Design and Performance Prediction for Supply Chain Systems with Graphical Structures

by

Shuyu Chen

Department of Business Administration
Duke University

Date: _____

Approved:

_____

Jing-Sheng Song, Co-supervisor

_____

Yehua Wei, Co-supervisor

_____

Bora Keskin

_____

Can Zhang

Dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy
in the Department of Business Administration
in the Graduate School of
Duke University

2022

ABSTRACT

Design and Performance Prediction for Supply Chain Systems with Graphical Structures

by

Shuyu Chen

Department of Business Administration
Duke University

Date: _____

Approved:

_____

Jing-Sheng Song, Co-supervisor

_____

Yehua Wei, Co-supervisor

_____

Bora Keskin

_____

Can Zhang

_____

An abstract of a dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Business Administration in the Graduate School of Duke University

2022

Abstract

This dissertation studies the optimal design and performance evaluation of large-scale supply chain systems with graphical structures such as assemble-to-order (ATO) systems and e-commerce fulfillment networks. It consists of three essays.

The first essay studies the design of effective operational policies in the assemble-to-order (ATO) systems. We consider continuous-review ATO systems with general bills of materials (BOM) and general leadtimes. First, we characterize the asymptotically optimal policy for the M-system. The policy consists of a periodic review priority (PRP) allocation rule and a coordinated base-stock (CBS) replenishment policy. We then construct heuristic policies using insights from the asymptotically optimal policy. In particular, we adopt the PRP allocation rule and develop a decomposition approach for inventory replenishment. This approach decomposes a general system into a set of assembly subsystems and constructs a linear program to compute the optimal policy parameters. However, both the CBS and the assembly decomposition approach are limited to simple systems. We then consider a second approach, which decomposes a system into a set of distribution subsystems and each subsystem has a straightforward optimal solution, which is similar to the newsvendor problem. Finally, in a numeral test, we find that the assembly decomposition is very effective but computationally expensive and thus only good for small-scale systems; the distribution decomposition performs as effective as the optimal independent base-stock (IBS) policy, but is highly scalable than finding the optimal IBS policy for large-scale systems.

The second essay focuses on optimizing the operational decision at one layer of the supply chain network when some operational decisions at another layer are unknown to the decision-maker. More specifically, we consider the transportation network

design problem for the e-commerce marketplace. A salient feature in this problem is decentralized decision-making. While the middle-mile manager decides the network configuration on a weekly or bi-weekly basis, the real-time flows of millions of packages on any given network configuration (which we call the *flow response*) are controlled by a fulfillment policy employed by a different decision entity. Thus, we face a fixed-cost network design problem with *unknown flow response*. To meet this challenge, we first develop a predictive model for the unknown response leveraging observed shipment data and machine learning techniques. Apart from the most natural network-level predictive model, we find that the more parsimonious destination-level and arc-level predictive models are more effective. We then embed the predictive model to the original network design problem and characterize this transformed problem as a c-supermodular minimization problem. We develop a linear-time algorithm with an approximation guarantee that depends on c. We demonstrate that this algorithm is scalable and effective in a numerical study.

The third essay investigates how to use the Graph Neural Network (GNN) model to predict the operational performances of supply chain networks. GNN is a newly developed machine learning tool to leverage the graphical structure information. It has demonstrated good prediction accuracy in various contexts, including social, bioinformatics and citation networks. Surprisingly, GNNs have not received much attention in supply chain systems despite the fact that many systems exhibit a graphical structure, such as assemble-to-order systems and process flexibility networks. To the best of our knowledge, we are the first to explore the application of GNN to supply chain problems. As operational performances of the entire network can often be decomposed into node-level or edge-level performances, we study the decompose-then prediction approach. We find that while the existing GNN model can generate reasonable node-level predictions, special tailoring is needed for edge-level predictions

of supply chain networks. A key contribution of our research is to develop a novel graph transformation approach, which allows an edge to learn from its neighborhood edges. Tested on different synthetic datasets from two different supply chain systems, we implement the GNN model with our proposed graph transformation and several benchmark methods, including an existing GNN model and the traditional machine learning methods, such as the convolutional neural network and random forest. The results indicate that our approach significantly outperforms the benchmarks in edge-level prediction. We also observe the importance of utilizing the graphical structure and edge directions. Our comparison reveals that it is beneficial to follow the decompose-then prediction approach, instead of the direct graph-level prediction commonly used in other applications.

Contents

List of Tables

List of Figures

Acknowledgements

I would like to express my deepest gratitude to my advisor, Professor Jeannette Song, for her tremendous support and detailed guidance. Her rigorous thinking and invaluable experience have taught me to become an independent researcher. I cannot complete my Ph.D. studies with her great support. I would also like to extend my deepest gratitude to my co-advisor, Professor Yehua Wei. His excellent ideas on research have encouraged me to explore new ideas and learn new knowledge. I would also like to thank my dissertation committee members, Professor Bora Keskin and Professor Can Zhang, for their valuable suggestions to my research.

Moreover, I want to thank Professor Lijian Lu at the Hong Kong University of Science and Technology, for being a great support for my Ph.D. studies. I must also thank Professor Hanqin Zhang at the National University of Singapore. He has provided me with many useful suggestions for my research paper. I am thankful to other faculty members in my program, Professor Robert Swinney, Professor Fernando Bernstein, and Professor Kevin Shang, for their guidance.

I am grateful to the members of the Ph.D. program at Fuqua, particularly Chen Chen and Yuan-Mao Kao, for their support during these years.

Finally, I would like to thank my parents for their support and encouragement throughout my study.

Chapter 1

Introduction

In today's fast-developing digital economy, new and complex supply chain problems are constantly emerging. Manufacturers and service providers now interact with users repeatedly and need to make operational decisions frequently based on data collected at a rapid pace. While many of these supply chain networks have graphical structures and can be formulated as large-scale and multi-stage stochastic optimization models when the model parameters are readily obtainable, finding the solutions to these models is computationally impossible due to the dimensionality of the problem. Even when such problems can be solved, practitioners often find the resulting optimal strategies hard to implement due to their complex structure. Therefore, there is a growing need for scalable approximation methods that operate efficiently while still being simple to implement and interpret. Thanks to today's data-rich world, one can also take an alternative data-driven approach which can predict the performance of the supply chain networks of a given operational mechanism without knowing all details of the mechanisms and the demand characteristics. The questions to answer are how to make the prediction, especially how to utilize the graphical structure to improve the prediction accuracy, and how to use the predictive model in optimizing the operational decision. Motivated by these challenges, this dissertation studies the design and prediction of the operational performance for supply chain systems with graphical structures.

The studies in this dissertation can be classified into three categories. First, we

study the design of the optimal operational decisions for assemble-to-order systems. Following the standard assumption that product demands follow independent, stationary Poisson processes, We focus on the theoretical analysis of the system and algorithm development. Second, we study the optimal design of e-commerce transportation networks. A crucial deviation from the classic transportation network design problems is that the network flow decision for any given network is exogenous and difficult to model. In this case, we consider a data-driven approach for predicting the unknown flow using machine learning (ML) models. We then study optimization techniques which are built on the predictive model. Third, we study predicting the operational performance of supply chain systems using advanced ML models that leverage the systems' graphical structures. Different from the analytical performance evaluation tools such as simulation, this new approach is data-driven, i.e., the prediction model relies on observed data to learn and train, without the full knowledge of all the details of the operational mechanisms, such as the component allocation policy in an ATO system, or the analytical properties of the demand process.

In the second chapter, "Optimizing Assemble-to-Order Systems: Decomposition Heuristics and Scalable Algorithms", we study the design of the inventory policy for assemble-to-order (ATO) systems. ATO is a manufacturing strategy to achieve mass customization. Under this strategy, the firm keeps inventory only at the component level, all products are assembled from component inventories only after customer orders with the customer-specified bill of materials are realized. The graphical structure in ATO is the component-product network specified by the bill of materials. The ATO system has the promise to make customized products cheap by avoiding unwanted final product inventory and sharing common components across different products at the same time. However, whether these savings can be fully realized depends heavily on whether component inventories are managed efficiently.

Despite decades of research, our understanding of the structure of optimal inventory policy of ATO systems remains limited. This is because the ATO system combines both the assembly structure (from a product point of view for each product) and the distribution structure (from a component point of view, for each component.) The assembly structure implies a coordinated replenishment policy across different components. The distribution structure implies a state-dependent allocation policy among different products. More recently, there has been some advancement in the form of optimal policy for simple ATO systems, such as the W-system in which there are two products sharing one common component and the M-system in which there are three products sharing two common components. In this chapter, we first join this effort by analyzing a continuous-review M-system with non-identical lead times. (Previous research focuses on the M-system with identical lead times.)

We show that an asymptotically optimal policy consists of a periodic review priority (PRP) allocation rule and a coordinated base-stock (CBS) replenishment policy. The PRP rule fulfills customer orders to myopically minimize the instantaneous total product backorder cost at discrete time points. The CBS policy coordinates the replenishment of the components by integrating the observed demand information. Unfortunately, the policy is very difficult to compute and implement. To overcome this shortcoming, we construct a heuristic replenishment policy by decomposing the system into a set of assembly subsystems, one for each product. Each assembly system follows a reserved base-stock (RBS) policy that takes both the simplicity of implementation and the previous demand information into consideration. A linear programming program is then constructed to compute the RBS policy parameters.

For general systems, it is difficult to coordinate the replenishment among components because each component has a set of components to coordinate. As a result, we study how to design the independent base-stock (IBS) policy where each component

is independently replenished, and we use PRP rule from Paper 2 as the allocation rule. We develop an approximation algorithm which decomposes the ATO system into a set of distribution subsystems, one for each component. It turns out each such subsystem is similar to the newsvendor problem with a straightforward solution. Each subsystem then follows an IBS policy with the base-stock level equal to this straightforward newsvendor solution. Thus, this approximation is scalable to compute IBS policy parameters and can handle large-scale inventory. We also use a primal-dual analysis to show that the limit of the (scaled) expected system cost under the optimal independent base-stock policy in the general system can be bounded by two newsvendor systems with properly set parameters. Finally, our numerical results show that the RBS policy is very effective, the distribution decomposition performs similarly effectively as the optimal IBS policy. Numerical tests also provide some interesting insights into the impact of system parameters on the value of past demand information.

In the third chapter, "Data-Driven Scalable E-commerce Transportation Network Design with Unknown Flow Response", we study the prediction of the shipment cost in an e-commerce network and the approximation of the optimal network. This work was motivated by my internship experience with an online marketplace. The objective of our study is to develop data-driven, scalable optimization tools for e-marketplaces to best utilize their in-house outbound transportation networks. The outbound transportation network of the online marketplace we consider consists of regional warehouses, called fulfillment centers (FCs) and the smaller delivery stations (DSs) closer to the end customers. Based on the customer order, the company first decides how and from which FC to ship the package to which DS, a process called middle-mile transportation. After packages reach their destination DSs, they are shipped to customers' doors. This process is called last-mile transportation. The

middle-mile network is less known than the last-mile network because it is invisible to the customers, but its importance cannot be overlooked as it can be the most expensive part of the whole supply chain, and its market can reach 1 trillion. In this chapter, we focus on optimizing the middle-mile transportation network.

At first glance, the optimization problem appears to resemble some elements of the fixed-charge capacitated multicommodity network design (CMND) problem, but it departs from the classical problem with significant nuances and challenges. The first novel feature is decentralized decision-making. While the middle-mile manager decides the network configuration on a weekly or monthly basis, the real-time flows of millions of packages on any given network configuration are controlled by a fulfillment policy employed by a different decision entity. Due to the complexity of the real fulfillment policy, we face a fixed-cost network design problem with unknown flow response. The second novelty is due to the very feature of the granular customer order and delivery in online shopping – the so-called long-tail phenomenon, which makes it impossible to analytically characterize the demand process.

To meet these challenges, we take a predict-then-optimize approach. First, we develop a predictive model for the unknown flow response for any given network design, using the historical shipment data and ML techniques. Apart from the most natural network-level predictive model, we find that the more parsimonious origin-level and edge-level flow response predictive models are more effective in addressing the curse of dimensionality. We then embed the predictive model to the original network design model to obtain a data-driven network design problem. We characterize this new problem as a c-supermodular minimization problem and develop a linear-time approximation algorithm with a performance guarantee. We demonstrate that this algorithm is scalable and effective in a numerical study.

In the fourth chapter, "A Graph Neural Network Approach for Predicting Network

Performance", we study the prediction of the operational performance in supply chain networks with graphical structures. In particular, we focus on a new ML model, Graph Neural Network (GNN). Different from previous ML models, the GNN model leverages the graphical structure in making predictions. It has demonstrated good prediction accuracy in various contexts, including social, bioinformatics and citation networks, but it has not received much attention in supply chain systems despite the fact that many systems exhibit a graphical structure.

One of our main contributions is to adapt GNNs to evaluate the performances of various supply chain systems. We find that even though the ultimate goal of the prediction is for graph-level operational performance, one should start by building a GNN model to predict node-level or edge-level operational performances and then aggregating predictions together. For edge-level prediction, we propose an edge-node graph transformation approach which improves the prediction accuracy. We also highlight the importance of utilizing the graphical structure in making predictions and considering the direction of the edges in the transformed graph.

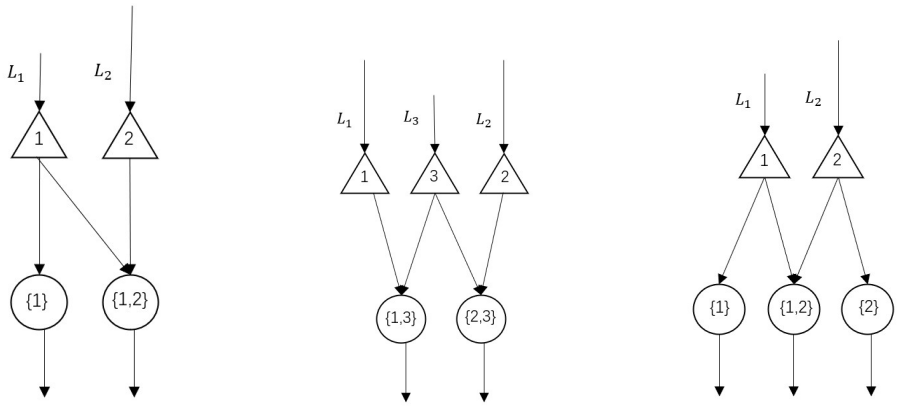
Chapter 2

Optimizing Assemble-to-Order Systems: Decomposition Heuristics and Scalable Algorithms

2.1 Introduction

This chapter considers effective control of continuous-time multiproduct *Assemble-to-order* (*ATO*) systems with non-identical component replenishment leadtimes and full backlogging. In an ATO system, the manufacturer begins to assemble a final product only after customer demand arrives. This approach allows customization of products by only stocking various standardized components, many of which are *common components* shared across different products, thus exploiting the benefit of risk pooling to reduce inventory cost. Figure 2.1 illustrates three simple ATO systems – the N-, the W-, and the M-system, in which the top nodes represent components and the bottom nodes represent products labeled by their component compositions. For example, in the M-system, product-$\{1\}$ (product-$\{2\}$) requires one unit of component-1 (component-2) only, while product-$\{1, 2\}$ requires one unit of component-1 and one unit of component-2. Component i is replenished from an outside supplier with leadtime L_i. Random customer demand occurs at the product level. In both the N- and the W-systems, there is one common component shared by the two end products, while the other component(s) are product specific. In the M-system, there are three products sharing two common components.

With advanced information and production technologies, more and more man-



(a) N-system (b) W-system (c) M-system

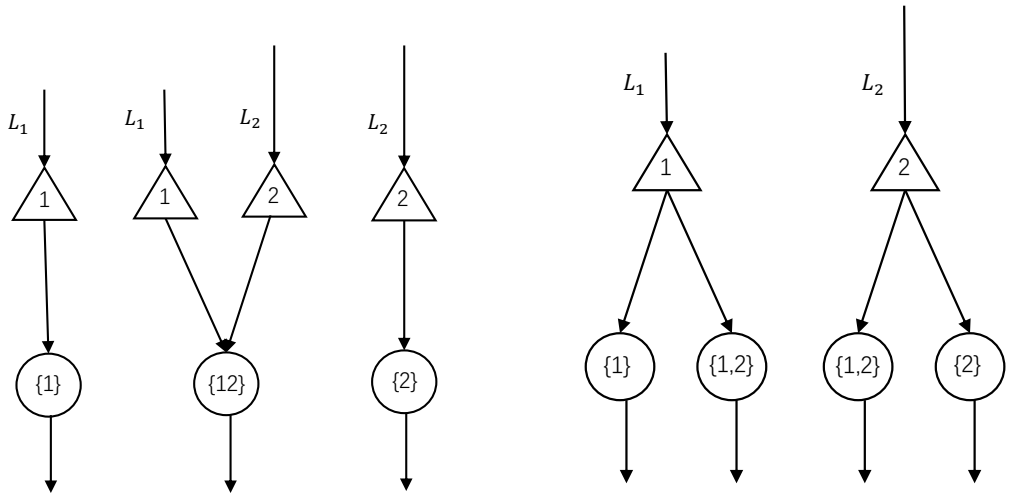
Figure 2.1: Illustration of ATO systems

ufacturing companies, such as Dell (Kapuscinski et al. 2004), Lenovo (Xiao et al. 2010) and BMW (Muller 2010), and e-commerce platforms, such as Amazon (Xu et al. 2009), have adopted this strategy. For example, 35% of the customer orders in an off-peak-season day and 44% in a peak-season day in 2004 consist of multiple items for a large online retailer (see Xu et al. 2009, Zhao et al. 2020). In this context, the items can be viewed as components and customer orders can be viewed as products. As the e-commerce platform starts to prepare for the delivery only after receiving the order, the process to pick and ship the ordered items can be viewed as an assembly process. These practical ATO systems are also large-scale systems with many different components and products. Xu et al. (2009) note that an e-commerce platform can involve more than 400 thousand unique items in five random days.

