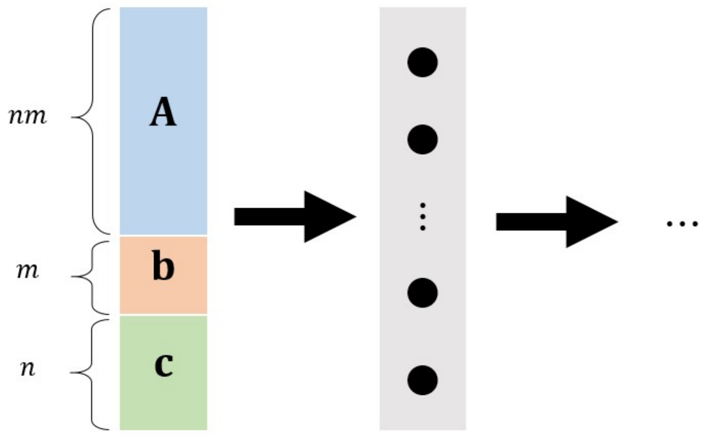
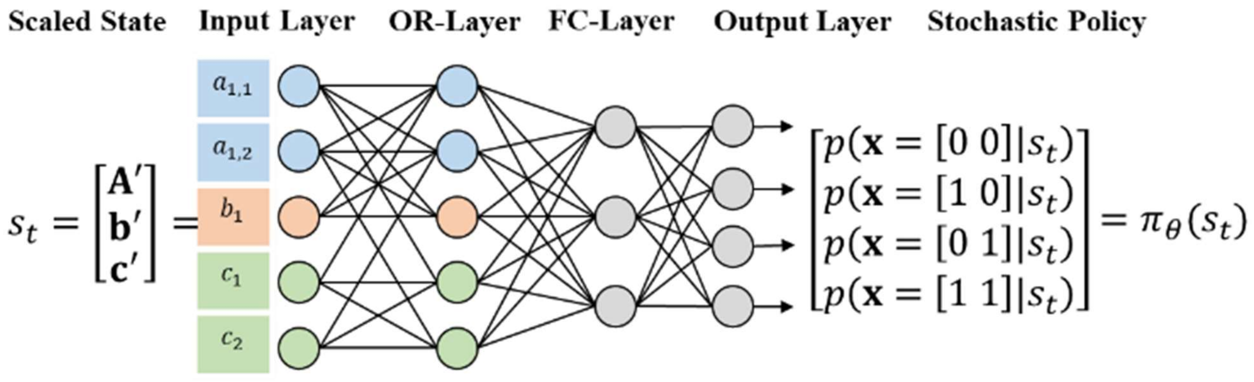


Figure 5.3 ILP coefficients are flattened to be used as the input features for a NN

One approach might be to explore all the possible relationships between the ILP coefficients using a fully connected (FC) layer of equal size to create connections between each coefficient. This would result in a total of $(nm + m + n)^2$ connections in this FC-layer. The total number of connections of interest in a comparative OR-layer is $nm^2 + n^2m + 3nm + n^2 + m^2$ (see Table 5.1 for details). This is far fewer connections in comparison to a fully connected layer. Figure 5.4 displays an example network architecture for an OR-Net being applied to a very small

ILP problem. Figure 5.5 shows how quickly the number of connections in a FC-layer grows in comparison to the connections in an OR-layer.



Scaled State
Input Layer
OR-Layer
FC-Layer
Output Layer
Stochastic Policy
$s_t = \begin{bmatrix}\mathbf{A}' \\ \mathbf{b}' \\ \mathbf{c}'\end{bmatrix} =$
$a_{1,1}$
$a_{1,2}$
b_1
c_1
c_2
$\begin{bmatrix} p(\mathbf{x}=[0\ 0]|s_t) \\ p(\mathbf{x}=[1\ 0]|s_t) \\ p(\mathbf{x}=[0\ 1]|s_t) \\ p(\mathbf{x}=[1\ 1]|s_t) \end{bmatrix} = \pi_\theta(s_t)$

Figure 5.4 Example OR-Network architecture for a small ILP problem (only 2 decision variables and 1 constraint) featuring an OR-layer that only considers the ILP's orthogonal relationships.

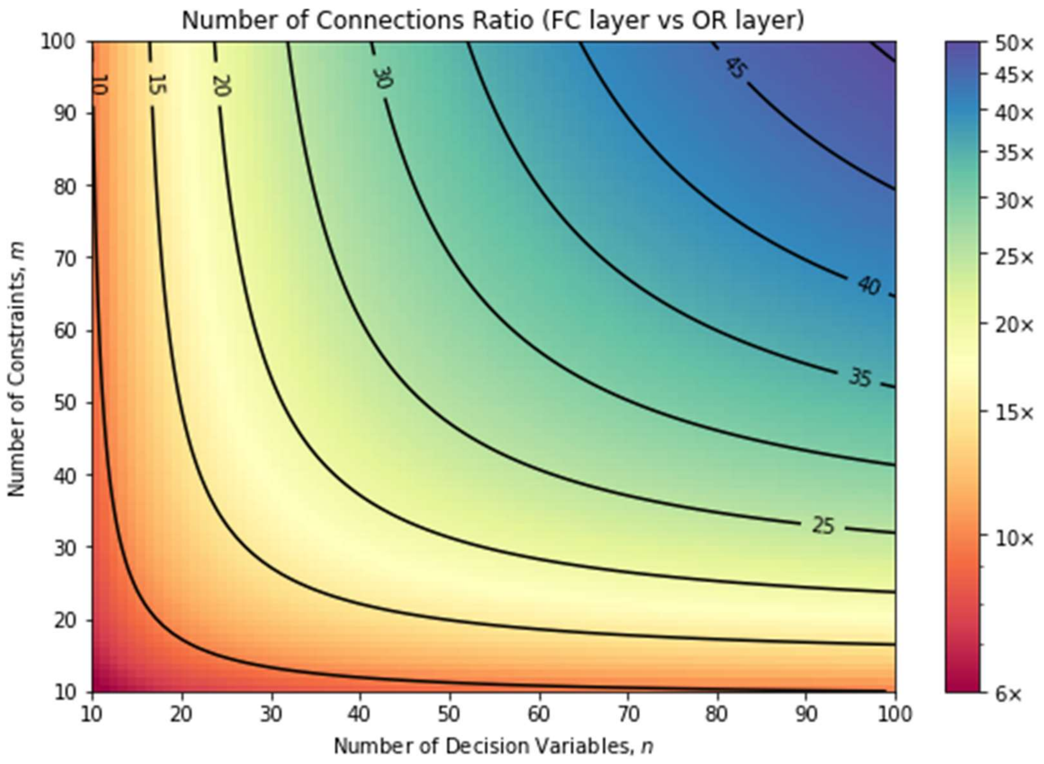


Figure 5.5 Heat-map showing the ratio of the number of connections in a FC-layer vs an OR-layer based on the size of the ILP (number of constraints m and variables n)

In our proposed NN design, the input layer is the scaled state $s_t = [\mathbf{A}'|\mathbf{b}'|\mathbf{c}']$ which is a flattened vector containing all the scaled coefficients of the ILP problem (which are described in detail subsection 5.3.3). The output layer is the policy and is the size $|X|$ of the action space so the actions are mutually exclusive. As displayed in Figure 5.5, even for reasonably sized problems (n and m are < 100) there is the large potential for a gain in efficiency since we would be training a smaller network (with fewer weights θ to learn) with the OR-Net.