Despite the popularity of ATO systems, the structure of the optimal inventory policy of these systems remains largely unknown, and scholars are actively searching for efficient and effective approximate control policies. As stated in Song and Zipkin (2003), inventory control of an ATO system consists of two decisions: component inventory replenishment and common-component inventory allocation. An ATO system combines both the assembly structure (from each product's view) and the dis-

tribution structure (from each common component's view). The assembly structure implies that a good component replenishment policy should be a coordinated policy across different components. Heterogeneous replenishment leadtimes complicate such coordination. The distribution structure implies that a good component allocation policy should be state-dependent. The more the common components, the more complex the policy is. Even for a pure distribution system alone, the optimal policy is unknown. For this reason, the majority of the literature focuses on performance evaluation/optimization of commonly used simple policies (see Section 2.2 for a literature review.). Nonetheless, there has been recent progress in the literature beginning to characterize the optimal policies for simple ATO systems, such as the N- and the W-system, which offer some new insights into more sophisticated policies. The purpose of this chapter is to push this line of research further by analyzing the optimal policy for the more complex M-system, and then using the insights to develop scalable and near-optimal control policies for the general ATO system. Section 2.3 introduces the model basics and some preliminaries.

In Section 2.4, we focus on the M-system and examine whether the optimal policy structure for the N- and the W-system discovered in the literature still holds for this system with two common components. We show that while the exact results in the previous work break down, the structure of the asymptotically optimal policy remains the same, which constitutes a *periodic review priority* (PRP) allocation rule and a *coordinated base-stock* (CBS) replenishment policy. Under the PRP rule, the allocation decision is made at discrete time points to myopically minimize the system cost. Under the CBS policy, the component with the longest leadtime follows a constant base-stock policy, and the base-stock level of the other component follows a state-dependent base-stock policy, with the base-stock level depending on the previous demand information of the product requiring this component. Unfortunately,



(a) Assembly Decomposition (b) Distribution Decomposition

Figure 2.1: Illustration of Decomposition Approaches for the M-system

due to two common components, the CBS policy has a complex structure and is computationally inefficient. This motivates us to develop a computationally more efficient heuristic replenishment policy.

To construct a heuristic policy, we first take an *assembly decomposition* approach that decomposes the M-system into three *assembly subsystems*, each of which is an assembly system with one product and all its components, as shown in Figure 2.1(a). Specifically, the M-system is decomposed into three subsystems: subsystems-$\{1\}$, -$\{2\}$, and -$\{1,2\}$, each of which contains exactly one product with all required components. The heuristic policy, which we call the *reserved base-stock or "RBS"* policy, is similar to the CBS policy in that it also coordinates the component inventory replenishment in each assembly subsystem by using a constant base-stock policy for the component with the longest leadtime and state-dependent base-stock policies for the other components. The structure of the state-dependent base-stock policy in the RBS policy is much simpler: the base-stock level of a component does not exceed the available inventory that is virtually reserved for this component at a previous time stamp. This structure is similar to the *echelon balanced base-stock* policy in

the assembly system studied by Rosling (1989), under which the replenishment of a component does not exceed the available inventory of each component with a longer leadtime. (In the V-system in which there is only one product with two components, the CBS, RBS and the echelon balanced base-stock policies are equivalent.) Applying the sample path characterization and the *sample average approximation or "SAA"* method, the RBS policy can be computed through a linear program and is thus easy to compute.

In Section 2.5, we study the general ATO system, i.e., a system with a general bill of materials (BOM) and general leadtimes. For the allocation policy, we adopt the PRP policy that is asymptotically optimal for the M-system. For the replenishment policy, it is natural to consider the extension of the CBS and RBS policies as well as the assembly decomposition. Unfortunately, these extensions turn out to be computationally challenging. For a larger system, the CBS policy requires solving a multistage stochastic program problem. The RBS policy has a large number of policy parameters, and there are many non-convex terms to be transformed into the linear program. For this reason, we focus on developing scalable and effective heuristic replenishment policies. We consider a structurally simpler replenishment policy, the *independent base-stock or "IBS"* policy, under which each component is replenished according to an independent base-stock policy with a constant base-stock level. Because once the policy parameters (the constant base-stock levels) are determined, the implementation of the policy is very simple, which does not need coordination across components, IBS is commonly used in practice. But what should be the base-stock level for each component is not trivial; it should take into account the interrelationships of the components.

To devise effective IBS policy parameters, we take another decomposition approach – the *distribution decomposition*. That is, the ATO system is decomposed

into several distribution subsystems, each of which consists of only one component and all products that require this component, as illustrated in Figure 2.1(b) for the M-system, where the system is decomposed into subsystems-1 and -2, corresponding to each component and all the products requiring the component. We show that a distribution subsystem can be explicitly solved in a manner similar to a *newsvendor* solution by exploiting the optimal sample-path property in each distribution system, and the corresponding optimal newsvendor order quantity is used as the base-stock level in the IBS policy. For this reason, we also call the distribution decomposition approach the *newsvendor decomposition* for short. We further apply a *primal-dual analysis* to show that the expected cost in a scaled general system under the optimal IBS policy is bounded by the expected cost under a series of newsvendor problems with properly selected parameters.

In Section 2.6, we conduct extensive numerical simulations to examine the performance of the RBS policy with the assembly decomposition and IBS policy with the distribution decomposition. We first show they are both effective compared to the expected lower bound of the total system cost under the optimal CBS policy. We then compare the two decomposition approaches to two IBS policy heuristics proposed in the literature: the constant markup newsvendor decomposition heuristic and the linear programming rounding heuristic proposed by DeValve et al. (2020), and use the latter as the benchmark. For simple systems, while the IBS policy with distribution decomposition is much more efficient, the RBS policy with assembly decomposition is more effective with an average cost savings of 1.23% and 3.64% maximum savings compared to the benchmark in the W-system. We find that the RBS policy is more effective when there is a higher chance of products being stock-out. This is consistent with the numerical observations by several other authors, including Benjaafar and ElHafsi (2006) and Nadar et al. (2016). Among the IBS policies, the distribution

decomposition approach performs comparably to the benchmark but is much more efficient as it takes only less than 0.01 seconds to compute, while the benchmark takes 0.38 and 0.36 seconds in the M- and the W-system correspondingly. The distribution decomposition also results in a much lower average system cost compared to the constant markup newsvendor decomposition heuristic, which is computationally comparable. For general systems, we only test IBS policies. We show that the distribution decomposition is highly scalable and very efficient using two real-world examples. Its computational time can be about 10^{-4} of a benchmark with similar effectiveness. The effectiveness of the distribution decomposition is close to the benchmark (slightly worse with 1.46% average extra cost and slightly better with 0.76% to 2.14%, varying according to the size of samples, average cost saving in the two examples), and much better than the constant markup newsvendor decomposition heuristic with similar efficiency. Based on these observations, we recommend RBS policy with assembly decomposition for small systems, and IBS policy with distribution decomposition for large-scale systems.

2.2 Literature Review

Most of the existing studies in the literature of the continues-review ATO system focus on a particular class of policies, see survey papers by Song and Zipkin (2003) and Atan et al. (2017). Early works are restricted to the first-come-first-served (FCFS) allocation rule and IBS replenishment policy, with an objective to evaluate or optimize the IBS policy, e.g., Song (1998), Song et al. (1999), Song (2002), Lu et al. (2003), Zhao and Simchi-Levi (2006), Huang and de Kok (2015) and van Jaarsveld and Scheller-Wolf (2015).

More recently, various scholars start to identify the form of the optimal policy.

Lu et al. (2010) show that under IBS policy, the no-holdback (NHB) allocation rule is optimal among all allocation policies in the W-System and its generalizations with non-identical leadtimes when the costs are symmetric. The NHB rule was first described by Song and Zhao (2009): the component is assigned to a product demand if and only if all the required components are available. Doğru et al. (2010) show that NHB rule with IBS policy can achieve the sample-path lower bound in the W-System only when the costs are symmetric and leadtimes are identical. Lu et al. (2015) further show that NHB rule with CBS policy can achieve the sample-path lower bound in the N- and W-System when the costs are symmetric and leadtimes are non-identical. Different from the above works, we study the optimal policy for the M-system, which is structurally more challenging than the N- and the W-system with one extra common component, with general leadtimes and cost parameters.

For systems with general cost parameters, scholars also identify the form of the asymptotically optimal policy. Plambeck and Ward (2006) consider a general ATO system that has in-house component production with finite capacity and random production times. They develop an asymptotically optimal policy (in high demand volume) for dynamic product pricing, component production capacity setting, and sequencing orders for assembly. They show that the PRP rule is the asymptotically optimal assembly policy. Under this rule, no product is assembled between review time points. At each time point, this rule myopically minimizes the instantaneous system cost, i.e., priority is assigned to products with higher unit inventory cost. Lu et al. (2015) show that the combination of the PRP allocation rule and a CBS inventory replenishment policy is asymptotically optimal for the N-and the W-system with general leadtimes when demand volume is large. In the current chapter, we extend their result to the M-system. Due to the existence of the extra common component, the replenishment policy in the M-system has a more complex structure. Reiman

and Wang (2015) propose a variant of the PRP rule, which serves demand continuously. They develop an asymptotically optimal IBS policy parameters in the general system with long, identical leadtimes. Doğru et al. (2017) consider specially structured ATO systems with "chained BOM", and show the problem is $L^\natural$ convex and the IBS policy parameters can be solved efficiently. Reiman et al. (2020) consider the general system with general leadtimes and develop an asymptotically optimal policy for long leadtimes. Their replenishment policy first uses the demand information of all products to compute the targeted inventory position of one component, and then uses the maximum value between the current net inventory level of this component and the targeted inventory position as the final inventory position. Our policy only considers the demand information of the product assembled by this component. This simpler structure helps the development of the heuristic RBS replenishment policy.

All of the above works assume that the unfulfilled demands are backordered as in our setting. Several scholars have studied systems with lost sales systems. Benjaafar and ElHafsi (2006) consider a system with a single product (i.e., no common components) demanded by different customer classes, and show that the optimal policy consists of a state-dependent base-stock and a state-dependent rationing policy. Nadar et al. (2014) extend this work to consider a general M-system where the individual product can require different units of components and components are produced in batches. They show the optimal policy is a lattice-dependent base-stock and lattice-dependent rationing, which generalizes the state-dependent base-stock and state-dependent rationing policy. Both policies share similar features of the CBS policy and the PRP allocation rule: the policy parameters are coordinated based on the current system state, which depends on the previous demand information.

To the best of our knowledge, there is no prior work developing a heuristic replenishment policy which integrates the previous demand information as in the CBS

policy. In other words, our RBS policy with assembly decomposition is new to the literature. Albrecht (2014) also uses assembly decomposition, but the author focuses on IBS policies in Make-to-Stock systems. To compute the base-stock levels, the author solves each subsystem separately and then aggregates the solutions. ElHafsi et al. (2020) study a ATO system with a general BOM and random leadtimes. They propose a two-step decomposition approach: the first step decomposes the system into assembly subsystems and the second step decomposes the assembly subsystem into M-systems where each M-system includes one of the products from the assembly subsystem, one of the components required by that product, and another component consisting of all the other components required by that product. Different from these computational approaches, we formulate a linear program to simultaneously determine RBS policy parameters in the entire system.

For systems using IBS replenishment policies, several authors use a newsvendor decomposition approach to develop heuristic base-stock levels. Lu and Song (2005) study the optimal IBS policy in a general dynamic ATO system with non-identical leadtimes under the FCFS allocation rule. For a heuristic policy, they propose a newsvendor decomposition in which the component shortage cost is set to be the demand-rate weighted average of product backorder costs. DeValve et al. (2020) consider a single-period ATO system with a general BOM and identical leadtimes. In their newsvendor decomposition, the component shortage cost is set conservatively based on the constant markup heuristic. They study the performance bound of this newsvendor decomposition heuristic. They also propose a linear programming rounding heuristic which is asymptotically optimal in both the single-period and the continuous-review system with identical leadtimes if the leadtimes are large. Different from these works, we develop a newsvendor decomposition heuristic where the component shortage cost is set based on the optimal sample-path properties in the

subsystem.

2.3 Model and Preliminaries

We consider a continuous-review ATO system with multiple components and multiple products. Each product uses several components. Components are indexed by $i \in \mathcal{I} = \{1, 2, \ldots, m\}$, and products are indexed by K, a subset of $\mathcal{I}$, if it requires one unit of each component in K and none in $\mathcal{I} \setminus K$. Let $\mathcal{K}$ be the set of all products. We use subscripts i to index the quantities related to component-i, and superscripts K to index the quantities related to product-K. For each component-i, let $\mathcal{K}_i$ be the set of the products that require component-i. For instance, the following are the generalize M- (with $m+1$ products) and W-system (with $m-1$ products):

$$\text{General M-system}: \quad \mathcal{K} = \{\{1\}, \{2\}, \ldots, \{m\}, \mathcal{I}\}, \mathcal{K}_i = \{\{i\}, \mathcal{I}\}, i \in \mathcal{I};$$

$$\text{General W-system}: \quad \mathcal{K} = \{\{i, m\}, i \in \mathcal{I}, i \neq m\}, \mathcal{K}_m = \mathcal{K}, \mathcal{K}_i = \{\{i, m\}\}, \forall i \neq m.$$

In the general M-system, product $\mathcal{I}$ is a super product assembled by all components, and the rest products only require one component. This can be seen as a case where the seller sells a series of products (e.g., novels). The customer is likely to either buy one of the series or the entire series together. In the general W-system, component-m is a super component which is required by all products. We can think the system as a seller selling a core item (e.g., a video game console), which is always bundled with other items (e.g., different video games) in cross-selling.

For $i \in \mathcal{I}$, $K \in \mathcal{K}$, and any time $t \geq 0$, denote

$$\begin{aligned}
&\mathcal{D}^K(t) = \text{ cumulative demand of product } K \text{ in } (0,t],\\
&\mathcal{D}_i(t) = \sum_{K \in \mathcal{K}_i} \mathcal{D}^K(t) = \text{ cumulative demand for component } i \text{ in } (0,t],\\
&\mathcal{D}^K(t-a,t] = \mathcal{D}^K(t) - \mathcal{D}^K(t-a),\ 0 \leq a < t,\\
&\mathcal{D}_i(t-a,t] = \mathcal{D}_i(t) - \mathcal{D}_i(t-a),\ 0 \leq a < t.
\end{aligned}$$

We assume that the demand process $\{(\mathcal{D}^K(t), K \in \mathcal{K}), t \geq 0\}$ is stationary and has independent increments with mean demand rate $\lambda^K = \mathsf{E}[D^K(1)]$. This implies that the component-i demand process $\{\mathcal{D}_i(t), t \geq 0\}$ is also stationary and has independent increments with mean demand rate $\lambda_i = \sum_{K \in \mathcal{K}_i} \lambda^K = \mathsf{E}[D_i(1)]$. We assume continuous demand. A similar analysis can be carried through to the discrete demand case with the derivative operation replaced by the difference operation. For each component i, let L_i be the replenishment leadtime for component-i, a nonnegative constant. Denote $L = \min\{L_i :\ i \in \mathcal{I}\}$.

We now describe the state of the system and the control policies. Let $I_i(0)$ and $B^K(0)$ be the initial on-hand inventory for component-i and initial backorders for product-K, respectively. For any $t \geq 0$, define

$$\begin{aligned}
O_i(t) &= \text{cumulative replenishment orders for component-}i \text{ in } [0,t] \text{ with } O_i(0) = 0,\\
A^K(t) &= \text{total product-}K \text{ demands satisfied in } [0,t], \text{ with } A^K(0) = 0,\\
\mathcal{F}_t &= \text{the } \sigma\text{-field generated by } \{(\mathcal{D}^K(s), K \in \mathcal{K}), 0 \leq s \leq t\}.
\end{aligned}$$

Definition 1. (Admissible Policy) *A policy* $\pi = \{(O_i(t), A^K(t), i \in \mathcal{I}, K \in \mathcal{K}), t \geq 0\}$

is called admissible if it satisfies, for any $t > 0$,

(i) $O_i(t)$ *and* $A^K(t)$ *are jointly determined by the system's available information until time* t, $\mathcal{F}_t$;

(ii) $\sum_{K \in \mathcal{K}_i} A^K(t) \leq I_i(0) + O_i(t - L_i), i \in \mathcal{I}$;

(iii) $A^K(t) \leq B^K(0) + \mathcal{D}^K(t), K \in \mathcal{K}$.

The set of all admissible policies is denoted as $\mathcal{A}$. We call $\{(O_i(t), i \in \mathcal{I}), t \geq 0\}$ a *replenishment* policy or process, and $\left\{(A^K(t), K \in \mathcal{K}), t \geq 0\right\}$ an *allocation* rule or process. For any $\pi = \left\{(O_i(t), A^K(t), i \in \mathcal{I}, K \in \mathcal{K}), t \geq 0\right\} \in \mathcal{A}$, let

$$
\begin{aligned}
I_i(t, \pi) &= \text{on-hand inventory of component-}i \text{ at time } t, \\
B^K(t, \pi) &= \text{backorders for product-}K \text{ at time } t, \\
IP_i(t, \pi) &= \text{inventory position of component-}i\text{, after ordering at time } t, \\
N_i(t, \pi) &= IP_i(t - L_i, \pi) - \mathcal{D}_i(t - L_i, t] = \text{ net inventory of component-}i \text{ at time } t.
\end{aligned}
$$

The system dynamics are:

$$
\begin{aligned}
I_i(t, \pi) &= I_i(0) + O_i(t - L_i) - \sum_{K \in \mathcal{K}_i} A^K(t), && (2.1) \\
B^K(t, \pi) &= B^K(0) + \mathcal{D}^K(t) - A^K(t), && (2.2) \\
IP_i(t, \pi) &= IP_i(0) + O_i(t) - \mathcal{D}_i(t), && (2.3)
\end{aligned}
$$

where $IP_i(0)$ is the initial inventory position. Without loss of generality, we assume $IP_i(0) = I_i(0)$ and $B^K(0) = 0$.

Let h_i be the unit holding cost rate of component-i and b^K be the unit backorder

cost rate of product-K, the total inventory cost at time t is

$$C(t,\pi) = \sum_{i\in\mathcal{I}} h_i \cdot I_i(t,\pi) + \sum_{K\in\mathcal{K}} b^K \cdot B^K(t,\pi), \tag{2.4}$$

The net inventory of component-i can be calculated from the state variables as follows:

$$I_i(t,\pi) - \sum_{K\in\mathcal{K}_i} B^K(t,\pi) = N_i(t,\pi),\ i \in \mathcal{I}. \tag{2.5}$$

Then the system cost in (2.4) can be reformulated as

$$C(t,\pi) = \sum_{i\in\mathcal{I}} h_i \cdot N_i(t,\pi) + \sum_{K\in\mathcal{K}} c^K \cdot B^K(t,\pi) \ \text{ with } \ c^K = b^K + \sum_{i\in K} h_i. \tag{2.6}$$

c^K can be interpreted as the unit inventory cost, which reflects the marginal cost saved from the system by fulfilling one unit of product-K. Our objective is to find an admissible $\pi \in \mathcal{A}$ to minimize the long-run average expected total system cost:

$$\limsup_{T\to\infty} \frac{1}{T}\mathsf{E}\left[\int_0^T C(t,\pi)\mathsf{d}t\right]. \tag{2.7}$$

Our analysis is based on *the sample-path approach*, which considers the system cost from the system dynamics given by (2.1)-(2.3) and (2.5) without taking the expectation in (2.7). As our policy (see Definition 1) has two parts: replenishment and allocation. The replenishment decision $\{(O_i(t), i \in \mathcal{I}), t \geq 0\}$ determines $\{(N_i(t,\pi), i \in \mathcal{I}), t \geq 0\}$ while the allocation decision $\{(A^K(t), K \in \mathcal{K}), t \geq 0\}$ determines $\{(B^K(t,\pi), K \in \mathcal{K}), t \geq 0\}$. For a given replenishment decision $\{(O_i(t), i \in \mathcal{I}), t \geq 0\}$, we look at how to make allocation decision so as to minimize the system

cost given by (2.6). In view of (2.3) and (2.5), and the definition of $N_i(t,\pi)$, we have

$$\begin{aligned} C(t,\pi) &= \sum_{i\in\mathcal{I}} h_i \cdot N_i(t,\pi) + \sum_{K\in\mathcal{K}} c^K \cdot B^K(t,\pi) \\ &= \sum_{i\in\mathcal{I}} h_i \cdot \Big(I_i(0) + O_i(t-L_i) - \mathcal{D}_i(t)\Big) + \sum_{K\in\mathcal{K}} c^K \cdot B^K(t,\pi), \qquad (2.8) \\ \text{s.t.} \quad & \sum_{K\in\mathcal{K}_i} B^K(t,\pi) \geq [N_i(t,\pi)]^- = \Big[I_i(0) + O_i(t-L_i) - \mathcal{D}_i(t)\Big]^-, \\ & \qquad\qquad i \in \mathcal{I}; \qquad (2.9) \\ & B^K(t,\pi) \geq 0, \ K \in \mathcal{K}. \qquad (2.10) \end{aligned}$$

Now consider the following linear program corresponding to (2.8)-(2.10):

$$\begin{aligned} \varphi(x_1,\ldots,x_m) &:= \min_{y^K: K\in\mathcal{K}} \Big\{ \sum_{i\in\mathcal{I}} h_i \cdot x_i + \sum_{K\in\mathcal{K}} c^K \cdot y^K \Big\}, \qquad (2.11) \\ \text{s.t.} \quad & \sum_{K\in\mathcal{K}_i} y^K \geq x_i^-, \ i \in \mathcal{I}; \qquad (2.12) \\ & y^K \geq 0, \ K \in \mathcal{K}. \qquad (2.13) \end{aligned}$$

By (2.9)-(2.10), we know that $C(t,\pi) \geq \varphi(N_1(t,\pi),\ldots,N_m(t,\pi))$ for any $t \geq 0$. Hence, if an allocation policy $\{(A^K(t), K \in \mathcal{K}), t \geq 0\}$ can attain the expected lower bound $\mathsf{E}[\varphi(N_1(t,\pi),\ldots,N_m(t,\pi))]$, then this policy must be optimal among the whole feasible allocation decisions for the given replenishment policy which determines $\{(N_i(t,\pi), i \in \mathcal{I}), t \geq 0\}$. Certainly, if an allocation policy can attain the sample-path lower bound $\varphi(N_1(t,\pi),\ldots,N_m(t,\pi))$, then it can attain the expected lower bound.

2.4 The M-system: Asymptotically Optimal Policy

We first look at the sample-path lower bound for the M-system, which then facilitates us to develop the periodic review priority allocation rule and the optimal replacement policy. For notational simplicity, we write product-$\{i\}$ as product-i, $i \in \{1,2\}$, and product-$\{1,2\}$ as product-12. Therefore, we have $\mathcal{K} = \{1,2,12\}$.

2.4.1 Properties of Sample-path Lower Bound

At any given time $t \geq 0$, conditioning on the net component inventory $(N_1(t,\pi), N_2(t,\pi))$, in view of the linear program (2.11)-(2.13), the sample-path lower bound is

$$\begin{aligned} \varphi(N_1(t,\pi), N_2(t,\pi)) \;=\; & \min_{y^K : K=1,2,12} \Big[\sum_{i=1,2} h_i \cdot N_i(t,\pi) + \sum_{K=1,2,12} c^K \cdot y^K \Big], \quad (2.14) \\ \text{s.t.} \quad & y^i + y^{12} \geq [N_i(t,\pi)]^-, \; i = 1,2 \\ & y^1, y^2, y^{12} \geq 0. \end{aligned}$$

The solution to (2.14) will show us what the backorders for different products can attain the lower bound $\varphi(N_1(t,\pi), N_2(t,\pi))$ for the given net inventory of each component. Except for the dependence on the net inventory at time t, the solution also depends on the relationship between c^1, c^2 and c^{12}, and is given by the following proposition.

Proposition 1. (Properties of the Sample-path Lower Bound) *The solution of* (2.14) *denoted by* (y_*^1, y_*^2, y_*^{12}) *is given by*

(i) *If* $c^1 + c^2 = c^{12}$, *then* $y_*^1 + y_*^{12} = [N_1(t,\hat{\pi})]^-, y_*^2 + y_*^{12} = [N_2(t,\hat{\pi})]^-$;

(ii) *If* $c^1 + c^2 < c^{12}$, *then* $y_*^{12} = 0, y_*^1 = [N_1(t,\hat{\pi})]^-, y_*^2 = [N_2(t,\hat{\pi})]^-$;

(iii) *If* $c^1 \vee c^2 < c^{12} < c^1 + c^2$, *then* $y_*^{12} = [N_1(t,\hat{\pi})]^- \wedge [N_2(t,\hat{\pi})]^-, y_*^1 = [[N_2(t,\hat{\pi})]^- - [N_1(t,\hat{\pi})]^-]^-, y_*^2 = [[N_1(t,\hat{\pi})]^- - [N_2(t,\hat{\pi})]^-]^-$;

(iv) *If* $c^2 < c^{12} \leq c^1$, *then* $y_*^{12} = [N_1(t,\hat{\pi})]^-, y_*^1(t,\hat{\pi}) = 0, y_*^2 = [[N_1(t,\hat{\pi})]^- - [N_2(t,\hat{\pi})]^-]^-$;

(v) *If* $c^1 < c^{12} \leq c^2$, *then* $y_*^{12} = [N_2(t,\hat{\pi})]^-, y_*^1 = [[N_2(t,\hat{\pi})]^- - [N_1(t,\hat{\pi})]^-]^-, y_*^2 = 0$;

(vi) *If* $c^{12} \leq c^1 \wedge c^2$, *then* $y_*^{12} = [N_1(t,\hat{\pi})]^- \vee [N_2(t,\hat{\pi})]^-, y_*^1 = 0, y_*^2 = 0$.

Moreover, if there exists an admissible policy $\pi_* \in \mathcal{A}$ *such that the sample-path lower bound* (2.14) *can be achieved, it must have*

$$B^1(t,\pi_*) = y_*^1;\ B^2(t,\pi_*) = y_*^2;\ B^{12}(t,\pi_*) = y_*^{12}.$$

Proposition 1 raises the question of whether there are any allocation rules achieving the sample-path lower bound. One possible rule is the NHB rule, that is formally defined as follows.

Definition 2. (NHB Rule) *We say that an admissible policy* $\pi = \{(O_i(t), A^K(t), i \in \mathcal{I}, K \in \mathcal{K}), t \geq 0\} \in \mathcal{A}$ *is no-holdback (NHB) if the following conditions are satisfied for all* $t \geq 0$ *and* $K \in \mathcal{K}$,

$$B^K(t,\pi) \times \min\{I_i(t,\pi), i \in K\} = 0. \tag{2.15}$$

For the N- and the W-system, Lu et al. (2015) show that an NHB rule can achieve the sample-path lower bound under certain symmetric conditions on the cost parameters. However, this result cannot be extended to the M-system. It is obvious that no allocation rule is optimal in case (ii), (iv), (v) and (vi): it is impossible for a product to always have zero backorder because the demand is not controllable and it

is always to encounter a demand larger than the current inventory. The following is a counterexample to show no allocation rule, including NHB, can achieve the sample-path lower bound in case (i) and (iii).

Consider the following case which satisfies the sample-path properties before time t,

$$\begin{aligned} &I_1(t-) = 1,\ I_2(t-) = 0,\ B^1(t-) = 0,\ B^2(t-) = 0,\ B^{12}(t-) = 0, \\ &N_1(t-) = 1,\ N_2(t-) = 0, \end{aligned} \tag{2.16}$$

where we use $t-$ to denote an instant right before time t. Suppose there is a demand of product-12 at time t, then this demand cannot be fulfilled regardless of the allocation rule. As a result, we have

$$\begin{aligned} &I_1(t) = 1,\ I_2(t) = 0,\ B^1(t) = 0,\ B^2(t) = 0,\ B^{12}(t) = 1, \\ &N_1(t) = 0,\ N_2(t) = -1. \end{aligned} \tag{2.17}$$

In particular, $B^1(t)+B^{12}(t) = 1 \neq [N_1(t)]^- = 0$ (case (i)) and $B^{12}(t) \neq [N_1(t)]^- \wedge [N_2(t)]^- = 0$ (case (iii)). Thus, it is impossible to maintain the sample-path property.

The counterexample shows that the NHB allocation rule is not optimal for the M-system even if we assume a symmetric cost condition as in case (i). This is because the sample-path properties in case (i) imply that one unit increase in backorder of product-12, $B^{12}(t, \hat{\pi})$ requires one unit increase in the shortage of all the required components, $[N_1(t, \hat{\pi})]^-$ and $[N_2(t, \hat{\pi})]^-$. Obviously, this condition is not always true as an increase in backorder of product-12 can be caused by the shortage of only one of the required components. In contrast, in the N- and the W-system, the sample-path properties imply that one unit increase in the common product backorders requires

one unit increase in the shortage of at least one of its required components, which can be achieved by the NHB rule.

2.4.2 PRP Allocation Rule

As no policy can exactly attain the sample-path lower bound, we conduct an asymptotic analysis with high-volume demand on the diffusion scale and show there exists an allocation rule which can achieve the limit of the (scaled) expected sample-path lower bound cost. Similar to Lu et al. (2015), we consider a sequence of M-systems, indexed by n, in which all problem parameters remain the same as before, but the product demand rates are linear in n. All processes in the nth system are superscripted by n. In particular, the product-K demand rate in the nth system is

$$\mathsf{E}\mathcal{D}^{K,n}(1) = n\lambda^K, \ K = 1, 2, 12, \tag{2.18}$$

where superscript n refers to the nth system.

We first equally partition time interval $[0, \infty)$ into subintervals such that in each subinterval there is only one component allocation decision. For any $\pi^n = \{(O_i^n(t), A^{K,n}(t), i = 1, 2, K = 1, 2, 12), t \geq 0\} \in \mathcal{A}^n$, we modify the allocation process $\{(A^{K,n}(t), K = 1, 2, 12), t \geq 0\}$ into a discrete-review policy $\{(A_\star^{K,n}(t), K = 1, 2, 12), t \geq 0\}$ such that $\pi_\star^n = \{(O_i^n(t),\ A_\star^{K,n}(t)\ , i = 1, 2, K = 1, 2, 12), t \geq 0\} \in \mathcal{A}^n$. The modified policy allocates components for products at review points $\delta^n, 2\delta^n, 3\delta^n, \ldots$ with

$$\delta^n = \Big(\frac{1}{n\sqrt{(\lambda^1)^2 + (\lambda^2)^2 + (\lambda^{12})^2}}\Big)^{2/3},$$

and does not allocate any component at all other times.

Consider any $K \in \{1, 2, 12\}$. Initially, set $A_\star^{K,n}(0) = 0$ and $B_\star^{K,n}(0) = 0$. Then, the process is defined recursively by

$$A_\star^{K,n}(\ell\delta^n) \;\; = \;\; \mathcal{D}^{K,n}(\ell\delta^n) - B_\star^{K,n}(\ell\delta^n),$$

where $B_\star^{K,n}(\ell\delta^n)$ solve

$$\min_{B^K \geq 0} \left\{ b^1 B^1 + b^{12} B^{12} + b^2 B^2 + h_1 I_1 + h_2 I_2 \right\} \tag{2.19}$$

$$\begin{aligned} \text{s.t.} \;\; I_1 &= I_1^n(0) + O_1^n(\ell\delta^n - L_1) - \mathcal{D}^{1,n}(\ell\delta^n) - \mathcal{D}^{12,n}(\ell\delta^n) + B^1 + B^{12} \\ &\geq 0; \end{aligned} \tag{2.20}$$

$$\begin{aligned} I_2 &= I_2^n(0) + O_2^n(\ell\delta^n - L_2) - \mathcal{D}^{2,n}(\ell\delta^n) - \mathcal{D}^{12,n}(\ell\delta^n) + B^2 + B^{12} \\ &\geq 0; \end{aligned} \tag{2.21}$$

$$A^{K,n}((\ell-1)\delta^n) + B^K \leq \mathcal{D}^{K,n}(\ell\delta^n), \quad K = 1, 2, 12. \tag{2.22}$$

For $t \in [\ell\delta^n, (\ell+1)\delta^n)$, define $A_\star^{K,n}(t) = A_\star^{K,n}(\ell\delta^n)$. We now formally define the PRP rule.

Definition 3. (PRP Allocation Rule) *We say that an admissible policy* $\pi^n = \{(O_i^n(t), A^{K,n}(t),\ i = 1, 2, K = 1, 2, 12), t \geq 0\} \in \mathcal{A}^n$ *is PRP (periodic-review priority allocation) policy in the* n*th system if for all* $t \geq 0$ *and* $K = 1, 2, 12$, $A^{K,n}(t) = A_\star^{K,n}(t)$.

We also introduce an algorithm to implement the PRP rule for $t = \ell\delta^n$ in practice. We sort the products in $\{1, 2, 12\}$ in descending order based on c^K. Instead of computing $B_\star^{K,n}(t)$, we consider $\hat{B}_\star^{K,n}(t)$, the solution to (2.19) without constraint (2.22), to ease the computation burden. As a result, $\hat{B}_\star^{K,n}(t)$ is not always reachable. We use $\hat{A}^{K,n}(t)$ to denote the allocation of product-K at time t.