5.3.3. Coefficient Scaling

To ensure some uniformity across different problems, it is suggested to first scale the problem's coefficients between 0 and +1 or -1 and +1. This must be done in such a way that our problem maintains proportionality.

For attributes and bounds this is easy to describe. Simply divide all attributes that relate to a specific bound by the value of the bound and do this to the bound as well, see Equations (5.8) and (5.9). This results in attributes that are percentages of contribution towards the bound. Note, you may end up with values for $a_{i,j}$ that are not between 0 and 1, but that would be for specific problems containing offsetting attributes. If all attributes and bounds are positive, then they will be between 0 and 1 because any values larger than 1 can be omitted from the problem as they would violate the constraint set by that bound. The main point is that all attributes related to a specific bound remain proportional to each other and to the bound.

$$a'_{i,j} \leftarrow \frac{a_{i,j}}{b_i} \; \forall i, j \tag{5.8}$$

$$b'_i \leftarrow \frac{b_i}{b_i} = 1 \; \forall i \tag{5.9}$$

Many feasible options exist to scale the cost coefficients, but our suggestion is to scale them to the cost element with the largest magnitude as in Equation (5.10). This keeps all costs proportional to each other and scaled values between -1 and +1. If all cost coefficients are positive, the outcome scaled costs would be between 0 and +1.

$$c_j' \leftarrow \frac{c_j}{\max\limits_k |c_k|} \quad \forall j \tag{5.10}$$

5.3.4. Problem Uniqueness

Problem uniqueness might be confusing for a neural network to identify. For instance, let's say rows i and k were swapped in the attribute matrix. This means their corresponding bounds would also have to be swapped. Similar operations could be done with columns in the attribute matrix and the corresponding costs. In either of these swapping scenarios, the resulting ILP is exactly the same problem and would have the same optimal solution. There exist $n!$ total permutations for rows (number of constraints) and $m!$ for columns (number of decision variables). This results in $n!\,m!$ total equivalent representations of the same problem, even after scaling.

This creates a powerful training opportunity, as different representations of the same problem (for labeled examples where we already know the optimal solution) can be used for additional data samples to train the network. This is a nearly effortless method for implementing a high-quality form of data augmentation.

The challenge here, is that the problem might look completely different to the neural network since the flattened vector $[\mathbf{A}|\mathbf{b}|\mathbf{c}]^\top$, which represent the input features, might not look the same. We believe this may possibly result in the network forming unique weight structures in the OR-Layer that need to be common across certain relationships. The interesting part though, is that this should happen organically through training if the network is exposed to several different representations of the same problem.

5.3.5. Building the OR-layer in PyTorch

Feasible and efficient implementation of an OR-layer is necessary for the OR-Net's potential speed benefits to be realized. This comes with some challenges when working within the confines of an existing programming framework. The below paragraphs describe possible implementation concepts using PyTorch.

Iterations: Build and update individual connections through iterations. This ensures only the connections that we care about are being made. Because of the inability to take advantage of vectorization, this is useful for small toy problems and likely not suitable for our interests.

Masking Matrix: Create an equal sized 0-1 matrix to mask the matrix of a fully connected layer and convert it into an equivalent matrix of only the values we are interested in, where values we do not care about are all zeroed out (see Figure 5.6).

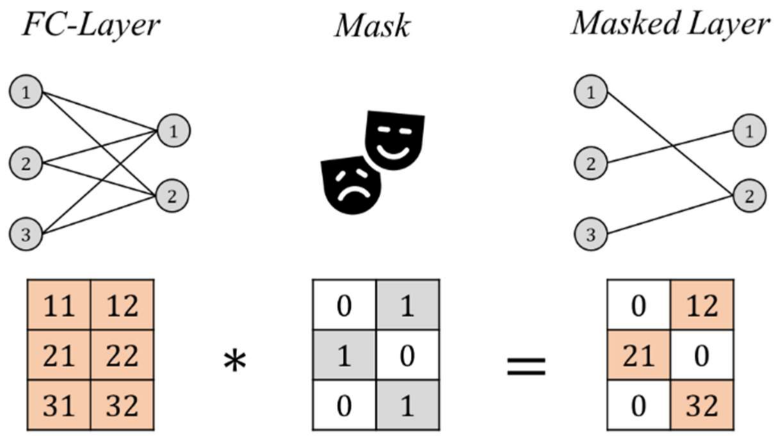


Figure 5.6 Diagram describing how the application of a masking matrix effectively creates a limited connectivity network layer.

With modern CPUs / GPUs, it's easier to zero-out particular weights of a large linear layer and do dense matrix operations, than to do iterations by keeping a linear list and performing those operations locally. One caveat is that masking requires element-wise multiplication of potentially large matrices, but that is nowhere near the complexity of matrix multiplication. Some of this may be alleviated by using Sparse Tensor operations in PyTorch, but that was not explored.

Selective Updating: Conceptually, if the masking matrix can be applied once upon initialization and can also be used to specify which values require gradients to be calculated, then the mask shouldn't need to be regularly applied. This cuts out a potentially unnecessary and long operation that could ruin the potential benefits of the sparsely connected layer. Unfortunately, PyTorch does not have any current options for the updating to be applied to individual edges and not just the entire layer by variables.

Selective Dilution: Dilution is another term for dropout but normally refers to reducing the number of connections by removing edges whereas dropout usually refers to removing nodes. This concept would be to see if a dropout/dilution method can be applied to the specific edges desired, rather than at random by some probability. Currently there are no known methods to accomplish this using PyTorch.

Table 5.2 Comparison of OR-layer Implementation Techniques

Technique	**Benefits**	**Drawbacks**
Iterations	Easiest to control and implement. Low memory requirements.	Unlikely to perform well for large problems
Masking Matrix	Easy to control and implement. Much faster than iteration.	Requires element-wise multiplication. Weights are zeroed out, but they still exist.
Selective Updating	Eliminates having to calculate gradients that would be zero anyways repetitively.	Not an option in PyTorch.
Selective Dilution	Eliminates the unwanted edges entirely.	Not an option in PyTorch.

Table 5.2 outlines the major benefits and drawbacks for each of the proposed implementation techniques. After considering each of these methods, we believed the masking matrix was the most straightforward option that allowed for scaling. The operations should be much faster than iterations, even for a very sparse mask matrix. A custom OR-layer module was developed in PyTorch and tested using the masking matrix. After this implementation was validated and optimized, performance comparisons of an OR-layer versus a FC-layer needed to be executed for ILP problems.

5.4. Experimental Results

5.4.1. Experimental Problem Setup

A custom OpenAI Gym (Brockman, et al., 2016) environment was created with the features listed in section 5.3 to explore the stochastic rolling time horizon variation of the mix and blend problem.

Mix and Blend Problems: These problems are very common in the processing industries where input materials are mixed or blended to create new output products. The diet problem is another popular variation of a mix and blend problem that is also common in research or academic literature. We will consider the binary integer program version of this problem, which is specialized version of the multi-dimensional knapsack problem (Freville, 2004).

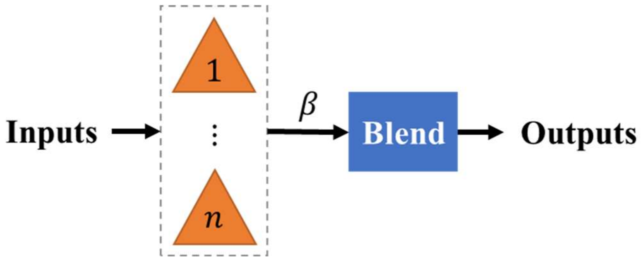


Figure 5.7 Process flow diagram of a standard mix and blend process

For the following experiments, we will learn to solve a version of the mix and blend where there are n total bins and we must select which bins to fully mix and blend together to make the output product. Our decision variables x_j are 1 when bin i is selected to be mixed and blended and 0 when not selected. There is a total of n decision variables (or actions) so the action space X grows very quickly. The resulting action space size is shown in Equation (5.11).

$$|X| = 2^n \tag{5.11}$$

In our problem definition, we are going to enforce that all orders are of a standard size and that exactly β number of bins should be selected. The way we enforce this exactness constraint is shown in Equation (5.12).

$$\sum_{i=0}^{n} x_i = \beta \tag{5.12}$$

Action Space Reduction: Due to the constraint in Equation (5.12), the action space can be reduced to omit any actions that would violate this constraint. The end effect is that the modified feasible action space size changes due to β. The problem can still become quite large, but it is reduced to only include n choose β possible actions, see Equation (5.13). This makes the problem a complexity of $\Theta(n^\beta)$ which is polynomial on average. For instance, a problem with $n = 20$ and $m = 5$ has $|X_\beta| = 15{,}504$ unique actions in the reduced action space versus the full action space size of $|X| = 2^{20} > 10^6$.

$$|X_\beta| = \binom{n}{\beta} = \frac{n!}{\beta!\,(n-\beta)!} \tag{5.13}$$

This effect may be noticeable depending on the size of the problem. If we compare Figure 5.6 (in section 5.3.2) to Figure 5.8 below ($\beta = 1$), even in this small example, that the reduction in action space size from $|X| = 4$ to $|X_\beta| = 2$ also reduces the size of the output layer of the neural network which should improve training performance.

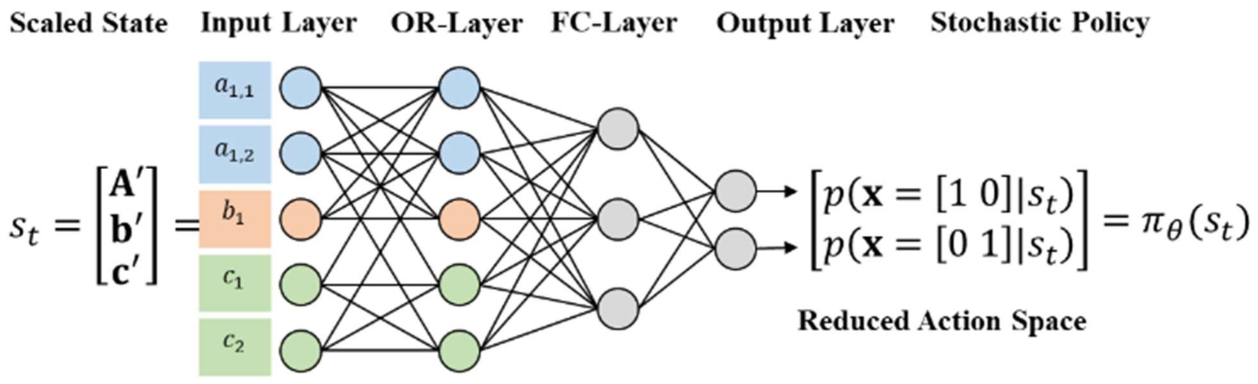

Scaled State
Input Layer
OR-Layer
FC-Layer
Output Layer
Stochastic Policy
$s_t = \begin{bmatrix}\mathbf{A}' \\ \mathbf{b}' \\ \mathbf{c}'\end{bmatrix} =$
$a_{1,1}$
$a_{1,2}$
b_1
c_1
c_2
$\begin{bmatrix} p(\mathbf{x} = [1\ 0] | s_t) \\ p(\mathbf{x} = [0\ 1] | s_t) \end{bmatrix} = \pi_\theta(s_t)$
Reduced Action Space

Figure 5.8 Example OR-Network architecture for a small ILP problem with a reduced feasible action space and smaller output layer.

Stochastic Rolling Time Horizon Problem: To make this problem a stochastic problem, we attempt to solve the mix and blend problem repeatedly across a rolling time horizon. There are 3 major sources of uncertainty that are introduced.

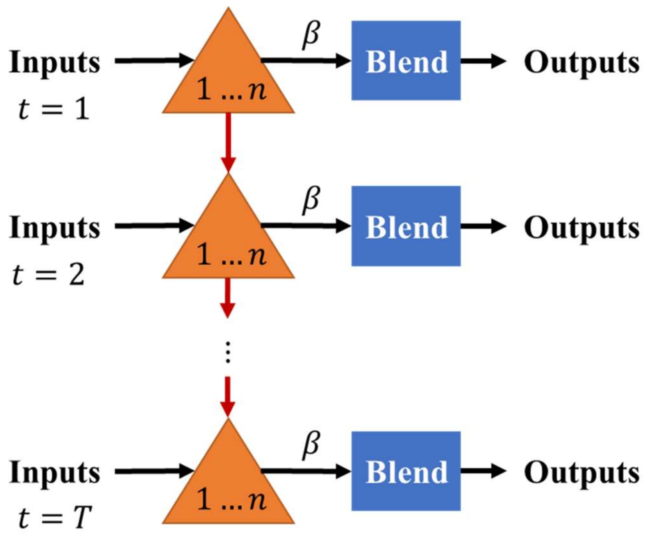


Figure 5.9 Flow of materials (black) and information (red) for a rolling time horizon problem

The first is that when a bin is selected, its contents are consumed in the current iteration to fulfill an order. This means that those contents must be replaced with new materials whose coefficients must be brought into the epoch's problem. This means we must introduce new attributes a_{ij} and costs c_j for the bins that were selected. These new coefficients are unknown but assumed to follow common distributions ξ in the Gym environment.

If $x_j^t = 1$ then for all i:

$$a_{i,j}^{t+1} \sim \xi_{a,i} \tag{5.14}$$

$$c_j^{t+1} \sim \xi_c \tag{5.15}$$

The second source of uncertainty is the bounds. We assume that the other requirements of the problem may have changed based on new order specifications.

$$b_j^{t+1} \sim \xi_{b,j} \tag{5.16}$$

Lastly, we will consider degradation of products overtime that are not selected. This was applied using a degradation factor δ, but capping the max attribute value $a_i^{\max}$ (which was set to 1 for our scaled attributes).

If $x_j^t = 0$ then for all i:

$$a_{i,j}^{t+1} = \min\{a_{i,j}^t (1+\delta), a_i^{\max}\} \tag{5.17}$$

The degradation feature makes this problem more challenging, even for traditional optimization methods such as stage-wise integer programming or heuristics as they will have difficulties quantifying the dynamic nature of the problem. Depending on the problem design, some methods may be myopic and fail to select locally sub-optimal solutions in the current state that will lead to better possible solutions in future states.

5.4.2. Results

Initially we tried testing this out on larger problems but ran into memory issues due to the excessive action space. This issue will be discussed in more detail in future work (section 6.2.3) as we address the challenge of complex actions and explain the motivation for direct policies.