Algorithm 1: PRP Algorithm

1: Input $I_i^n(t-), i = 1, 2,\ B^{K,n}(t-),\ \hat{B}_\star^{K,n}(t), K = 1, 2, 12$

2: Initialize $\hat{A}^{K,n}(t) \leftarrow 0, K = 1, 2, 12$

3: **for** $K = 1, 2, 12$ **do**

4: $\quad \hat{A}^{K,n}(t) = \min(B^K(t-) - \hat{B}_\star^{K,n}(t), \min_{i\in K}\{I_i^n(t-)\})$

5: $\quad I_i^n(t-) = I_i^n(t-) - \hat{A}^{K,n}(t), \forall i \in K$

6: **end for**

The difference between PRP and NHB is that PRP only makes allocation decisions at discrete time points. The delayed decision gives the system more flexibility to achieve the (scaled) expected sample-path lower bound cost of (2.7), which is shown in the following proposition.

Proposition 2. (Lower Bound's Attainability under PRP Rule) *For each* $\pi^n = \{(O_i^n(t), A^{K,n}(t),\ i = 1, 2, K = 1, 2, 12), t \geq 0\} \in \mathcal{A}^n$*, there exists a PRP policy* $\pi_*^n = \{(O_i^n(t), A_*^{K,n}(t), i = 1, 2, K = 1, 2, 12), t \geq 0\} \in \mathcal{A}^n$ *with* $A_*^{K,n}(t) = A_\star^{K,n}(\ell\delta^n)$ *for* $t \in [\ell\delta^n, (\ell+1)\delta^n)$ *and* $\ell \geq 0$ *such that the limit of the (scaled) expected sample-path lower bound cost can be achieved. That is, for any* $T > 0$*,*

$$\limsup_{n\to\infty} \frac{1}{\sqrt{n}}\frac{1}{T}\mathsf{E}\int_0^T \Big(\sum_{i\in\mathcal{I}} h_i \cdot I_i^n(t, \pi_*^n) + \sum_{K\in\mathcal{K}} b^K \cdot B^{K,n}(t, \pi_*^n)\Big)\mathsf{d}t$$
$$= \limsup_{n\to\infty} \frac{1}{\sqrt{n}}\frac{1}{T}\mathsf{E}\int_0^T \varphi\Big(O_1^n(t - L_1) - \mathcal{D}_1^n(t), O_2^n(t - L_2) - \mathcal{D}_2^n(t)\Big)\mathsf{d}t \quad (2.23)$$

2.4.3 Asymptotically Optimal Policy

Using the PRP rule as the allocation policy, we now focus on identifying the structure of the replenishment policy (in terms of $IP_1^n(t, \cdot)$ and $IP_2^n(t, \cdot)$) to minimize (2.23), the limit of the (scaled) expected lower bound of the system cost, and hence complete the

characterization of the asymptotically optimal policy π^n_*. We only present analysis for $L_1 \le L_2$ with $L = L_1$ and $\Delta = L_2 - L_1$ ($L_1 \ge L_2$ can be analyzed in the same way). For ease exposition, we assume that the variances and covariance of $\mathcal{D}^{K,n}\left(\frac{1}{n}\right)$ are independent of n, and (in this subsection) $\mathsf{E}\left(\mathcal{D}^{K,n}\left(\frac{1}{n}\right)\right)^5 < \infty,\ K = 1, 2, 12$. Denote

$$\sigma^K = \sqrt{\mathsf{Var}\Big[\mathcal{D}^{K,n}\Big(\frac{1}{n}\Big)\Big]}, \quad \sigma^{K,K'} = \sqrt{\mathsf{Cov}\Big[\mathcal{D}^{K,n}\Big(\frac{1}{n}\Big), \mathcal{D}^{K',n}\Big(\frac{1}{n}\Big)\Big]},\ K, K' = 1, 2, 12$$

and $K \neq K'$.

For notation convenience, we denote $\sigma^{K,K} = \sigma^K$.

Before analyzing (2.23), we first consider a scaled system where the demand during $[t - L, t]$ and the inventory positions are centered and then scaled down by a factor $\sqrt{n}$, that is, recalling $\lambda_i = \lambda^i + \lambda^{12}$,

$$\frac{\mathcal{D}^{K,n}(t - L, t] - n\lambda^K L}{\sqrt{n}},\ K = 1, 2, 12; \tag{2.24}$$

$$\frac{IP^n_i(t - L_i, \cdot) - n\lambda_i L_i}{\sqrt{n}},\ i = 1, 2. \tag{2.25}$$

We know that with probability one, for $t \ge L$,

$$\Big(\frac{\mathcal{D}^{1,n}(t - L, t] - n\lambda^1 L}{\sqrt{n}}, \frac{\mathcal{D}^{2,n}(t - L, t] - n\lambda^2 L}{\sqrt{n}}, \frac{\mathcal{D}^{12,n}(t - L, t] - n\lambda^{12} L}{\sqrt{n}}\Big)$$

converge to a multivariate normal variable $(\Theta^1(t), \Theta^2(t), \Theta^{12}(t))$ with zero mean and covariance matrix $\Gamma = (\Gamma^{K,K'})_{3\times 3}$ with $\sqrt{\Gamma^{K,K'}} = \sigma^{K,K'}\sqrt{L}$, $K, K' = 1, 2, 12$. For any $t \ge L + \Delta$,

$$\Big(\frac{\mathcal{D}^{2,n}(t - L - \Delta, t - L] - n\lambda^2 \Delta}{\sqrt{n}}, \frac{\mathcal{D}^{12,n}(t - L - \Delta, t - L] - n\lambda^{12} \Delta}{\sqrt{n}}\Big)$$

converge to a bivariate normal variable $(\Delta\Theta^2(t-L), \Delta\Theta^{12}(t-L))$ with zero mean and covariance matrix $\Xi = (\Xi^{K,K'})_{2\times 2}$ with $\sqrt{\Xi^{K,K'}} = \sigma^{K,K'}\sqrt{\Delta}$, $K, K' = 2, 12$. Suppose that with probability one, for $t \geq L_i$,

$$\lim_{n\to\infty} \frac{IP_i^n(t - L_i, \cdot) - n\lambda_i L_i}{\sqrt{n}} = IP_i^\infty(t - L_i, \cdot), \quad i = 1, 2. \tag{2.26}$$

Define $\Theta_i(t) = \Theta^i(t) + \Theta^{12}(t), \Delta\Theta_i(t) = \Delta\Theta^i(t) + \Delta\Theta^{12}(t), i = 1, 2$. Then by Propositions 1 and 2, the expected sample-path lower bound in the scaled system is

$$\mathsf{E}\varphi\Big(IP_1^\infty(t - L, \cdot) - \Theta_1(t), IP_2^\infty(t - L - \Delta, \cdot) - \Theta_2(t) - \Delta\Theta_2(t - L)\Big) \tag{2.27}$$

Note that (2.27) is a multistage stochastic program, which is notoriously difficult to solve in general. But utilizing the fact $\mathcal{F}_{t-L-\Delta}^\infty \subseteq \mathcal{F}_{t-L}^\infty$ and applying the tower property of the conditional expectations, we have

$$\begin{aligned} & \mathsf{E}\Big[\varphi\Big(IP_1^\infty(t - L) - \Theta_1(t), IP_2^\infty(t - L - \Delta) - \Theta_2(t) - \Delta\Theta_2(t - L)\Big)\Big] \qquad (2.28) \\ = \; & \mathsf{E}\Big\{\mathsf{E}\Big(\mathsf{E}\Big[\varphi\Big(IP_1^\infty(t - L) - \Theta_1(t), IP_2^\infty(t - L - \Delta) - \Theta_2(t) - \Delta\Theta_2(t - L)\Big)\Big|\mathcal{F}_{t-L}^\infty\Big] \\ & \Big|\mathcal{F}_{t-L-\Delta}^\infty\Big)\Big\} \end{aligned}$$

To find the (asymptotically) optimal replenishment policy $IP_1^n(t, \cdot)$ and $IP_2^n(t, \cdot)$, we first determine the optimal $IP_1^\infty(t - L, \cdot)$ and $IP_2^\infty(t - L, \cdot)$ in (2.27). To this end, let $(\Theta^1, \Theta^2, \Theta^{12})$ to be a multivariate normal random vector with zero-mean and covariance matrix Γ, and $(\Delta\Theta^2, \Delta\Theta^{12})$ to be a bivariate normal variable with zero-mean and covariance matrix Ξ. Define $\Theta_i = \Theta^i + \Theta^{12}, i = 1, 2$, and define the limit

cost functions and their minimizers:

$$
\begin{aligned}
H(y_1, y_2) &= \mathsf{E}\varphi\left(y_1 - \Theta_1, y_2 - \Theta_2\right), && (2.29)\\
H_1(y_1, y_2) &= \mathsf{E}H(y_1, y_2 - \Delta\Theta^2), && (2.30)\\
\eta_1(y_2) &= \arg\min_{y_1} H_1(y_1, y_2), && (2.31)\\
H_2(y_2) &= \mathsf{E}H_1\left(\eta_1(y_2 - \Delta\Theta^{12}), y_2 - \Delta\Theta^{12}\right), && (2.32)\\
\eta_2 &= \arg\min_{y_2} H_2(y_2). && (2.33)
\end{aligned}
$$

The following theorem fully characterizes the asymptotically optimal policy.

Theorem 1. (Asymptotically Optimal Policy) *The PRP allocation rule and the CBS-type replenishment policy with*

$$
\begin{aligned}
IP_1^n(t) &= \sqrt{n}\cdot\eta_1\Big(\eta_2 - \frac{\mathcal{D}^{12,n}(t-\Delta, t] - n\lambda^{12}\Delta}{\sqrt{n}}\Big) + n\lambda_1 L; && (2.34)\\
IP_2^n(t) &= \sqrt{n}\cdot\eta_2 + n\lambda_2(L+\Delta) && (2.35)
\end{aligned}
$$

together denoted by π_*^n belongs to $\mathcal{A}^n$ and is asymptotically optimal in $\mathcal{A}^n$, that is, for any $\pi^n \in \mathcal{A}^n$ and $T > 0$,

$$
\begin{aligned}
&\limsup_{n\to\infty} \frac{1}{\sqrt{n}}\frac{1}{T}\mathsf{E}\int_0^T \Big(\sum_{i\in\mathcal{I}} h_i\cdot I_i^n(t,\pi_*^n) + \sum_{K\in\mathcal{K}} b^K\cdot B^{K,n}(t,\pi_*^n)\Big)\mathsf{d}t\\
&\le \limsup_{n\to\infty} \frac{1}{\sqrt{n}}\frac{1}{T}\mathsf{E}\int_0^T \Big(\sum_{i\in\mathcal{I}} h_i\cdot I_i^n(t,\pi^n) + \sum_{K\in\mathcal{K}} b^K\cdot B^{K,n}(t,\pi^n)\Big)\mathsf{d}t.
\end{aligned}
$$

Theorem 1 establishes the asymptotically optimal inventory policy. We briefly explain the main ideas in the proof of Theorem 1. The first step is to show that the (scaled) cost functions, i.e., $H(y_1, y_2)$ is convex, thus, $\eta_1(y)$ and η_2, the (scaled) inventory positions, are well-defined. Next, we show that the inventory replenishment

policy given by the theorem is coordinated, meaning that it is implementable and feasible. Obviously, the base-stock level of component-2 is feasible. We show the feasibility of the component-1's base-stock level by establishing that the function $\eta_1(y)$ is increasing and $\mathsf{d}\eta_1(y)/\mathsf{d}y \leq 1$. Finally, we need to show that series of the (scaled) inventory positions $\left(\frac{1}{\sqrt{n}}(IP_1^n(t) - n\lambda_1 L), \frac{1}{\sqrt{n}}(IP_2^n(t-\Delta) - n\lambda_2(L+\Delta))\right)$ converge to $(\eta_1(\eta_2 - \Delta\Theta^{12}(t)), \eta_2)$. This shows that the expected cost in the scaled system (2.27) is equivalent to the limit of the (scaled) expected lower bound of the system cost in (2.23).

A CBS policy requires recording the realized demand for product-12 in the last interval $(t-\Delta, t]$. If the realized demand is large, the chance of component-2 shortage is higher, correspondingly, we need to reduce component-1 order quantity to avoid excess component-1 inventory. When the leadtimes are identical, the CBS policy reduces to the IBS policy, i.e., $IP_1^n(t)$ and $IP_2^n(t)$ are kept at two constants for any $t > 0$ for the nth system, which requires no replenishment coordination.

Observe that we need to consider six different regions (discussed in Proposition 1) of the cost parameters and leadtime relations in the M-system. In some scenarios, compared with the N- and the W-system studied in Lu et al. (2015), the optimal CBS policy in the M-system is more complex due to the fact that each of two components is a common component and hence needs to coordinate between two products.

2.4.4 RBS Policy: Assembly Decomposition

Computing the optimal CBS policy requires solving a multistage stochastic program, which is notoriously difficult. Therefore, we consider a simple heuristic replenishment policy, RBS policy, which mimics the CBS policy. We take an assembly-decomposition approach illustrated in Figure 2.1(a). We assume that the inventory

replenishment of a component is managed separately in each assembly subsystem. Within each subsystem, the replenishment policy is similar to the balanced echelon base-stock policy (Rosling 1989, Chen and Zheng 1994). At time $t \geq 0$, the base-stock level of component-1 consists of two parts: a constant base-stock level r_1^1 in subsystem-1 and a dynamic base-stock level $r_{1l}^{12} \wedge (r_{1u}^{12} - \Delta\Theta^{12})$ in subsystem-12. The dynamic base-stock level in subsystem-12 assumes that the subsystem virtually reserves r_{1u}^{12} units of inventories by Δ time units earlier, and naturally, the base-stock level of component-1 should not exceed $r_{1u}^{12} - \Delta\Theta^{12}(t)$ currently. Meanwhile, to smooth the fluctuation incurred by the demand during the time window with the width Δ given by $\Delta\Theta^{12}$ over time, we cap the dynamic base-stock level in subsystem-12 by r_{1l}^{12}; The base-stock level of component-2 consists of two constant base-stock levels r_2^2 in subsystem-2 and r_2^{12} in subsystem-12.

As we analyze the scaled version of the nth system, we have $IP_1^n(t) = \sqrt{n}(r_1^1 + r_{1l}^{12} \wedge (r_{1u}^{12} - \Delta\widehat{D}^{12,n}(t))) + n\lambda_1 L$ and $IP_2^n(t) = \sqrt{n}(r_2^2 + r_2^{12}) + n\lambda_2(L+\Delta)$ at any time $t \geq 0$ in the RBS policy for the nth system. Here $\Delta\widehat{D}^{12,n}(t) = (\mathcal{D}^{12,n}(t-\Delta, t] - \lambda^{12}\Delta n)/\sqrt{n}$.

The inventory replenishment policy characterized in the RBS policy is well coordinated. This coordination feasibility is given by the following inequality:

$$\begin{aligned}
&\sqrt{n}r_{1l}^{12} \wedge \left(\sqrt{n}r_{1u}^{12} + n\lambda^{12}\delta - \mathcal{D}^{12,n}(t-\delta, t]\right) - \mathcal{D}^{12,n}(t, t+\varepsilon] \\
&\quad \leq \sqrt{n}r_{1l}^{12} \wedge \left(\sqrt{n}r_{1u}^{12} + n\lambda^{12}\delta - \mathcal{D}^{12,n}(t+\varepsilon-\delta, t+\varepsilon]\right) \text{ for } \varepsilon > 0.
\end{aligned}$$

Note that when $r_{1u}^{12} = \infty$, the above RBS policy is an IBS policy. Also, when all the leadtimes are identical, the RBS policy uses only constant base-stock levels $r_i^K, i \in K, K = 1, 2, 12$, hence the RBS policy becomes the IBS policy. Thus, we have the following theorem.

Theorem 2. (Properties of the RBS Policy) (i) *The inventory replenishment policy*

characterized in the RBS policy is well coordinated; (ii) *The cost under the optimal RBS policy is less than the one under the optimal IBS policy if the same allocation rule is used*; (iii) *The system is fully coordinated when we use the RBS policy and PRP rule*; (iv) *When the leadtimes are identical, the RBS policy degenerates to the IBS policy.*

The RBS policy has a similar but simpler structure than the CBS policy. The similarity means that we still keep using the demand information among the leadtime differences. The simpler structure means that the replenishment policy characterized by the minimizer of a convex cost function in (2.31) for the shorter leadtime is simplified as a parameterized piecewise linear function. Consider the N-system with component-1 having the shorter leadtime and let y_2 denote the on-hand inventory of component-2 at time $t + L$, Lu et al. (2015) show that the base-stock level of component-1 at time t has the following structure under symmetric cost at time t. (i) If y_2 is large, then the base-stock level of component-1 has a constant upper bound. (ii) If y_2 decreases, then the base-stock level of component-1 also decreases but at a slower rate. (iii) When y_2 further decreases, the inventory position of component-1 decreases at the same rate as y_2. We can see RBS policy exactly captures all the scenarios of the CBS policy in this case.

We now discuss how to compute the policy parameters of the RBS policy. One simple-minded approach would solve the assembly subsystems separately and aggregate their solutions, but this would forgo the pooling effect of the common components. Instead, we develop a linear programming heuristic to jointly determine the value of all parameters $\boldsymbol{r}$ in the entire system.

First, let $\boldsymbol{r}$ denote (scaled) parameters under the RBS policy $\boldsymbol{r} = (r_1^1, r_{1l}^{12}, r_{1u}^{12}, r_2^2, r_2^{12})$. Let Π_{RBS} be the set of inventory policies which consists of the PRP rule and RBS policy. Our goal is to find the optimal RBS policy $\pi_{\text{RBS}}^* \in \Pi_{\text{RBS}}$ that minimizes

the expected cost in the scaled system (2.27). This is equivalent to solve the following problem

$$\min_{\boldsymbol{r}\in\mathbb{R}^5_+} \quad \mathsf{E}[\varphi(r_1^1 + r_{1l}^{12} \wedge (r_{1u}^{12} - \Delta\Theta^{12}(t-L)) - \Theta_1(t), r_2^2 + r_2^{12} - \Theta_2(t) - \Delta\Theta_2(t-L))].$$

Because the demand is stationary, this problem becomes

$$\boldsymbol{P}_{\text{RBS}}\text{:} \quad \min_{\boldsymbol{r}\in\mathbb{R}^5_+} \quad \mathsf{E}[\varphi(r_1^1 + r_{1l}^{12} \wedge (r_{1u}^{12} - \Delta\Theta^{12}) - \Theta_1, r_2^2 + r_2^{12} - \Theta_2 - \Delta\Theta_2)].$$

Unfortunately, $\boldsymbol{P}_{\text{RBS}}$ is not jointly convex in $\boldsymbol{r}$ due to the fact that $r_{1l}^{12} \wedge (r_{1u}^{12} - \Delta\Theta^{12})$ is jointly concave in $(r_{1l}^{12}, r_{1u}^{12})$ for any $\Delta\Theta^{12}$. For this reason, we transform the problem into a convex program. Specifically, for any $\beta \in [0,1]$, we have

$$h_1\mathsf{E}[r_1^1 + r_{1l}^{12} \wedge (r_{1u}^{12} - \Delta\Theta^{12})] \leq h_1\mathsf{E}[r_1^1 + \beta r_{1l}^{12} + (1-\beta)(r_{1u}^{12} - \Delta\Theta^{12})],$$

where the inequality follows from facts that $x \wedge y \leq \beta x + (1-\beta)y$ for any $\beta \in [0,1]$. Then the following problem, $\boldsymbol{P}_{\text{RBS}-\beta}$ is an upper bound of $\boldsymbol{P}_{\text{RBS}}$.

$$\begin{aligned}\boldsymbol{P}_{\text{RBS}-\beta}\text{:} \quad \min_{\boldsymbol{r}\in\mathbb{R}^5_+} \quad & \mathsf{E}[\varphi(r_1^1 + r_{1l}^{12} \wedge (r_{1u}^{12} - \Delta\Theta^{12}) - \Theta_1, r_2^2 + r_2^{12} - \Delta\Theta_2 - \Theta_2) \\ & -h_1(r_1^1 + r_{1l}^{12} \wedge (r_{1u}^{12} - \Delta\Theta^{12})) \\ & +h_1(r_1^1 + \beta r_{1l}^{12} + (1-\beta)(r_{1u}^{12} - \Delta\Theta^{12}))]\end{aligned}$$

which is convex in $\boldsymbol{r}$. We then choose the value of β which leads to the lowest cost.

$$\min_{\beta\in[0,1]} \boldsymbol{P}_{\text{RBS}-\beta} \tag{2.36}$$

A natural way to solve $\boldsymbol{P}_{\text{RBS}-\beta}$ is to take derivatives and solve the first-order

conditions (FOC). Unfortunately, it appears to be difficult to derive the FOC, let alone solving it. Using the Monte Carlo method to solve the FOC would also be time-consuming because it requires calculating the sample mean for each parameter $\boldsymbol{r}$. To overcome this challenge, we use a Monte Carlo simulation-based solution approach, the SAA (Sample Average Approximation) method, to solve $\boldsymbol{P}_{\mathrm{RBS}-\beta}$. Under the SAA method, the expected value function is approximated by the corresponding sample average function. We generate a total number of Q samples for each random variable (See Theorem 5.18 in Shapiro et al. (2014) for the choice of Q to find an ε-optimal solution with portability $1-\alpha$. In practice, we decide the value of Q to achieve the balance between accuracy and computational complexity.). For each $\omega = 1, 2, \ldots, Q$, denote $\delta^K(\omega)$ and $d^K(\omega)$ to be the ωth realization of $\Delta\Theta^K$ and Θ^K, and $z^K(\omega)$ and $z_i(\omega)$ product-K backorders and component-i shortages under the ωth realization of the demand, respectively, $K = 1, 2, 12$ and $i = 1, 2$. Let $\boldsymbol{z}$ denote $\{z_i(\omega), i = 1, 2, z^K(\omega), K = 1, 2, 12, \omega = 1, \ldots, Q\}$. We relax the integer constraint here and consider rounding down the optimal base-stock levels to the nearest integers. The following is the linear programming formulation for the cost parameters $c^{12} \leq c^2 \wedge c^1$ below; the details can be found in Appendix A.3. The

formulations under other cost parameters can be similarly constructed.

$$
\begin{aligned}
\boldsymbol{LP}_{\mathrm{RBS}-\beta}\text{:} \quad & \min_{\boldsymbol{r},\ \boldsymbol{z}} \tfrac{1}{Q}\textstyle\sum_{\omega=1}^{Q} \Big[h_1[r_1^1 + \beta r_{1l}^{12} + (1-\beta)(r_{1u}^{12} - \delta^{12}(\omega))] \\
& \quad + h_2(r_2^2 + r_2^{12}) + c^{12} z^{12}(\omega)\Big] \\
\text{s.t.} \quad & z_1(\omega) \geq d^1(\omega) + d^{12}(\omega) - r_1^1 - r_{1l}^{12}, \\
& z_1(\omega) \geq d^1(\omega) + d^{12}(\omega) - r_1^1 - (r_{1u}^{12} - \delta^{12}(\omega)), \\
& z_2(\omega) \geq \delta^2(\omega) + \delta^{12}(\omega) + d^2(\omega) + d^{12}(\omega) - r_2^2 - r_2^{12}, \\
& z^{12}(\omega) \geq z_1(\omega), \\
& z^{12}(\omega) \geq z_2(\omega), \\
& \boldsymbol{r} \in \mathbb{R}_+^5, \boldsymbol{z} \in \mathbb{R}_+^{5Q}, \omega = 1, 2, \ldots, Q.
\end{aligned}
\tag{2.37}
$$

Then we consider a series of the values of β, and solve the corresponding $\boldsymbol{P}_{\mathrm{RBS}-\beta}$. Finally, we choose the β^* which minimizes $\boldsymbol{P}_{\mathrm{RBS}-\beta}$, and report the corresponding value of $\boldsymbol{r}$.

Note that the formulation for the IBS policy can be seen as a special case by setting $\beta = 1$, thus, (2.37) also provides a linear programming formulation to approximate the optimal base-stock level under an IBS policy.

When the leadtimes are identical, this linear programming formulation also generalizes the linear programming rounding heuristic in DeValve et al. (2020). DeValve et al. (2020) show that the inventory policy, which consists of an IBS replenishment policy derived by the linear programming rounding heuristic and an allocation principle algorithm that was initially proposed by Reiman and Wang (2015), is asymptotically optimal for the system with identical leadtimes when the identical leadtime is very long. We expect similar asymptotic optimality of the policy which consists of the IBS policy derived by $\boldsymbol{LP}_{\mathrm{RBS}-1}$ and PRP allocation principle for the system with identical leadtimes when the demand rates are very large.

2.5 General System: Heuristic Policy

In this section, we study the general system. Similar to Subsection 2.4.2, we construct a sequence of scaled general systems, indexed by n, in which all problem parameters remain the same as the original system, but the product demand rates are linear in n. All processes in the nth system are superscripted by n, and the product-K demand rate in the nth system has the same form as in (2.18). Our analysis focuses on the nth system with any given n. Similarly, we have

$$\Big(\frac{1}{\sqrt{n}}\sum_{K\in\mathcal{K}_1}\Big(\mathcal{D}^{K,n}(t-L_1,t]-n\lambda^K L_1\Big),\ldots,\frac{1}{\sqrt{n}}\sum_{K\in\mathcal{K}_m}\Big(\mathcal{D}^{K,n}(t-L_m,t]-n\lambda^K L_m\Big)\Big)$$

converge to a stationary zero-mean multivariate normal vector $\mathbf{\Theta}=(\Theta_1,\ldots,\Theta_m)$ with the $m\times m$ covariance matrix $\Gamma=(\Gamma_{ij})_{m\times m}$ with $\sqrt{\Gamma_{ij}}=\sqrt{\sum_{K\in\mathcal{K}_i,\hat{K}\in\mathcal{K}_j}(\sigma^{K,\hat{K}})^2}\cdot\sqrt{L_i\wedge L_j}$. We adopt the PRP rule as the allocation policy, which is very similar to that in the M-system in Section 2.4.2, so we omit the details here for brevity. The focus then is on the replenishment policy. In particular, we decide the levels of component inventory positions $\mathbf{IP}^n(t,\pi)=(IP_1^n(t,\pi),\ldots,IP_m^n(t,\pi))$ at any given time $t\geq 0$ to minimize the expected cost in the scaled system, $\mathsf{E}\varphi(IP_1^n(t,\pi)-\Theta_1,\ldots,IP_m^n(t,\pi)-\Theta_m)$. Similar to (2.14), $\varphi(\mathbf{IP}^n(t,\pi)-\mathbf{\Theta})$ is defined as

$$\begin{aligned}\varphi(\mathbf{IP}^n(t,\pi)-\mathbf{\Theta}) \;=\; & \min_{y^K:K\in\mathcal{K}}\Big[\sum_{i\in\mathcal{I}}h_i\cdot(IP_i^n(t,\pi)-\Theta_i)+\sum_{K\in\mathcal{K}}c^K\cdot y^K\Big], \quad (2.38)\\ \text{s.t.}\quad & \sum_{K\in\mathcal{K}_i}y^K\geq[IP_i^n(t,\pi)-\Theta_i]^-,\ i\in\mathcal{I}\\ & y^K\geq 0,\ K\in\mathcal{K}.\end{aligned}$$

The extension of the asymptotic optimality of the CBS turns out to be very difficult; see the online Appendix A.4 for detail. Hence, we do not consider this

policy for a general system. In Subsection 2.5.1, we extend the RBS policy and demonstrate that it is too complex to be of practical value for large systems. For this reason, in Subsection 2.5.2, we introduce the IBS policy for large-scale systems and develop newsvendor-type closed-form solutions for effective policy parameters.

2.5.1 Extension of the RBS Policy

We generalize the RBS-policy to the general system using the assembly decomposition. The base-stock level of component-i is the sum of the component base-stock levels in each subsystem at time $t \geq 0$. Specifically, for subsystem-K determined by product-K, the leadtimes of the components in K are denoted by $\{L_i^K, i \in K\}$ with $L_1^K \leq L_2^K \leq \cdots \leq L_{|K|}^K$; Component-$i$ with $L_i = L_{|K|}^K$ is controlled by a constant base-stock level r_i^K, while component-i with $L_i < L_{|K|}^K$ is controlled by dynamic base-stock level $r_{il}^K \wedge (r_{iu}^K - \Delta\Theta^K(L_{|K|}^K - L_i))$ where $\Delta\Theta^K(L_{|K|}^K - L_i)$ is a zero-mean normal with variance $(\sigma^K)^2(L_{|K|}^K - L_i)$. Noting that $\Delta\widehat{D}^{K,n}(L_{|K|}^K - L_i) = (\mathcal{D}^{K,n}(t-(L_{|K|}^K - L_i), t] - \lambda^K(L_{|K|}^K - L_i)n)/\sqrt{n}$ converges to $\Delta\Theta^K(L_{|K|}^K - L_i)$, similar to the M-system, for the nth system, component-i with $L_i = L_{|K|}^K$ is controlled by a constant base-stock level $\sqrt{n}r_i^K + n\lambda^K L_{|K|}^K$, and component-$i$ with $L_i < L_{|K|}^K$ is controlled by a dynamic base-stock level $\sqrt{n}(r_{il}^K \wedge (r_{iu}^K - \Delta\widehat{D}^{K,n}(L_{|K|}^K - L_i))) + n\lambda^K L_i$. To summarize, the following is a formal definition of the RBS Policy for the general system:

Definition 4. (RBS Policy) *We say that an admissible policy* $\pi^n = \{(O_i^n(t), A^{K,n}(t), i \in \mathcal{I}, K \in \mathcal{K}), t \geq 0\} \in \mathcal{A}^n$ *is an RBS policy in the* n*th system if the following*

conditions are satisfied for all $t \geq 0$ *and* $i \in \mathcal{I}$,

$$IP_i^n(t,\pi) = \sum_{K\in\mathcal{K}_i}\Big[\sqrt{n}\Big(r_{il}^K \wedge (r_{iu}^K - \Delta\widehat{D}^{K,n}(L_{|K|}^K - L_i))\cdot \mathbf{I}_{\{L_i\neq L_{|K|}^K\}} + r_i^K\cdot \mathbf{I}_{\{L_i = L_{|K|}^K\}}\Big) + n\lambda^K L_i\Big],$$

where $\{r_{il}^K, r_{iu}^K, r_i^K, i\in\mathcal{I}, K\in\mathcal{K}\}$ *are the policy parameters and* $\mathbf{I}_{\{\cdot\}}$ *is an indicator function.*

It is intuitive that Theorem 2 also holds for RBS policy in the general system. To facilitate the understanding of the practicality of the RBS policy, we consider the general M-system introduced in Section 3. According to Definition 4, we have $m + 2(m-1) + 1$ parameters for an arbitrary RBS policy. For the first step in the linear program given by (2.11)-(2.13), we can find the objective value function (see Appendix A.2). When moving to the second step to optimize these $m+2(m-1)+1$ parameters, the objective value function is not convex, which makes it difficult to solve the optimal RBS policy parameters. In the next subsection, we take a different approach to develop a much simpler but highly scalable replenishment policy for general systems.

2.5.2 IBS Policy: Distribution Decomposition

As observed in the above, the complexity of determining the optimal CBS and RBS policies mainly comes from the multistage stochastic program. To overcome this difficulty from the sequential conditional expectations, we use the average demand from the leadtime differences to approximate the random demand. This approximation gives us a simpler replenishment policy, an IBS policy.

Under the IBS policy with base-stock levels $s_1, \ldots, s_m$ in the scaled system, the

component inventory position in the nth system is kept at the constant base-stock level

$$IP_i^n(t,\pi) = \sqrt{n}s_i + n\lambda_i L_i, \quad i \in \mathcal{I}. \tag{2.39}$$

The expected cost in the scaled system is

$$\min_{\boldsymbol{s}\in\mathbb{R}_+^m} \quad \mathsf{E}\varphi(s_1 - \Theta_1, \ldots, s_m - \Theta_m). \tag{2.40}$$

Observe that the objective function in (2.40) is convex. Thus the solution to the multistage stochastic program exists but may be difficult to solve when the system size gets large or the structure of the system becomes complicated. Now we use a *distribution decomposition approach* (illustrated in Figure 2.1(b)) to construct a simpler IBS policy.

We focus on subsystem-i instead of the entire system. The following linear program solves the conditional sample-path lower bound in the subsystem-i

$$\min_{y^K, K\in\mathcal{K}_i} \sum_{K\in\mathcal{K}_i} (b^K + h_i)\cdot y^K \tag{2.41}$$

$$\text{s.t.} \quad \sum_{K\in\mathcal{K}_i} y^K \geq [s_i - \Theta_i]^-,$$

$$y^K \geq 0, K \in \mathcal{K}_i. \tag{2.42}$$

Clearly, the solution to the above linear program (2.41) takes a simple form: let

$$b_i \equiv \min\{b^K, K \in \mathcal{K}_i\}, \quad i \in \mathcal{I}. \tag{2.43}$$

If product-$\hat{K}$ has a backorder cost equal to b_i, then the solution to (2.41) is given

by $y_*^{\hat{K}} = [s_i - \Theta_i]^-$ and $y_*^K = 0$ for $K \neq \hat{K}$ and $K \in \mathcal{K}_i$. (2.40) in this subsystem is changed into

$$\min_{s_i \geq 0} \mathsf{E}\Big(h_i(s_i - \Theta_i) + (b_i + h_i)(s_i - \Theta_i)^-\Big). \tag{2.44}$$

Thus, deriving the optimal base-stock level in the nth distribution subsystem-i is equivalent to solving a newsvendor problem with holding cost h_i and shortage cost b_i. Let $\Phi_i(\cdot)$ be the cdf of Θ_i, then the corresponding optimal base-stock level in the nth subsystem-i can be computed as

$$s_i = \Phi_i^{-1}\left(\frac{b_i}{b_i + h_i}\right), \quad i \in \mathcal{I}. \tag{2.45}$$

This shows that the distribution decomposition effectively reduces the original system into multiple serial systems, each with one component and the cheapest product that requires it, which is equivalent to the newsvendor problem. Note that we only need to solve m independent newsvendor problems. Thus, this approach is the *most efficient* and highly scalable heuristic to a large-scale system. We transform the notoriously challenging multistage stochastic program into a computationally efficient and scalable linear program. We expect that the distribution decomposition becomes more effective when the importance of coordinating the component replenishment is not high. For example, if the product backorder costs are much higher than the component holding costs, then the component is less likely to be out-of-stock, which makes the distribution decomposition more favorable.