Example Problem Details: The results in this section are based on an example mix and blend problem with $n = 10$ total bins where the decision maker must select $\beta = 2$ for each epoch t. There are $m = 6$ quality related constraints and a penalty of $\lambda = 10$ is applied for constraint violations. Attributes of bins' contents were degraded at a rate of $\delta = 10\%$ of their current values between each epoch if their bin was not selected. If their bin was selected, the product attributes and costs were replaced with new randomized product values. The bounds $\mathbf{b}_t$ of the problem (the desired quality metrics of the order) change with each epoch t.

Training: The FC-Net and OR-Net were trained with 180 batches (episodes) each and 128 epochs in each batch. The following hyper-parameters were used for the REINFORCE algorithm. A discount factor of $\gamma = 0.99$ was used for calculating the discounted rewards. A learning rate of $\eta = 0.01$ was used for SGD during batch training. This was repeated for 30 trials to generate the training curves seen in Figure 5.10. The top chart shows the mean reward earned at each batch and the bottom chart shows the standard deviation in the rewards earned at each batch.

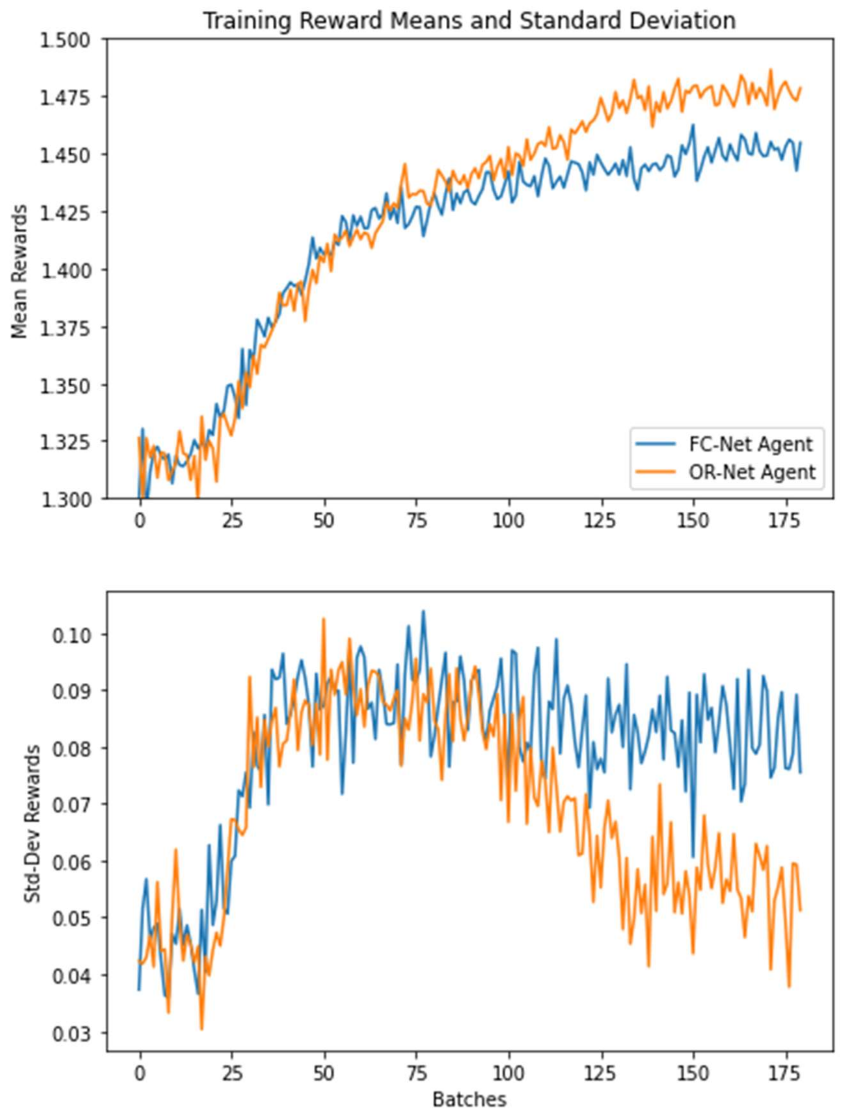


Figure 5.10 Training curves for the FC-Net and OR-Net showing convergence in 180 batches

The OR-Net performed very similarly to the fully connected network (FC-Net), slightly edging it out over the same training duration. The higher on average training performance we believe is due to better variance reduction. This may very likely be due to less noise being created from the OR-Net having fewer total weights to learn.

Testing: We tested 5 agents: the LP solver, a random decision maker, a greedy algorithm, the fully connected network (FC-Net) and the OR-Net. These were tested on a problem of the exact same size running 30 independent trials with 250 sequential epochs each. The 95% confidence intervals for the discounted total rewards Q_{250} along with average solve time and gap between the LP Solver's performance can be seen in Table III.

$$Q_{250} = \sum_{t=1}^{250} \gamma^t r_t \tag{5.18}$$

Table 5.3 Testing results for all agents after 30 trials of 250 epochs

Agent	**CI of Sum of Discounted Rewards Q_{250}**	**Avg Solve Time**	**Avg Gap**
LP Solver	(142.54, 135.77)	6.73 seconds	-
Random	(127.98, 114.44)	0.17 seconds	17.94
Greedy	(137.34, 126.74)	0.20 seconds	7.11
FC-Net	(145.56, 123.73)	0.30 seconds	4.50
OR-Net	(141.41, 131.47)	0.29 seconds	2.71

Although the DRL agents did not consistently receive rewards as high as the stage-wise LP solver, they did better on average than the greedy agent and the random decision maker. They also showed improvement through their training curves as seen in Figure 5.10 which indicates that with finer hyper-parameter tuning or more sophisticated algorithms these NN architectures could more closely approximate or maybe even exceed the linear programming solver's performance depending on the stochastic problem's design. For a problem of this size, the differences in the solve times were nonconsequential, but larger problems might see benefits using the neural network agents versus the LP solver. Based on their 95% confidence intervals for the discounted sum of rewards Q_{250}, the OR-Net and the LP solver were the only two agents that showed they performed statistically better than the random decision maker.

5.5. Conclusions

This chapter’s primary achievements were to introduce, build, and test a new NN architecture called OR-Net. The distinguishing component of this new network is the OR-layer, which only builds connections based on the orthogonal relationships that exist between an ILP’s coefficients. Proper evaluation of this new architecture relies on building and maintaining the OR-layer, which was successfully implemented using a masking matrix in PyTorch. This chapter also outlined the groundwork for other important features such as coefficient scaling and problem uniqueness which enabled us to create a generalized framework for testing OR-Net and other DRL agents on ILP problem environments.

We then built a custom OpenAI Gym environment for a stochastic rolling time horizon mix and blend problem with product quality degradation to test out the OR-Net vs other agents. We discovered that OR-Net and DRL show promise on these problems with reduced variance in its performance versus a fully connected network. It also noticeably outperformed greedy and randomized agents. By leveraging the strengths and flexibility of DRL for applications to stochastic problems where traditional methods in combinatorial optimization often underperform, OR-Net may offer benefits that would be highly valuable in many real-world applications.