The IBS policy developed via the distribution decomposition approach can also be regarded as a special case of a CBS policy. The IBS policy implies that we do not consider the inventory status of other components. Thus, when we analyze the inven-

tory position of component-i, we assume other components have a sufficiently large inventory. We use the M-system as an example: the base-stock level of complement-1 is $\lim_{y_2\to\infty}\sqrt{n}\eta_1(y_2)+n\lambda_1 L$. To see this, for instance, consider $c^2\vee c^1 < c^{12} < c^1+c^2$, and we have

$$\begin{aligned}\eta_1(y_2) = \max\{y_1:\ & h_1 - c^1\cdot \mathsf{Pr}(y_1\le \Theta_1, y_1\le y_2+\Theta_1-\Theta_2-\Delta\Theta^2)\\ & -(c^{12}-c^2)\cdot\mathsf{Pr}(y_1\ge y_2+\Theta_1-\Theta_2-\Delta\Theta^2, y_1\le\Theta_1)\le 0\},\\ \lim_{y_2\to\infty}\eta_1(y_2) = \max\{y_1:\ & h_1 - c^1\cdot\mathsf{Pr}(y_1\le\Theta_1)\le 0\} \quad = s_1 \ \text{ (in view of (2.45)).}\end{aligned}$$

The same results can be verified for other cost parameters. These results are intuitive because when we analyze one single component and assume that other components have inventory close to infinity, then this system collapsed to a distribution system: the shortage can only occur on this single component and the backorder can only occur on the products assembled by this component.

At the same time, the IBS policy given by (2.39) with s_i determined by (2.45) through the distribution decomposition approach becomes the policy determined by the constant makeup newsvendor heuristic from DeValve et al. (2020) when the lead-times and the ratio between b_i and h_i cross different components are identical. Hence it directly follows from their Propositions 6 and 8 in DeValve et al. (2020) that the system cost incurred by the IBS policy given by (2.39) with s_i defined by (2.45) provides the asymptotically approximation factor performance guarantee for the optimal system cost.

Next, we develop useful properties of the optimal IBS policy. We first show that (2.40), the expected cost in the scaled system under the optimal IBS policy, is bounded from both above and below through the primal-dual analysis. We introduce a *shadow price* $\boldsymbol{v} = \{v_1,\dots,v_m\}$ for the first constraint of the linear program (2.11)-

(2.13) and formulate the *dual problem* to the linear program (2.11)-(2.13) as

$$\begin{aligned} \max_{\boldsymbol{v}} \quad & \sum_{i\in\mathcal{I}} v_i \cdot [N_i(t,\pi)]^- \\ \text{s.t.} \quad & v_i \geq 0, \quad i \in \mathcal{I}; \\ & \sum_{i\in K} v_i \leq c^K, \quad K \in \mathcal{K}. \end{aligned} \tag{2.46}$$

By selecting two sets of values of $\{v_i : i \in \mathcal{I}\}$, we construct two newsvendor-type bounding systems which both consist of m independent newsvendor problems. Namely, let $v_i^\ell = 2h_i$, $v_i^u = \min\{c^K : K \in \mathcal{K}_i\}$, $i \in \mathcal{I}$, $\boldsymbol{s} = \{s_1, \ldots, s_m\}$, and

$$C^l(\mathbf{s}) = \mathsf{E}\left[\sum_{i\in\mathcal{I}} h_i \cdot (s_i - \Theta_i)^+ + \sum_{i\in\mathcal{I}} (v_i^\ell - h_i) \cdot (s_i - \Theta_i)^-\right], \tag{2.47}$$

$$\mathbf{s}^l = \arg\min\{C^l(\mathbf{s})\},\ \underline{C}^* = C^l(\mathbf{s}^l), \tag{2.48}$$

$$C^u(\mathbf{s}) = \mathsf{E}\left[\sum_{i\in\mathcal{I}} h_i \cdot (s_i - \Theta_i)^+ + \sum_{i\in\mathcal{I}} (v_i^u - h_i) \cdot (s_i - \Theta_i)^-\right], \tag{2.49}$$

$$\mathbf{s}^u = \arg\min\{C^u(\mathbf{s})\},\ \overline{C}^* = C^u(\mathbf{s}^u), \tag{2.50}$$

where $v_i^u - h_i$ and $v_i^\ell - h_i$ reflect the shortage cost of component-i. The following theorem establishes the upper and lower bounds for the expected cost in the scaled system under the optimal IBS policy in (2.40).

Theorem 3. (Upper and Lower Bounds for the Optimal IBS policy) *Assume that* $b^K \geq \sum_{i\in K} h_i$ *holds, then the expected cost in the scaled system* (2.40)*, under the optimal IBS policy, is bounded below by* $\underline{C}^*$ *and above by* $\overline{C}^*$*.*

Theorem 3 shows that the expected system cost in the scaled system under the optimal IBS policy are bounded by two simple systems. The bounds could be easily

computed by solving multiple independent newsvendor-type problems, in particular,

$$s_i^l = \Phi_i^{-1}\Big(\frac{v_i^\ell - h_i}{v_i^\ell}\Big), \quad s_i^u = \Phi_i^{-1}\Big(\frac{v_i^u - h_i}{v_i^u}\Big), \quad i \in \mathcal{I}. \tag{2.51}$$

The condition $b^K \geq \sum_{i\in K} h_i$ says that the backorder cost of a product is no less than inventory holding cost for its components, which naturally holds in practice. Establishing easy-to-compute bounds provide a benchmark for evaluating the effectiveness of the IBS type of heuristics for the general system, where the CBS and RBS policies are extremely challenging to compute. Also, the values of $\boldsymbol{s}^l$ and $\boldsymbol{s}^u$ provide good starting points to compute the base-stock levels.

The optimal solution $\{v_i^*, i \in \mathcal{I}\}$ in linear program (2.46) can also be approximated by $\hat{v}_i = h_i + b_i$. The resulting newsvendor solutions are then equivalent to the one from the distribution decomposition approach. This helps us estimate the expected cost in the scaled system under our distribution decomposition approach by the following function:

$$\min_{\boldsymbol{s}} \sum_{i\in\mathcal{I}} \mathsf{E}\Big(h_i \cdot (s_i - \Theta_i) + \hat{v}_i \cdot (s_i - \Theta_i)^-\Big). \tag{2.52}$$

Denoting this estimation as C^{NV}, the following theorem shows that C^{NV} is also bounded between $\underline{C}^*$ and $\overline{C}^*$.

Theorem 4. (Bounds for Distribution Decomposition) *Assume that $b^K \geq \sum_{i\in K} h_i, K \in \mathcal{K}$ holds, the estimated expected cost in the scaled system under distribution decomposition are bounded, specifically, $C^{\mathrm{NV}} \in [\underline{C}^*, \overline{C}^*]$.*

2.6 Numerical Studies

In this section, we conduct extensive numerical tests on the effectiveness and efficiency of the proposed heuristics in both simple and general systems, and also to gain insights into the impact of the system parameters on the performance of the RBS-policy. Recall that the PRP rule is adopted as the allocation rule, so our focus is on evaluating various heuristics for computing the heuristic replenishment policy. In particular, we evaluate the IBS policy with parameters determined by 1) the newsvendor decomposition given in (2.45) as "NV"; 2) the constant markup newsvendor decomposition heuristic as "CM" from DeValve et al. (2020); 3) the linear programming rounding heuristic as "RD" from DeValve et al. (2020). For simple systems, we also assess the RBS policy determined by the assembly decomposition and the linear programming (LP) heuristic (2.37), denoted as "RBS". For the computation of the policy parameters in NV and RBS, we select $n = 25$.

We use MATLAB 2017a as the platform and the YALMIP R20160930 and MOSEK 8.0 as the solvers. For the computation of heuristic policy parameters which uses the SAA approach (e.g., RBS and LR), we set $Q = 1,000$. For the evaluation of each heuristic policy $\mathsf{H} \in \{\text{NV,CM,RD,RBS}\}$, we carry out 100 simulation runs, each with a period of 10 years and components are allocated following the PRP rule daily, to evaluate the system cost under a parameter set and compute the corresponding average simulated cost C_{H} among the 100 runs. The first 60 days are used for warm-up. We test the robustness of the sample size Q and simulation run with different values and find the standard error of the system cost is less than 0.1, indicating that the results are robust. We also assume the Poisson demand processes of all products in this section and the demand process of each product is independent of the other products.

2.6.1 Performance of the Heuristic Policy Relative to the Expected Lower Bound

We first measure the performance of each heuristic by the optimality loss relative to the expected lower bound of the system cost under the optimal CBS policy C_{CBS^*}:

$$\Delta_{\mathsf{H}}\% = \frac{C_{\mathsf{H}} - C_{\text{CBS}^*}}{C_{\text{CBS}^*}} \times 100\%, \quad \mathsf{H} \in \{\text{NV,CM,RD,RBS}\}. \tag{2.53}$$

Each heuristic replenishment policy and the benchmark is evaluated by the average simulated system cost under the heuristic H. To have an unbiased estimator of C_{CBS^*}, we compute the average sample-path lower bound cost under the optimal CBS policy for $10,000$ samples of the set of random variables.

We consider an M-system which consists of three types of computers: high-end, mid-end, and low-end computers. The components are two graphic cards. The holding costs are $h_1 = 0.48, h_2 = 0.34$ (\$/day), which are estimated by 25% of components' sale price annually. The leadtimes are $L_1 = 4, L_2 = 10$ (days). The demand rates are $\lambda^1 = \lambda^2 = 3, \lambda^{12} = 24$ (unit/day) and the backorder costs are $b^1 = 1, b^2 = 3.8, b^{12} = 1.2$ (\$/day). Table 2.1 shows optimality loss in each heuristic. We find that the optimality loss of the RBS policy is 5.06%, which implies a good performance of our policy even when demand rates are not large. We also find RBS has a better performance than all the other IBS heuristics in this example.

Table 2.1: Computational Results for IBS- and RBS-Policies

Policy	Heuristic	Heuristic Policy Parameters			Performance		Optimality Loss
		IP_1		IP_2	C_{H}	CPU time (sec)	$\Delta_l\%$
IBS	NV	113		283	13.38	0.00	8.38
	CM	110		274	14.52	0.00	17.67
	RD	111		282	13.34	0.55	8.07
		IP_1 :		IP_2			
		$r_1^1 + r_{1l}^{12}$	$r_1^1 + r_{1u}^{12}$	$r_2^{12} + r_2^2$			
RBS	LP	111	264	280	12.97	6.62	5.06

Computing C_{CBS^*} is difficult, especially when the system is large. To have a thorough test of the proposed decomposition approaches, we then evaluate their performances using a different performance metric.

2.6.2 Performance of the Heuristic Policy Relative to the Benchmark Algorithm

In this subsection, we evaluate the performance of each heuristic relative to the RD heuristic, which approximates the optimal IBS policy well.

Each heuristic is evaluated by cost savings relative to the RD heuristic, which is defined as

$$\Delta_{\mathsf{H}}\% = \frac{C_{\mathsf{H}} - C_{\text{RD}}}{C_{\text{RD}}} \times 100\%, \quad \mathsf{H} \in \{\text{NV,CM,RBS}\}. \tag{2.54}$$

We then do the comparison in two specially structured system and two general systems.

2.6.2.1 Specially Structured Systems

For the RBS heuristic, we run simulations for various weight β in the set of $\{0, 0.1, \ldots, 1\}$ and select the weight that has the lowest system cost in the report.

The first simple system we consider is an M-system with the same holding cost and leadtimes parameters as in Table 2.1. To cover various situations of demand asymmetry and cost parameters, we examine the performance of various heuristics under 30 sets of demand rates $\lambda^1, \lambda^2, \lambda^{12}$ and backorder costs b^1, b^2, b^{12}.

The second simple system is a W-system which consists of two types of computers: high-end and low-end computers. The components are two graphic cards and one

processor. We examine the performance of various heuristics under 12 sets of demand rates $\lambda^{13}, \lambda^{23}$ and leadtimes L_1, L_2, L_3.

Performance perspective. We summarize the performance of the heuristics in Table 2.2. The detailed performance of different heuristics as well as the system parameters are shown in Appendix A.5.1. First, we find that the NV has a similar average performance to the benchmark in both the M- and the W-system. In particular, compared to the benchmark RD, the NV results in average Δ_{H} (std) 0.40% (1.06%) in the M-system and 0.50% (0.23%) in the W-system. The CM on average has a much higher cost compared to the benchmark. Second, while the RBS has a better performance than the benchmark in the W-system (on average saves 1.23% with std 1.11% and maximum 3.64%), they perform similarly in the M-system: On average, the RBS saves 0.15% with std 1.38% relative to the benchmark.

We also investigate the condition when the RBS policy is more effective than the IBS policy. We do so by examining the complete numerical results in Appendix A.5.1 (Recall Table 2.2 only reports the summary statistics). In the M-system, the RBS policy becomes more effective when the backorder cost for the common product (product-12) is small and demand volatility for this product is big. This is consistent with the results in Proposition 1, as the component-2 that has a shorter leadtime is more likely to be out-of-stock and synchronization becomes more valuable when the demand of product-12 during the leadtime difference is larger and more variable; In both the M- and the W-system, the RBS policy is more effective when the average product service level coefficient is smaller due to the higher chance for the component to be out-of-stock.

Computation perspective. We also compare the efficiency of each heuristic in Table 2.2 for both the M- and the W-system. The computational time of the RD is short (0.38 seconds for the M-system and 0.36 seconds for the W-system). The

Table 2.2: Heuristic Performance

Heuristic	M-system			W-system		
	Average Δ_H%	Best Δ_H%	Time (sec)	Average Δ_H%	Best Δ_H%	Time (sec)
NV	0.40	-1.15	0.00	0.50	0.2	0.00
CM	8.83	1.44	0.00	8.60	4.58	0.00
RD	-	-	0.38	-	-	0.36
RBS	-0.15	-4.9	2.74	-1.23	-3.64	10.17

NV and CM are the most efficient in computation (0.00 for both the M- and the W-system), and the RBS is the slowest (2.74 seconds for the M-system and 10.17 second for the W-system).

To conclude this subsection, we recommend the RBS heuristic together with a PRP allocation rule in small-scale simple systems.

2.6.2.2 General Systems

We examine the effectiveness and efficiency of the above proposed heuristics to a general system in a *PC assembly system* example with 15 components and 6 products in Cheng et al. (2002). We adopt the recent price of the latest computer components, which are illustrated in Table A.3 in Appendix A.5.2 together with the BOM of the system and the component leadtimes.

We set the backorder cost based on a constant product service level coefficient θ in the newsvendor problem: $b^K = \sum_{i \in K} h_i \cdot (1-\theta)/\theta$, and vary $\theta \in \{0.01, 0.05, 0.1, 0.15, 0.2\}$. We also consider three different product demands

$$(\lambda^1, \lambda^2, \lambda^3, \lambda^4, \lambda^5, \lambda^6) \in \{(10,10,10,10,10,10), (7,7,9,9,14,14), (14,14,9,9,7,7)\}.$$

The performance and computational time are reported in Table 2.3. Compared to the RD heuristic, the CM heuristic is efficient, but with a much higher cost; the NV heuristic is similarly efficient as the CM heuristic, with only a 2.29% higher cost

Table 2.3: Performance of Selected Algorithms on Small and Large Networks

Heuristic		NV	CM	NV	CM	RD
	Q	$\Delta_{\mathsf{H}}\%$		Time (sec)		
PC Assembly System	1,000	2.29	24.72	0.00	0.00	0.71
Maintenance Organization	1,000	-2.14	340.88	0.02	0.01	7.82
	5,000	-0.94	346.28	0.01	0.01	41.37
	10,000	-0.77	347.05	0.01	0.01	85.41
	20,000	-0.76	347.11	0.01	0.01	172.28
	50,000	-0.77	347.07	0.02	0.01	485.21

than the RD heuristic on average.

We further examine these heuristics in a *maintenance organization problem* with 110 components and 3 products studied in van Jaarsveld and Scheller-Wolf (2015). There are 3 repair types a, b, c. Each component has usage probabilities p_a, p_b and p_c in each of the repair types. For every arriving repair of a given type, it uses each spare part with the probability prescribed in Table A.4 in Appendix A.5.2, independent of the usage of other spare parts. We keep the original holding cost and leadtimes as in their settings, which is also summarized in Table A.4. The backorder cost is set using the same rule as the PC assembly example with identical product service level coefficient $\theta = 0.2$. The demand rates of the three repair types are $13, 10, 35$, respectively. Using $Q = 1,000$ becomes insufficient due to the scale of the system. We select various Q as shown in Table 2.3.

The RD heuristic requires a large sample size Q greater than 5,000 to have a comparable performance compared to the NV heuristic. Increasing Q also increases the computation time at a similar rate. The CM is computational scalable, but it is very ineffective. This is expected because the maximum number of components required by any product in the system, $\breve{m}$, may be an obstacle for the newsvendor decomposition approach (DeValve et al. 2020).