CHAPTER 6. CONCLUSIONS AND FUTURE WORK

This dissertation provided a general framework for modeling dynamic problems dealing with uncertainty in agri-food supply chains. A general yet flexible model using Markov decision processes was formulated to track the supply chain step and quality level of the products at each step. To leverage modern tools and data collection practices, this research also integrated data mining techniques to learn the supply chain dynamics from data to make these models data driven. Studies were also completed analyzing computation time challenges that arise in these integrated data mining and decision-making models. Novel techniques that leverage machine learning and Big Data mining were introduced. This research then acknowledged that many optimization problems in agri-food supply chains and other industries rely on solving sequential integer programs, but these models struggle with accounting for the underlying sources of uncertainty. A novel concept referred to as OR-Net was introduced to leverage the power and promise of deep reinforcement learning specifically for the stochastic variants of these problems. This idea was then tested on the mix and blend problem, which is commonly found in agri-food supply chains such as those in grain, oilseeds, and other granular agricultural products' export processes.

A summary of the main contributions of this research is provided below in section 6.1. In section 6.2 we conclude this thesis with a discussion of ideas for future work including real-world applications of this research and its extensions.

6.1. Conclusions

In Chapter 3, the generalized model for optimizing dynamic food supply chains was extended to account for the stochastic behavior of the outcomes given decisions made and actions taken. This model considers the effect these individual decisions (including supplier selection) may have on the product's final quality. A basic food supply chain was modeled dynamically with the decisions and actions at each step as a Markov decision process. Sensitivity analysis was then performed to study the impact of threshold values and reward function on the optimal policy selection. This reveals a strategy shift for the optimal policy from selecting a best weighted average with lower restrictions towards becoming aspirational as the system became more restricted by the quality constraints.

It was noted that the action space grows exponentially. Therefore, the final solution technique may need to be modified as larger problems that are more complex may not be able to converge to a final solution in an efficient timeframe using Policy Iteration. Reinforcement learning is proposed to solve these more complex and general problems.

Chapter 4 covered the integration of data mining techniques to determine the transition probabilities for the generalized MDP model formulated in Chapter 3. This chapter also addresses a few complexity issues with modeling dynamic food supply chains to account for uncertainties present and the vast array of potential decisions and possible quality states. The general mathematical model we formulated for agri-food supply chains was a Markov decision that considers the stochastic effect these decisions have on a product's quality. Since the transition model is typically unknown in practice, it is proposed to determine these probabilities from historical data. The concept of integrating a Bayes net is first introduced, and we believe this is sufficient for smaller problems. However, it is recognized that as a problem's complexity increases, the states and potential actions that need to be considered grow exponentially. A novel approach that involves Big Data mining tools and techniques is then presented as a possible improvement. We formalize a procedure to determine values for our transition matrix using this new method. Performance tests show how the running times improve when using this Big Data technique versus standard methods for complex problems. Lastly, we conduct experiments showing how the optimal policy changes as the quality threshold for rejection becomes more restrictive which identifies when it is either valuable or unnecessary to invest in higher quality actions.

In chapter 5, the strengths of approximation techniques like deep reinforcement learning were investigated for their suitability for stochastic integer linear programs, such as the mix and blend problem commonly found in agri-food supply chains. This chapter's primary achievements were to introduce, build, and test a new NN architecture called OR-Net. The distinguishing component of this new network is the OR-layer, which only builds connections based on the orthogonal relationships that exist between an ILP's coefficients. Proper evaluation of this new architecture relies on building and maintaining the OR-layer, which was successfully implemented using a masking matrix in PyTorch. This chapter also outlined the groundwork for other important features such as coefficient scaling and problem uniqueness which enabled us to create a generalized framework for testing OR-Net and other DRL agents on ILP problem environments.

To test this approach, we then built a custom OpenAI Gym simulation environment for a stochastic rolling time horizon mix and blend problem with product quality degradation to evaluate the performance of OR-Net vs other agents. We discovered that an OR-Net trained by DRL shows promise on these problems with reduced variance in its performance versus a fully connected network. OR-Net also noticeably outperformed greedy and randomized agents. By leveraging the strengths and flexibility of DRL for applications to stochastic problems where traditional methods in combinatorial optimization often underperform, OR-Net may offer benefits that would be highly valuable in many real-world applications.

In summary, this research provides scientists and practitioners new models and techniques that can be customized for problems in their own research or industries. These models are applicable to problems associated with improving or controlling quality for output products in agri-food production. This reduces the impact that inefficient supply chains have on precious resources such as land, water, energy, and human capital. Supply chain managers and policy makers will also benefit from knowledge of how these models can be fueled through the integration of scalable data mining techniques to take advantage of their historical and future process data. Lastly, tools such as OR-Net when paired with emerging technologies like deep reinforcement learning may eventually help global supply chains approach optimality and reduce their inadvertent contributions to food loss. This can improve food security and minimize the negative environmental and societal impacts of agri-food supply chains.

6.2. Future Work

In the following sections, future research opportunities will be discussed as they related to the problems explored in this thesis.

6.2.1. Modeling Additional Sources of Uncertainty – Partial Observability

Future opportunities that might exist within this body of work might include modeling market conditions, as the price of final products or inputs can easily change within the timeframe of a full supply chain production cycle. This could require converting this general MDP model to a Partially Observable Markov decision process (POMDP) because the factors that influence markets are not as easily recognizable.

6.2.2. Validating Models with Real World Data – Swine Nutrition Optimization

The models introduced in Chapter 3 and data-driven techniques mentioned in Chapter 4 were designed to be both extremely flexible and relevant to many problems in agri-food supply chain processes. To demonstrate these capabilities, we will discuss an example of how to apply this to a swine production and optimal nutrition problem.

The life cycle of a market pig is approximately 6 to 7 months long (National Pork Board, 2021). They are fed and nurtured from 2-3 lbs. at birth to up to 280 lbs. when they go to market for slaughter. Pork is the most widely produced meat across the world. The US alone imported approximately 6.5 million metric tons of pork in 2021, even with significant local production capabilities (Kenner, Jiang, & Russell, 2022).

Table 6.1 Production life cycle of swine

Step	**Pregnancy to Birth**	**Farrowing**	**Nursery**	**Growing and Finishing**	**Market**
Duration	16 weeks	3 weeks	6-7 weeks	16-17 weeks	
Diet		Milk	Corn or Soybean Meal		
Amount			1.4-4 lbs.	6-10 lbs.	

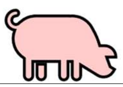
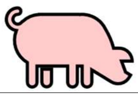
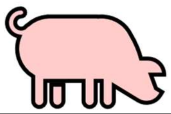
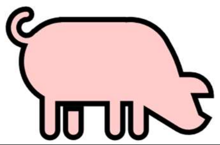
				…	
Age	0	3 weeks	9-11 weeks	…	25-28 weeks
Weight	2-3 lbs.	12-15 lbs.	50-60 lbs.	…	280 lbs.

Table 6.1 shows that swine production has a set of identifiable and separable steps, each having different inputs and target outcomes. In this example, the life cycle of a market pig can be modeled as a set of sequential steps $k \in K$. There are observable quality metrics y_k at each step such as weight and general health. In each step k, actions $\mathbf{x}_k \in X_k$ could include the diet makeup and amount of feed, nutritional supplements such as vitamins and minerals, and the controllable

features of the environment (e.g., after weaning, piglets are moved to a nursery with specialized temperature controls and ventilation). In the initial steps, the first decision $\mathbf{x}_1$ might need to select which breed of pig to purchase and the supplier to get them from or whether to raise your own piglets from birth. These actions $\mathbf{x}_k$ at every step k would come at varying costs, some of which might depend on the unique operational setup or current market conditions. The decision maker must avoid making myopic decisions because the major contribution to the value function is the delayed rewards that depend on our final state (i.e., when the pig goes to market).

This problem includes all the necessary pieces to be modeled and solved using the approaches outlined in Chapter 3. The transition model (which would describe the connections between growth and health related states between steps given certain actions were taken) can be determined using either of the approaches discussed in Chapter 4, with the recommended approach depending of course on the quantity and complexity of data available.

6.2.3. Further Refining OR-Nets for Better Performance

The body of research of trying to solve combinatorial optimization problems (COPs) including integer linear programs (ILPs) using reinforcement learning (RL) and Deep RL is just beginning. Ideas like OR-Nets often require slight refinements before their true potential is observed and they become major contributions in their field. In this section, we will discuss a few of the ideas that we believe could deliver even stronger results for OR-Nets.

More advanced algorithms such as actor-critic methods including A2C (Sutton & Barto, 2018) and A3C (Mnih, et al., 2016), proximal policy optimization (PPO) (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017), or advanced hyper-parameter tuning would likely help ideas like the OR-Net close the gap or potentially outperform traditional methods like stage-wise LP or meta-heuristics depending on the complexity and nature of the problem. Also, tools and technology that allow parallelization and improved speedup of the training process may allow for different combinations of epochs and batches to be tested efficiently. Another common training process is to utilize a form of logic in the early stages of training, such as starting with a greedy decision maker for pre-training your network. This may help to expedite the training process, allowing for more time to explore alternative neural network architectures or algorithms.

Perhaps the most unique and challenging aspect of trying to approach solving combinatorial optimization problems (COPs) using RL or DRL is dealing with complex actions $\mathbf{x}$ where n elements (individual decision variables) are being determined simultaneously together instead iteratively or independently. To enable this, reinforcement learning methods such as Q-learning, DQN, and policy gradients depend on the full action space X (or a subset of identified feasible actions) being searchable and contractable. COPs in general have action spaces that often grow far too large for this to be possible, so direct-action policies may be an alternative way forward. This is where the individual actions' x_j have policies $p\left(x_j = 1 | s_t\right)$ that are determined directly. This would allow larger problems to be explored without exceeding memory limitations due to the output layer not needing to be the same size as the action space $|X|$, but can become the size of unique action elements n. We did some preliminary testing with this, but we found it to be too unstable until more research is completed to understand if such an approach is feasible.

An idea we haven't explored yet would be to include a new layer in the neural network to be used for deciphering the final decisions into a synergistic policy, see Figure 6.1 where this is shown as a recurrent layer. Simple feed forward NNs may not offer the best solutions for such complex decisions, so testing with a recurrent neural network (RNN) like long short-term memory networks (Hochreiter & Schmidhuber, 1997) could be beneficial. Note that the output in Figure 6.1 is a direct policy for each individual decision variable element. This would allow the NN output layer to have only n elements and does not require a full representation of all elements in the searchable action space $|X|$ which drastically reduces the size of the NN. Ultimately, bridging the gap between unique action spaces and their individual elements could enable large complexity reductions and help move this area of research forward.

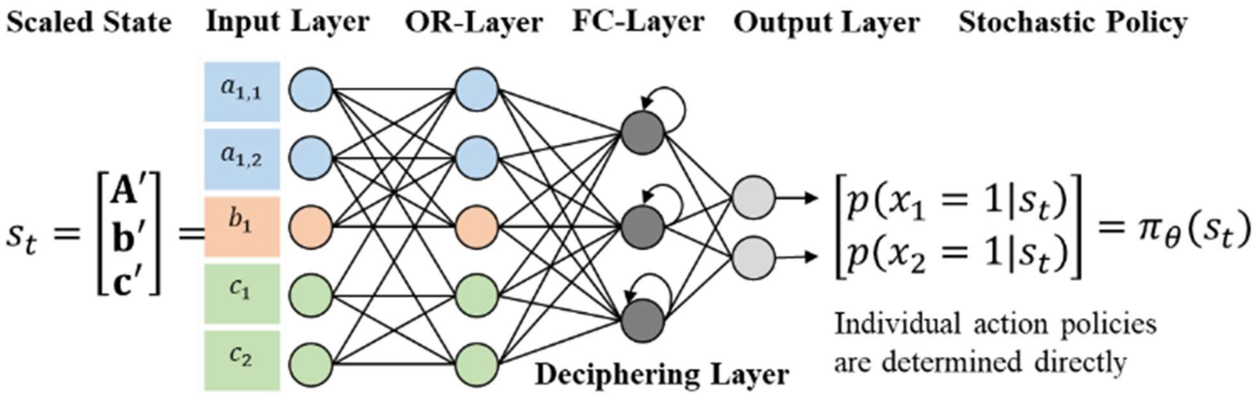

Scaled State
Input Layer
OR-Layer
FC-Layer
Output Layer
Stochastic Policy
$s_t = \begin{bmatrix} \mathbf{A}' \\ \mathbf{b}' \\ \mathbf{c}' \end{bmatrix} =$
$a_{1,1}$
$a_{1,2}$
b_1
c_1
c_2
$\begin{bmatrix} p(x_1 = 1|s_t) \\ p(x_2 = 1|s_t) \end{bmatrix} = \pi_\theta(s_t)$
Individual action policies are determined directly
Deciphering Layer

Figure 6.1 Example OR-Net architecture for a small ILP problem with a recurrent layer.

6.2.4. Expanding the Mix and Blend Problem to cover both space and time

This idea considers a large grain exporting operation that consists of several sequential stages of blending and mixing processes being performed across a rolling time horizon. This results in a very complex problem to solve, but these multi-step operations usually do not have a single decision maker controlling every action. See Figure 6.2 for an example of this process, which consists of several sequential mix and blend steps.

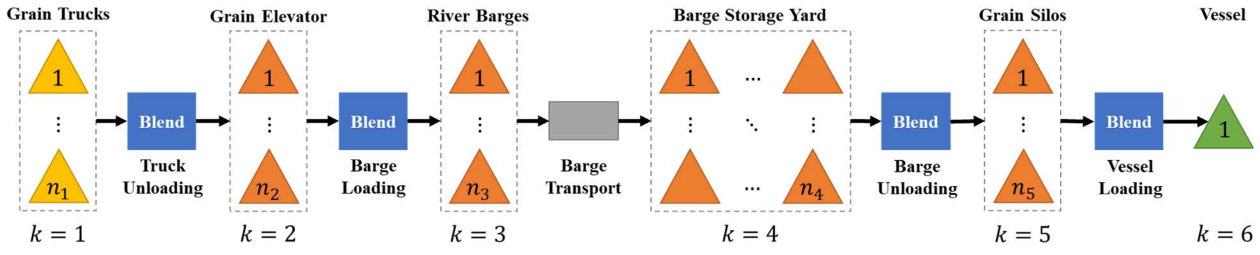


Figure 6.2 Example process flow diagram of an end-to-end grain exporting operation

In this problem, the previously developed stochastic modeling and optimization methods are applied to this impactful industry process. This model is built from principles detailed in earlier research tasks and expanded to consider both multiple-steps and a rolling time horizon. The sequential nature of this problem is both across process steps and time periods.

One concept to accomplish this would be by defining process modules and chaining them together to build more complicated processes. The changes in states can be more easily captured and predicted without needing an impossibly large transition model. See figure 9 for diagrams of these ideas.