We also observe that this maintenance organization problem has a similar structure to the application of e-commerce. As mentioned in the introduction, an online seller offering the subscription box service is one such example. The seller needs to send all items in one box, and different customer types will select different items randomly based on their preference. This system can be extremely large due to the various choices offered online: the number of items can be several million. Our NV heuristic would be even more attractive to deal with this kind of large-scale system in e-commerce.

Based on our extensive numerical tests, we recommend the adoption of the NV together with the PRP rule for general large-scale systems.

2.7 Conclusion

In this chapter, we study inventory policies for continuous-time review ATO systems with general leadtimes. First, we characterize an asymptotically optimal inventory policy for the M-system, which has three products sharing two common components. The policy consists of a periodic review priority (PRP) allocation rule and a coordinated based-stock policy (CBS) replenishment policy. This result is a non-trivial extension of the results in the literature on the N- and the W-system which contain only one single common component. Due to the difficulty of computing the asymptotically optimal inventory policy, we first develop an approach to decompose the system into a set of single-product assembly subsystems, one for each product. Then we develop a reserved-base stock (RBS) policy, which reserves component inventory similar to the CBS policy with a simple structure, and a linear programming heuristic to solve the policy parameters. To our knowledge, both the RBS policy and the assembly decomposition approach is new to the literature.

However, it is difficult to extend either the CBS or the RBS policy to a general system with a large size of components and products. We next develop another novel decomposition approach that is highly scalable to a large-scale system, which decomposes the system into a set of single-component distribution subsystems, one for each component. We show that each distribution subsystem is equivalent to a newsvendor problem with a closed-form solution. We then solve a series of newsvendor problems to derive the corresponding IBS policy. When restricted to IBS policy, our policy is comparable or outperforms the benchmarks proposed in the literature. In addition to the computational advantage, it provides insight into key determining factors of the inventor policy.

Our research can be extended in several ways. For example, our RBS policy is too complex to implement on a large-scale system. However, one can simplify the RBS policy which only coordinates the replenishment of several key components. It is interesting to investigate which components to synchronize and their impact on the system cost. Other possible extensions include showing the exact performance guarantee for IBS policy heuristics for both identical and non-identical leadtimes scenarios; developing similar heuristics for ATO system with random leadtimes.

Chapter 3

Data-Driven Scalable E-commerce Transportation Network Design with Unknown Flow Response

3.1 Introduction

3.1.1 Background and Objective

This chapter studies the optimization of transportation cost for e-commerce marketplaces. With digital development exploding, such as ubiquitous smartphones, social media, shopping apps, digital payment systems, and cloud-based computing, online shopping has experienced rapid growth globally. In 2019, it reached a new high, overtaking a major part of retail for the first time. It accounted for $4.2 billion in sales on Thanksgiving in the United States, a 14.5% increase from 2018 (Mitra 2019). In China, Alibaba's Single's Day recorded $38.3 billion online sales, 25% higher than sales in 2018 (Yu 2019). It is expected that e-commerce's share of retail sales in the United States will rise from 8.9% in 2017 to over 15% in 2022, reaching $892 billion (CBRE 2019). Worldwide, e-commerce's share is expected to reach 22 percent in 2023 (Statista 2019).

Consumers favor online shopping because online marketplaces offer almost unlimited product choices and, at the same time, fast and cheap (sometimes free) delivery services, bringing them instant gratification. For example, Amazon in the United States offered over 562 million different products in January 2018 (ScrapeHero 2018).

To win orders and create an entry barrier, Amazon is famous for the free two-day shipping service provided to its prime members and now it begins to offer free one-day shipping (Schoolov 2019). JD.com, Amazon's Chinese equivalent, differentiates itself by introducing the 211 program: orders placed by 11 am will be delivered on the same day, and orders placed by 11 pm will be delivered on the next day. Obviously, the growing online shopping trend will not lower customers' expectations on both fronts. In fact, the growth of online sales has further raised customers' demand for quick delivery speed. For instance, in 2018, Amazon's prime members ordered more than 2 billion products for one-day or faster delivery (Corbett 2018).

In the early days, most e-commerce marketplaces relied on third-party logistic providers (3PLs) for package delivery, which saved the investment in logistic infrastructure at the expense of losing full control of the delivery system. For example, after Christmas in 2013, Amazon was forced to cancel orders and refund customers for many of its packages that UPS failed to deliver on time (Levs 2013). Drawing lessons from such experiences, some online marketplaces, including Amazon and JD.com, have developed an in-house transportation network to gain direct control of packages delivery. The associated transportation network cost, however, is considerable: it accounts for 14.2% of total operating expenses for Amazon in 2017 (Amazon.com, Inc 2018) and 7.1% (JD.com, Inc 2018) for JD.com. To stay competitive, online marketplaces need to operate their transportation networks in the most cost-efficient way. With a plethora of customer order and delivery data available from the online shopping platform, it is also important to know how the information can be exploited to optimize the transportation network.

Motivated by our experience with an e-commerce marketplace, the objective of our study is to develop data-driven, scalable optimization tools for such companies to best utilize their in-house outbound transportation networks. We focus on reduc-

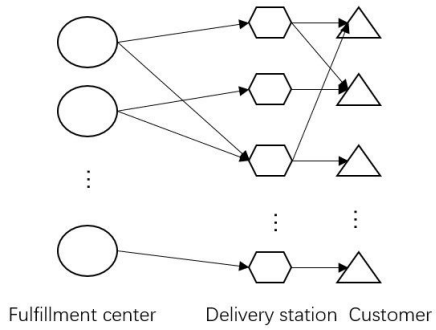


Figure 3.1: Outbound Transportation Network for E-commerce Marketplace.

ing the middle-mile outbound transportation costs, explained in Section 1.2 below. At first glance, the optimization problem appears to resemble some elements of the fixed-charge capacitated multicommodity network design (CMND) problem, but it departs from the classic problem with significant nuances and challenges. The novelty here is due to the very feature of the granular customer order and delivery in online shopping (see Section 1.3 for more detail)– the so-called long-tail phenomenon, which necessitates machine learning techniques to predict the flow response for any given network design. These challenges can also appear in omni-channel retailing, in particular the buy-online ship-to-store (BOSS) used by many omni-channel retailers, such as Walmart, Target, etc. The omni-channel retailers first ship BOSS products from warehouses to e-fulfillment centers (middle-mile) and then distributed to local brick-and-mortar stores for customers to pick up (last mile). In general, our methods are helpful for retailers with online channels and direct control of the middle-mile transportation network.

3.1.2 Middle-mile Transportation Network

The outbound transportation network of the online marketplace we consider is illustrated in Figure 3.1. Its inventories are stored in regional warehouses, called

fulfillment centers (FCs), usually located in rural areas for inexpensive real estate costs. The next tier in the network consists of the smaller delivery stations (DSs) in urban areas, closer to the end customers. Based on the customer order, the company first decides the first leg of the transportation network in Figure 3.1, i.e., how and from which FC to ship the package to which DS, a process called the middle-mile transportation. After packages reaching their destination DSs, they are then shipped to customers' doors. This process is called the last-mile transportation (the second leg in Figure 3.1). The middle-mile network is less known than the last-mile network because it is invisible to the customers, but its importance cannot be overlooked. According to Naughton and Boyle (2019), the middle-mile network can be the most expensive part of the whole supply chain, and its market can reach $1 trillion. In this chapter, we focus on optimizing the middle-mile transportation network.

The majority of the middle-mile network cost comes from two main shipping options: (1) the fixed per truck line-haul cost for the shipment using the company's own truck fleet or its partner truck carriers. To achieve timely delivery, the truck travels from the origin to the destination every day regardless of the actual volume being loaded as long as it is within the truck capacity. Thus, we assume the line-haul cost is independent of the number of packages loaded in the trailer. (2) the shortfall cost: each product that is not shipped by the in-house truck due to unavailable truck capacity is shipped by 3PLs, which incurs a unit cost. Because of the limited availability of truck routes, the truck capacity constraint, the inventory constraint at the FCs, the fulfillment resources constraint and unpredictable demand volume, the shortfall cost is non-trivial. For example, around 50% of Amazon US volumes are covered by 3PLs (Herrera 2019). These two costs are influenced by two decisions made in separate stages.

At the first stage, a *network configuration* decision is made, specifying whether

to assign a truck working between any given pair of FCs and DSs in the middle-mile network. For example, Amazon has a "middle-mile providers" program, where truck carriers are hired to haul Amazon freight. Such a decision is made once every week or every other week, partly due to contract arrangements, and partly due to the time needed to reposition trucks and drivers. The network configuration decision determines the line-haul cost.

At the second stage, given the first-stage network configuration decision, a *fulfillment policy* is formed to deploy the network. This policy specifies how an individual customer order is fulfilled in real time: from which FC to pick up the item, using which shipment method, and at what time to be loaded into the trailer, etc. The resulting *network flow* is the *total* number of products shipped by the in-house truck shipment under this network configuration, before the next network decision. Similarly, the resulting *network shortfall* is the *total* number of products shipped by the 3PLs under this configuration. Both the network flow and shortfall are thus in effect the second-stage *flow response* to the first-stage network decision. The flow response determines the shortfall cost.

3.1.3 Unknown Flow Response

In classic transportation problems that model the bricks-and-mortar operations, both the network configuration and the fulfillment policy are typically made jointly by a single decision-maker in the same time period, given the forecasted demand and supply for the entire period. The reality faced by the middle-mile manager in an online marketplace, however, is very different from the traditional setting. In this new context, at the time when the middle-mile manager decides the network configuration, the flow response to the configuration is unknown to him/her. This presents a unique challenge to our research.

Unknown flow response happens because the company has a decentralized decision-making structure. The middle-mile manager only decides the network configuration, while the network flow and shortfall are decided by the fulfillment policy developed by another decision entity, which acts as a black-box in the middle-mile manager's view.

This practice makes sense because the decisions are made on different time scales and likely with different objectives. The network configuration is changed once every week or every other week with the objective of cost minimization, using information such as demand distribution of each product at each DS, inventory level of each product at each FC at the time of the decision, etc. By contrast, the fulfillment policy makes real-time decisions in response to each incoming customer order, based on information such as delivery deadline, departure time of the truck, remaining inventory at each FC, etc., which may prioritize a faster delivery service. As the fulfillment process deals with a large number of SKUs (in hundreds of millions), operational decisions, and constraints, it is difficult to achieve integration of the fulfillment policy and network configuration in the foreseeable future. Omni-channel retailers also share the same type of decentralized decision-making structure as they need to ship BOSS products by trucks in their middle-mile network in high frequency (e.g., daily), but the truck assignments are updated at a lower frequency.

In theory, the middle-mile manager can take the fulfillment policy into account while designing the network configuration. However, due to a large amount of SKUs, fulfillment decisions, operational constraints and potential human errors, it is very time-consuming to simulate the actual fulfillment policy to determine the expected flow response. In our experience with a large online marketplace, simulation of the fulfillment policy on a single network configuration usually takes several days. Hence, it is impossible to optimize the network configuration over simulation. In practice,

the middle-mile manager may create ad-hoc rules to adjust the network configuration, which often leads to highly sub-optimal networks. The purpose of our research is to develop a new, systematic, and data-driven approach for designing the middle-mile network.

3.1.4 Our Approach and Contributions

We summarize our approach to the middle-mile network design problem and its implications.

A new network design problem with unknown flow response. Deviating from the classic network design problem that captures the brick-and-mortar operations, we present a network design problem with unknown flow response in Section 3.3 to reflect the unique operational features of the online marketplace and omni-channel retailing. To the best of our knowledge, this formulation is new to the literature.

Data-driven predictive models for unknown flow response. To solve this new problem, in Section 3.4, we introduce a data-driven framework to build a predictive model for the unknown flow response. The framework applies machine learning methods using historical shipment data. One advantage of using the data-driven approach is that we can directly predict the flow response aggregated at different levels instead of computing the flow response for millions of the products separately.

A novel network-decomposition prediction approach. The construction of the predictive model is a challenging task due to the sparsity of the data: the size of the input features is often much larger than the size of the historical shipment data. To cope with this issue, we decompose the network-level prediction problem into multiple prediction problems with much fewer input features. Our numerical study shows that this innovative approach outperforms that of the more natural and

direct approach, and its prediction accuracy is high even when the sample size is small.

A scalable algorithm with performance guarantee. In Section 3.5, we embed the predictive model into the original optimization problem, which results in a transformed network design problem with predicted flow response. However, finding a good network configuration is still difficult because the predictive model does not have a simple functional form and there is an exponential number of network configurations to predict. We first show that the problem can be reformulated as a mixed-integer linear program (MILP), which solves small problems, but is not viable for solving the large-scale problem faced by online marketplaces. To address this issue, we next identify this transformed problem as a c-supermodular minimization problem, which is equivalent to c-submodular maximization (See Baardman et al. (2020)). Then, we develop a scalable algorithm which is linear in network size. The scalability of the algorithm is especially important to the online marketplace with a large-scale network. We also prove that this algorithm has a theoretical approximation guarantee, and demonstrate that it is effective and efficient in a numerical study in Section 3.6.

Insights. Our research reveals several insights. First, we demonstrate the validity of the approach of using a data-driven predictive model to optimize the network configuration, in the setting where the exact fulfillment decision is unknown or computationally challenging. In a numerical study, we show that our approach is more appropriate than using the incumbent network configuration policy used by an online marketplace or the classic deterministic network design problem. Second, our approach generalizes to other business settings in which the problem has a decentralized decision-making structure and can be modeled as a multi-step stochastic problem, where the former step decisions are controlled by other decision entities. The

decision-maker of the latter step can focus on a single decision at a time by predicting the responses to his/her decision from the former steps. Third, when predicting the performance of a network, we find it is effective to use the network-decomposition approach to address the problem of the limited number of samples.

3.2 Related Literature

Our work is related to the literature in e-commerce fulfillment, combining predictive model with optimization, and submodular maximization.

E-commerce Fulfillment. Most existing works on e-fulfillment assume the network design is given and the packages are fulfilled only by 3PLs. Xu et al. (2009) consider the fact that physical pick-up of the product is usually delayed for hours after virtual assignment, so they have developed two swapping heuristics, resulting in fewer orders being split into multiple warehouses to ship. Acimovic and Graves (2014) consider a forward-looking fulfillment policy to minimize the shipment cost. They develop a linear program (LP) based heuristic, and use the dual value from the LP to guide warehouse selection in real time. Jasin and Sinha (2015) study a general fulfillment problem and propose two deterministic LP based heuristic fulfillment policies. They show the asymptotic competitive ratio for both heuristics compared to the expected cost under optimal control when the number of periods increases to infinity. DeValve et al. (2021) propose a class of spillover limit policies, which they use to evaluate the benefits of additional fulfillment flexibility to e-commerce under the local no-holdback constraint. Lei et al. (2018) consider a similar problem as Jasin and Sinha (2015) but study jointly optimizing pricing and fulfillment decisions. Acimovic and Graves (2017) study a periodic joint fulfillment and inventory replenishment policy by considering the possible spillover during the fulfillment process.

Different from all these works, we study network design for online marketplaces. Our methodology is also different as we use a data-driven approach.

Combining Predictive Model with Optimization. There is a growing literature on combining the optimization problem with a predictive model. Ferreira et al. (2015) optimize the pricing decision for an e-commerce retailer to maximize revenue. They use decision tree regression to predict the sales of an item given its own price and the relative price of competing styles. Cohen et al. (2017a) and Cohen et al. (2017) study promotion planning. They propose general classes of demand functions, which can be estimated from sales data. Baardman et al. (2018) study the schedule of promotion vehicles. Based on empirical data, they estimate the effects of assigning promotion vehicles. Baardman et al. (2020) also study a dynamic promotion targeting optimization problem and develop a novel customer trend demand model. Liu et al. (2021) study an order assignment problem faced by food service providers to minimize total delay. Their approach is to approximate the travel time of a deliveryman using LASSO. Chen et al. (2015) consider personalization in revenue management, which includes two applications of customized pricing and personalized assortment optimization. They estimate the parameters by regularized maximum likelihood estimation and then construct the decision policy based on this estimation. Chan et al. (2019) use neural network to develop a predictive model for the expected maximum flow in a network.

Our work combines the predictive model with optimization in a new application context with its own distinctive challenges. We propose a network-decomposition approach which significantly differs from the neural network approach of Chan et al. (2019), because the latter requires hundreds of thousands of sample data for model training even for a small network, which is infeasible in our application.

Submodular Maximization. The c-submodular maximization problem we

study is closely related to submodular maximization with non-monotone objective function under a cardinality constraint. Submodular maximization is known to be NP-hard. The scholars study different algorithms for different variations of submodular maximization problems. We use Random Greedy developed by Buchbinder et al. (2014), together with Deterministic Greedy from Nemhauser et al. (1978) and Stochastic Greedy from Mirzasoleiman et al. (2015) as benchmarks for comparison. Our focus is to design a scalable algorithm with a theoretical performance guarantee, which incorporates Double Greedy by Buchbinder et al. (2015) for unconstrained submodular maximization.