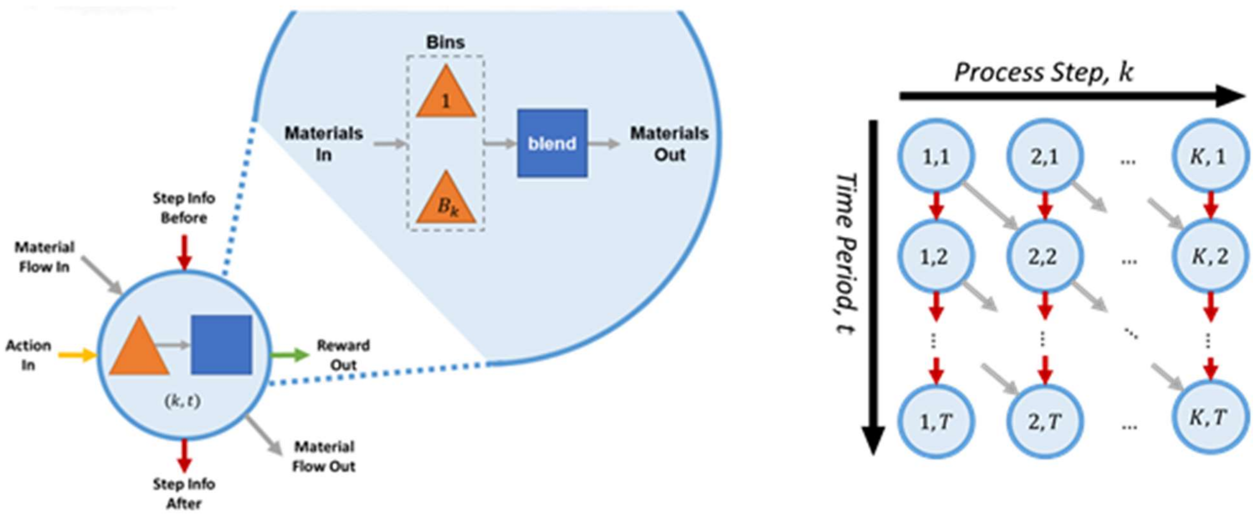


Figure 6.3 Diagram of a process module, and a chain of process modules describing an operation

The mix and blend problem we approached in Chapter 5 is expanded and tested over a series of subsequent process steps, where outputs of earlier steps become the inputs of later steps. Spoilage may also need to be considered further because quality degradation is a major issue in grain exporting operations that often goes unrecognized until later in the supply chain due to extended periods of storage, such as those experienced during transportation on river barges.

The initial idea to test this out would be to first create individual neural networks to solve the problems at each stage locally. After these decentralized (step specific) agents have been trained, their knowledge is then transferred to a centralized supply chain director agent. This would allow locally strong policies to be created at first. These policies would then be incrementally improved and integrated as their influence in the entire end-to-end supply chain is learned and understood by the higher-level agent. An alternative multi-agent approach would be for the centralized director to determine incentives for each agent at each step-time epoch.

This concludes the discussion on future work extensions and completes this thesis.

References

Aday, S., & Aday, M. S. (2020). Impact of COVID-19 on the food supply chain. *Food Quality and Safety*, 167–180.

Agrawal, R., Imielinski, T., & Swami, A. (1993). Mining Association Rules between Sets of Items in Large Databases. *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data. 22*, pp. 207-216. Washington D.C.: Association for Computing Machinery. doi:11.1145/170036.170072

Ahumada, O., & Villalobos, J. (2009). Application of planning models in the supply chain of agricultural products: a review. *European Journal of Operational Research*, 1–20.

Ahumada, O., & Villalobos, J. (2011). A tactical model for planning the production and distribution of fresh produce. *Annals of Operations Research*, 339–358.

Ahumada, O., & Villalobos, J. (2011). Operational model for planning the harvest and distribution of perishable agricultural products. *International Journal of Production Economics*, 677-687.

Ahumada, O., Villalobos, J., & Mason, A. (2012). Tactical planning of the production and distribution of fresh agricultural products under uncertainty. *Agricultural Systems*, 17-26.

Akkerman, R., Farahani, P., & Grunow, M. (2010). Quality, safety and sustainability in food distribution: a review of quantitative operations management approaches and challenges. *OR Spectrum, 32*(4), 863-904.

Anonymous. (n.d.). *Frequent Pattern Mining*. Retrieved from Apache Spark 2.4.4 Docs: https://spark.apache.org/docs/latest/ml-frequent-pattern-mining.html

Bellman, R. (1955). Functional Equations in the Theory of Dynamic Programming. V. Positivity and Quasi-Linearity. *Proceedings of the National Academy of Sciences of the United States of America*, 743–746.

Bertsekas, D., & Tsitsiklis, J. (1996). *Neuro-Dynamic Programming.* Athena Scientific.

Bishop, C. (2006). *Pattern Recognition and Machine Learning* (1st ed.). New York: Springer.

Board for International Food and Agricultural Development (BIFAD); International Food Policy Research Institute (IFPRI); and Association of Public and Land-Grant Universities (APLU). (2019). *How the United States benefits from agricultural and food security investments in developing countries.* Washington, DC: International Food Policy Research Institute (IFPRI).

Borodin, V., Bourtembourg, J., Hnaien, F., & Labadie, N. (2015). A multi-step rolled forward chance-constrained model and a proactive dynamic approach for the wheat crop quality control problem. *European Journal of Operational Research*, 631–640.

Borodin, V., Bourtembourg, J., Hnaien, F., & Labadie, N. (2016). Handling uncertainty in agricultural supply chain management: A state of the art. *European Journal of Operational Research*, 348-359.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). OpenAI Gym. *arXiv preprint arXiv:1606.01540v1.*

Chvátal, V. (1973). Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Math, 4*, 305-337.

Crowder, H., Johnson, E., & Padberg, M. (1983). Solving large-scale zero-one linear programming problem. *Operations Research, 31*, 803-834.

Dai, H., Khalil, E. B., Zhang, Y., Dilkina, B., & Song, L. (2018). Learning Combinatorial Optimization Algorithms over Graphs. *arXiv preprint arXiv:1704.01665v4.*

de Farias, D. P., & Van Roy, B. (2000). On the existence of fixed points for approximate value iteration and temporal-difference learning. *Journal of Optimization Theory and Applications, 105*, 589–608.

Dou, Z., Ferguson, J., Galligan, D., Kelly, A., Finn, S., & Giegengack, R. (2016). Assessing U.S. food wastage and opportunities for reduction. *Global Food Security*, 19-26.

Drexl, A. (1988). A simulated annealing approach to the multiconstraint zero-one knapsack problem. *Computing, 40*, 1-8.

FAO. (2013). *Food Wastage Footprint: Impacts on Natural Resources.* Rome.

Farias, D. P., & Roy, B. V. (2003). The Linear Programming Approach to Approximate Dynamic Programming. *Operations Research, 51*(6), 850-865.

Freville, A. (2004). The multidimensional 0–1 knapsack problem: An overview. *European Journal of Operational Research , 155*, 1–21.

Gaukler, G., Ketzenberg, M., & Salin, V. (2017). Establishing dynamic expiration dates for perishables: An application of RFID and sensor technology. *International Journal of Production Economics, 193*, 617-632.

Georgiadis, P., Vlachos, D., & Iakovou, E. (2005). A system dynamics modeling framework for the strategic supply chain management of food chains. *Journal of Food Engineering*, 351-364.

Gigler, J., Hendrix, E., Heesen, R., Hazelkamp, V., & Meerdink, G. (2002). On optimisation of agri chains by dynamic programming. *European Journal of Operational Research*, 613-625.

Gomory, R. E. (1658). Outline of an algorithm for integer solutions to linear programs. *Bull. Amer. Math. Soc., 64*, 275-278.

Hahsler, M., Grun, B., Hornik, K., & Buchta, C. (2005). Introduction to arules – A computational environment for mining association rules and frequent item sets. *Journal of Statistical Software*, 1-25.

Han, J., Pei, J., & Yin, Y. (2000). Mining frequent patterns without candidate generation. *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, (pp. 1-12). Dallas, TX.

Hasselt, H. v., Guez, A., & Silver, D. (2016). Deep Reinforcement Learning with Double Q-learning. *AAAI.*

Hembecker, F., Lopes, H. S., & Jr., W. G. (2007). Particle Swarm Optimization for the Multidimensional Knapsack Problem. *Adaptive and Natural Computing Algorithms.* Warsaw, Poland.

Heumesser, C., Fuss, S., Szolgayová, J., Strauss, F., & Schmid, E. (2012). Investment in irrigation systems under precipitation uncertainty. *Water Resource Management*, 3113-3137.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation, 9*, 1735-1780.

Kappelman, A. C., & Sinha, A. K. (2021, February 1). Optimal control in dynamic food supply chains using big data. *Computers & Operations Research, 126*.

Kappelman, A., & Sinha, A. (2019). Optimal Control in Dynamic Food Supply Chain under Quality Constraints. *Proceedings of the 2019 IISE Annual Conference*, (pp. 1749-1754). Orlando, FL.

Karp, R. M. (1972). Reducibility among Combinatorial Problems. In M. R.E., T. J.W., & B. J.D. (Eds.), *Complexity of Computer Computations. The IBM Research Symposia Series* (pp. 85-103). Boston, MA: Springer. doi:10.1007/978-3-540-68279-0_8

Kenner, B., Jiang, H., & Russell, D. (2022). *Outlook for U.S. Agricultural Trade: August 2022, AES-121.* USDA, Economic Research Service and USDA, Foreign Agricultural Service.

Kool, W., Hoof, H. v., & Welling, M. (2019). Attention, Learn to Solve Routing Problems! *arXiv preprint arXiv: 1803.08475v3.*

Kopanos, G., Puigjaner, L., & Georgiadis, M. (2012). Simultaneous production and logistics operations planning in semicontinuous food industries. *Omega*, 634-650.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet Classification with Deep Convolutional Neural Networks. *Communications of the ACM, 60*(6), 84-90.

Land, A. H., & Doig, A. G. (1960). An Automatic Method of Solving Discrete Programming Problems. *Econometrica, 28*(3), 497-520.

Li, H., Wang, Y., Zhang, D., Zhang, M., & Chang, E. (2008). PFP: Parallel FP-Growth for Query Recommendation. *Proceedings of the 2008 ACM Conference on Recommender Systems*, (pp. 107-114). Lausanne, Switzerland.

Li, Z., Liu, G., Liu, L., Lai, X., & Xu, G. (2017). IoT-based tracking and tracing platform for prepackaged food supply chain. *Industrial Management & Data Systems*, 1906-1916.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning, 8*, 293-321.

Manchanda, S., Mittal, A., Dhawan, A., Medya, S., Ranu, S., & Singh, A. (2020). Learning Heuristics over Large Graphs via Deep Reinforcement Learning. *arXiv preprint arXiv: 1903.03332v3.*

Manne, A. S. (1960). Linear Programming and Sequential Decisions. *Management Science, 6*(3), 259-267.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Harley, T., Lillicrap, T. P., . . . Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. *International Conference on Machine Learning (ICML).*

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., . . . Hassabis, D. (2015). Human level control through deep reinforcement learning. *Nature, 518*, 529-533.

Nagurney, A., Besik, D., & Yu, M. (2018). Dynamics of Quality as a Strategic Variable in Complex Food Supply Chain Network Competition: The Case of Fresh Produce. *Chaos*, 043124.1-043124.16.

National Pork Board. (2021). *Life Cycle of a Market Pig*. Retrieved from https://porkcheckoff.org/pork-branding/facts-statistics/life-cycle-of-a-market-pig/

Neapolitan, R. (2004). *Learning Bayesian Networks* (1st ed.). New Jersey: Prentice Hall.

Olsson, A. (2004). Temperature controlled supply chains call for improved knowledge and shared responsibility. *Proceedings of the 16th Annual Conference Nofoma* (pp. 569-582). Linköping, Sweden: Logistics Management.

Oroojlooyjadid, A., Nazari, M. R., Snyder, L., & Takáč, M. (2017). A Deep Q-Network for the Beer Game: A Deep Reinforcement Learning algorithm to Solve Inventory Optimization Problems. *Neural Information Processing Systems (NIPS), Deep Reinforcement Learning Symposium 2017*.

Powell, W. B. (2011). *Approximate Dynamic Programming: Solving the Curses of Dimensionality.* Hoboken, New Jersey: John Wiley & Sons, Inc.

Puterman, M. (2005). *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Hoboken, New Jersey: Wiley & Sons.

Roth, D. (1996). On the hardness of approximate reasoning. *Artificial Intelligence*, 273-302.

Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2016). Prioritized Experience Replay. *CoRR, abs/1511.05952*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347v2.*

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., . . . L. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature, 529*, 484-489.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., . . . Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science, 362*, 1140-1144.

Stygar, A. H., Kristensen, A. R., & Makulska, J. (2014). Optimal management of replacement heifers in a beef herd: model for simultaneous optimization of rearing and breeding decisions. *Journal of Animal Science, 92*, 3636-3649.

Sumanatra, J., & Ramirez, J. (1997). Optimal stochastic multi-crop seasonal and intraseasonal irrigation control. *Journal of Water Resources Planning and Management*, 39-48.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning.* Cambridge, MA: The MIT Press.

Tao, Q., Gu, C., Wang, Z., Rocchio, J., Hu, W., & Yu, X. (2018). Big Data Driven Agricultural Products Supply Chain Management: A Trustworthy Scheduling Optimization Approach. *IEEE Access, 6*, 49990-50002.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). Graph Attention Networks. *arXiv preprint arXiv:1710.10903v3.*

Vinyals, O., Fortunato, M., & Jaitly, N. (2015). Pointer Networks. *arXiv preprint arXiv:1506.03134v2.*

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., & Freitas, N. (2016). Dueling Network Architectures for Deep Reinforcement Learning. *Proceedings of the 33rd International Conference on Machine.* New York, NY.

Watkins, C. J., & Dayan, P. (1992). Q-Learning. *Machine Learning, 8*, 279-292.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning, 8*(3), 229-256.

Yan, B., Wu, X.-h., & Zhang, B. Y.-w. (2017). Three-level supply chain coordination of fresh agricultural products in the Internet of Things. *Industrial Management & Data Systems, 117*(9), 1842-1865.

Zhang, H. (2004). The Optimality of Naive Bayes. *Proceedings of the 17th International Florida Artificial Intelligence Research Society Conference* (pp. 562-567). Miami Beach, FL: The AAAI Press.

Zhang, K., Yang, Z., & Basar, T. (2019). *Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms.*