3.3 Network Design with Unknown Flow Response

In this section, we formulate the middle-mile network design problem. We use *origins*, indexed by i, to denote the FCs and *destinations*, indexed by j, to denote the DSs . The sets of origins and destinations are denoted by $\mathcal{I}$, $\mathcal{J}$, with cardinality I, J, respectively. *Arc* (i, j) denotes the travel route between origin i and destination j, and $\mathcal{G}$ denotes the ground set of all potential edges with *network size* $G = IJ$. The set of products sold by the online marketplace is denoted by $\mathcal{P}$ with cardinality P. A *network configuration* $\mathcal{Y} \subseteq \mathcal{G}$ is a set of activated edges, where activating an edge means assigning one truck traveling on the route every weekday. We also generalize the problem in Appendix B.2 to include the scenario where multiple trucks can be assigned to an activated arc.

The middle-mile manager decides the network configuration $\mathcal{Y}$ in *cycles*. A decision cycle is usually one or two weeks. After the network configuration $\mathcal{Y}$ is decided at the beginning of the cycle, it remains the same throughout the entire cycle. During each cycle, customer arrives sequentially, and the demand of each arriving customer

is described by vector $\boldsymbol{d} = \{d_{jp} : j \in \mathcal{J}, p \in \mathcal{P}\}$, where d_{jp} is the probability of a customer order being product p from destination j. We assume that the inventory at each origin is replenished only at the beginning of each cycle. Let the inventory levels at the beginning of the cycle be $\boldsymbol{s} = \{s_{ip} : i \in \mathcal{I}, p \in \mathcal{P}\}$, where s_{ip} is the quantity of product p stocked at origin i. Both the demand distribution and the inventory levels are assumed to be known parameters to the middle-mile manager. For example, the demand forecasting team can provide an estimation of $\boldsymbol{d}$. Similar to Jasin and Sinha (2015), we divide each day by an equal number of small periods, indexed by t; within each period, there is a constant probability that a customer order arrives. Each cycle, which consists of several days, has a total of T such periods.

Given a network configuration $\mathcal{Y}$, we use the term *expected network flow* to denote the expected number of each product shipped on each activated edge by the in-house truck shipment in one cycle. The network flow is determined by the fulfillment policy designed by another decision entity in the company. For an order arriving at destination j in period t, this policy decides from which origin i to fulfill the customer order using real-time information. Because the fulfillment policy is not controlled by the middle-mile manager and it is very difficult to compute the value of the network flow under the existing fulfillment policy, the middle-mile manager regards the expected network flow as an unknown response to the decision of network configuration $\mathcal{Y}$.

For each cycle, let κ_{ij} be the fixed charge of activating a truck route from origin i to destination j throughout the cycle. Given a network configuration $\mathcal{Y}$, the total line-haul cost in the cycle, i.e., the cost of creating a network $\mathcal{Y}$ of in-house truck routes, is then

$$C_1(\mathcal{Y}) = \sum_{(i,j)\in\mathcal{Y}} \kappa_{ij}. \tag{3.1}$$

The current truck routes may not be sufficient to ship all the ordered products, so online marketplaces would use 3PLs to cover the network shortfall. The *expected network shortfall* is denoted by $\boldsymbol{u}(\mathcal{Y}, \boldsymbol{s}, \boldsymbol{d}) = \{u_{ijp}(\mathcal{Y}, \boldsymbol{s}, \boldsymbol{d}) : i \in \mathcal{I}, j \in \mathcal{J}, p \in \mathcal{P}\}$, where u_{ijp} denotes the expected number of product p shipped from origin i to destination j by 3PLs in one cycle. Similar to the expected network flow, the expected network shortfall is also unknown.

For simplicity, to compute the shortfall cost given the expected network shortfall, we assume that each package contains exactly one product. Let c_{ij} be per unit charge by 3PLs for shipping one package from origin i to destination j. The *expected shortfall cost* for not being able to ship the products using truck capacity on activated edges is thus

$$C_2(\mathcal{Y}, \boldsymbol{s}, \boldsymbol{d}) = \sum_{(i,j)\in\mathcal{G}, p\in\mathcal{P}} c_{ij} u_{ijp}(\mathcal{Y}, \boldsymbol{s}, \boldsymbol{d}). \tag{3.2}$$

Our objective is to choose a network configuration $\mathcal{Y}$ to minimize the expected network cost $C(\mathcal{Y}, \boldsymbol{s}, \boldsymbol{d})$ in the current cycle, which is the sum of $C_1(\mathcal{Y})$ and $C_2(\mathcal{Y}, \boldsymbol{s}, \boldsymbol{d})$. Formally, the problem is:

$$\begin{aligned} \boldsymbol{P}: \min_{\mathcal{Y}\subseteq\mathcal{G}} C(\mathcal{Y}, \boldsymbol{s}, \boldsymbol{d}) &= C_1(\mathcal{Y}) + C_2(\mathcal{Y}, \boldsymbol{s}, \boldsymbol{d}) \\ &= \sum_{(i,j)\in\mathcal{Y}} \kappa_{ij} + \sum_{(i,j)\in\mathcal{G}, p\in\mathcal{P}} c_{ij} u_{ijp}(\mathcal{Y}, \boldsymbol{s}, \boldsymbol{d}) \\ s.t. \quad & |\mathcal{Y}| \leq K, \end{aligned} \tag{3.3}$$

where the cardinality constraint on the size of network configuration $\mathcal{Y}$ reflects the limited budget. However, because the expected network shortfall, and therefore $C_2(\mathcal{Y}, \boldsymbol{s}, \boldsymbol{d})$, is unknown, we cannot directly solve this problem. In the next section, we discuss how to predict $C_2(\mathcal{Y}, \boldsymbol{s}, \boldsymbol{d})$. With a slight abuse of the terminology,

we refer to $C_2(\mathcal{Y}, \boldsymbol{s}, \boldsymbol{d})$ as the unknown flow response in the rest of the chapter.

3.4 Predictive Models

In this section, we apply machine learning methods to predict the unknown flow response. We first introduce a general framework for constructing a data-driven predictive model. We then explain how we build a simulation model to generate a synthetic dataset, and use this dataset to illustrate and validate this framework.

3.4.1 Data-Driven Prediction Framework

Our prediction framework consists of three steps; the details are discussed below.

(i) Data requirement. In the first step of our framework, data is collected. We assume online marketplaces has collected a shipment dataset $S_N = \{(\mathcal{Y}^n, \boldsymbol{\alpha}^n, \boldsymbol{u}^n(\mathcal{Y}^n, \boldsymbol{\alpha}^n)) : n = 1, \ldots, N\}$ from the last N cycles, where $\mathcal{Y}^n$ is the implemented network configuration, $\boldsymbol{\alpha}^n$ are the network environment features that may affect the flow response, and $\boldsymbol{u}^n(\mathcal{Y}^n, \boldsymbol{\alpha}^n)$ is the recorded network shortfall (and hence $C_2^n(\mathcal{Y}^n, \boldsymbol{\alpha}^n)$).

Given the large number of SKUs for online marketplaces, we assume the covariates $\boldsymbol{\alpha}$ in our framework only contain the product aggregated inventory levels and demand distributions: $\{\breve{\boldsymbol{s}}^n = \{\breve{s}_i^n : i \in \mathcal{I}\}, \breve{\boldsymbol{d}}^n = \{\breve{d}_j^n : j \in \mathcal{J}\}\}$ where

$$\begin{aligned} \breve{s}_i &\equiv \sum_{p \in \mathcal{P}} s_{ip} = \text{the aggregated inventory level over all products at origin } i \qquad (3.4) \\ \breve{d}_j &\equiv \sum_{p \in \mathcal{P}} d_{jp} = \text{the aggregate demand distribution over all products} \\ &\quad \text{at destination } j. \qquad (3.5) \end{aligned}$$

$\boldsymbol{\alpha}^n$ can include additional features such as the cost parameters $\{(\kappa_{ij}, c_{ij}) : i \in$

$\mathcal{I}, j \in \mathcal{J}\}$, travel distance, truck routes schedule, holidays, weather forecasts, etc, as long as their values can be obtained.

(ii) Developing Predictive Models: In the second step, we consider different ways to construct $\hat{C}_2(\mathcal{Y}, \boldsymbol{\alpha})$, the estimator of the flow response for any given network configuration $\mathcal{Y}$ and covariates $\boldsymbol{\alpha}$. The first model is the most direct and natural approach:

- **Model 1: Network-level prediction.** We construct $\hat{C}_2(\mathcal{Y}, \boldsymbol{\alpha})$, an estimator of $C_2(\mathcal{Y}, \boldsymbol{\alpha})$, by using all available information from $(\mathcal{Y}, \boldsymbol{\alpha})$. We denote $\hat{x}(Y(\mathcal{Y}), \boldsymbol{\alpha}) \equiv \hat{C}_2(\mathcal{Y}, \boldsymbol{\alpha})$, where network configuration feature $Y(\mathcal{Y}) \equiv \{y_{ij}\}_{i=1,\ldots,I}$ and network environment features $\boldsymbol{\alpha} \equiv \{\{\breve{s}_i, i \in \mathcal{I}\}, \{\breve{d}_j, j \in \mathcal{J}\}\}$. Here, y_{ij} is a binary indicator variable that is set to 1 if edge (i, j) is activated. To train $\hat{x}(Y(\mathcal{Y}), \boldsymbol{\alpha})$ for any network, we use all possible sets of $(Y^n(\mathcal{Y}), \boldsymbol{\alpha}^n)$ from the data set.

A key challenge to Model 1 is that, in order to create an accurate estimator, the sample size N needs to be much larger than the input feature size $IJ + I + J$ to prevent overfitting. However, this requirement can hardly be met by the large online marketplace such as Amazon, where $IJ + I + J$ can be greater than 10 thousand, which is usually much larger than the N, the number of cycles in the shipment dataset. This problem is known as the curse of dimensionality in machine learning.

To overcome this shortcoming, we construct two prediction models, Models 2 and 3, in the same spirit as the principle of decomposition in data mining. The idea of decomposition is "to break down a complex Data Mining task into several smaller, less complex and more manageable, sub-tasks that are solvable by using existing tools, then joining their solutions together in order to solve the original problem." (Maimon and Rokach 2005). In Models 2 and 3, we decompose the flow response

into destination-level and arc-level flow responses, respectively, which require only partial information of $\mathcal{Y}$ to predict. As a result, the dimension of the input features is decreased and the effective sample size is increased.

- **Model 2: Destination-level prediction.** For Model 2, we decomposes the prediction of flow response to each destination in the network. For this purpose, for any $i \in \mathcal{I}$ and $j \in \mathcal{J}$, we define

$$m_i(\mathcal{Y}) \equiv \sum_j y_{ij} = \text{the out-degree at origin } i, \tag{3.6}$$

$$n_j(\mathcal{Y}) \equiv \sum_i y_{ij} = \text{the in-degree at destination } j. \tag{3.7}$$

  Then, for any destination $j \in \mathcal{J}$, we construct an estimator of $C_2(\mathcal{Y}, \boldsymbol{\alpha}|j)$, the total expected shortfall for all edges incident to j, as follows. The estimator is denoted by $\hat{x}(Y_j(\mathcal{Y}), \boldsymbol{\alpha}_j)$, which takes the destination-level network configuration feature $Y_j(\mathcal{Y}) \equiv \{\{y_{ij}\}_{i=1,\ldots,I}, n_j(\mathcal{Y}), \{m_i(\mathcal{Y}) \cdot y_{ij}\}_{i=1,\ldots,I}\}$ and destination-level network environment features $\boldsymbol{\alpha}_j \equiv \{\breve{d}_j\}$. Then, $\sum_{j\in\mathcal{J}} \hat{x}(Y_j(\mathcal{Y}), \boldsymbol{\alpha}_j)$ is used as the estimator of the total flow response. Note that due to decomposition, each destination in each cycle can be viewed as a sample for training $\hat{x}(Y_j(\mathcal{Y}), \boldsymbol{\alpha}_j)$, and our effective sample size is thus JN.

- **Model 3: Arc-level prediction.** Similar to Model 2, Model 3 decomposes the prediction of flow response to each edge in the network. For any possible edge $(i, j) \in \mathcal{G}$, we construct an estimator of $C_2(\mathcal{Y}, \boldsymbol{\alpha}|(i, j))$, the total expected shortfall at edge (i, j), as follows. The estimator is denoted by $\hat{x}(Y_{ij}(\mathcal{Y}), \boldsymbol{\alpha}_{ij})$, which takes the arc-level network configuration feature $Y_{ij}(\mathcal{Y}) \equiv \{m_i(\mathcal{Y}), n_j(\mathcal{Y}), y_{ij}\}$ and arc-level network environment features $\boldsymbol{\alpha}_{ij} \equiv \{\breve{s}_i, \breve{d}_j, c_{ij}\}$. Then, $\sum_{i\in\mathcal{I}, j\in\mathcal{J}} \hat{x}$ $(Y_{ij}(\mathcal{Y}), \boldsymbol{\alpha}_{ij})$ is used as the estimator of the total flow response. Note that due

Table 3.1: Input Feature Size and Effective Sample Size in Different Predictive Models.

	Model 1	Model 2	Model 3
Input Feature Size	$O(IJ)$	$O(I)$	$O(1)$
Effective Sample Size	N	JN	IJN

to decomposition, each possible edge $(i, j) \in \mathcal{G}$ in each cycle can be viewed as a sample for training $\hat{x}(Y_{ij}(\mathcal{Y}), \boldsymbol{\alpha}_{ij})$, and our effective sample size is thus IJN. Table 3.1 first summarizes the input feature size in each predictive model. It is observed that we achieve the dimensionality reduction in Model 2 and 3, where we project the network information to the lower-dimensional space by using the information of the degree of connectivity, which partially captures the connection status of the nodes. In addition, Table 3.1 describes the effective sample size in each model: Model 2 and 3 increase the effective sample size by a factor of J and IJ compared to Model 1.

Remark 1. *It is possible to further decompose the flow response prediction at the product level, i.e. to construct* $\hat{C}_2(\mathcal{Y}, \boldsymbol{\alpha}|(i, j, p))$*, an estimator of* $C(\mathcal{Y}, \boldsymbol{\alpha}|(i, j, p)) = c_{ij} u_{ijp}(\mathcal{Y}, \boldsymbol{\alpha})$ *for every single product on every arc. However, this is computationally impractical for online marketplaces with millions of products.*

(iii) Determining the Predictive Model. In the last step, we select the best predictive model from Model 1, 2 and 3.

First, we consider several common machine learning methods for each predictive model. The first two, ridge regression and LASSO, assume a linear relationship between the features and outcome. Both methods introduce a regularization term to shrink the coefficients of the parameters so that the resulting model can generalize better. The difference is that ridge regression uses the ℓ_1 norm regularization term, while LASSO uses the ℓ_2 norm regularization term. The remaining three, decision

tree regression, random forest regression, and support vector regression (SVR) with radial basis function (RBF), can handle non-linear relationships. The decision tree regression follows a tree structure to make the prediction: each non-leaf node represents a test on one of the features, each branch denotes the outcome of the test, and each leaf node represents the predictive value. The random forest regression constructs multiple decision trees at the same time and uses the average outcome as the predicted result. The idea of SVR is to make the prediction that has at most ε deviation from the actual observed response, and it uses RBF kernel to map data to higher dimensional space to help model non-linear relationship. It also has a regularization term to prevent over-fitting.

Each machine learning method has its own advantages: ridge regression, LASSO and SVR performs well if the sample size is smaller than the feature size due to the use of the regularization; decision tree regression is easy to interpret; random forest generalizes better to testing data than decision tree regression. There is no rule-of-thumb for deciding which machine learning method is more appropriate to construct the predictive model. For example, we can speculate that SVR with RBF kernel performs well for Model 1 since the dimensionality of its feature space is large. However, it may perform poorly if the inputs and the response actually have a linear relationship. It is also possible that random forest performs better than SVR with RBF kernel when the feature size is relatively small compared to the sample size. Thus, the selection of the machine learning method depends on the dataset.

Second, the values of the hyper-parameters also need to be optimized for each machine learning method. For a machine learning method, the model parameters are optimized via training, and the hyper-parameters are set prior to the training.

To compare predictive models with different machine learning methods and tuned hyper-parameters, we use nested cross-validation, which combines hyper-parameters

tuning and model selection at the same time. Then we identify the most accurate predictive model. The details of the machine learning methods and nested cross-validation are described in detail in Appendix B.3. Throughout, we use the mean absolute percentage error (MAPE) to evaluate the performance of an estimator. MAPE on a dataset with size N is defined as

$$\sum_{n=1}^{N} \frac{|C_2^n(\mathcal{Y}^n, \boldsymbol{\alpha}^n) - \hat{C}_2(\mathcal{Y}^n, \boldsymbol{\alpha}^n)|}{C_2^n(\mathcal{Y}^n, \boldsymbol{\alpha}^n)} \cdot 100\%, \tag{3.8}$$

where a lower value in MAPE implies a more accurate prediction.

Lastly, we derive the final model. We use another cross-validation with the identified predictive model and machine learning method with the lowest MAPE to tune the hyper-parameter on the entire dataset, and then train this model with the optimized hyper-parameter on the entire dataset.

3.4.2 Illustration of the Framework

In this subsection, we illustrate the above framework using a concrete example. For the data collection step, we use a synthetic dataset S_N generated via simulation to protect the confidentiality of online marketplaces we work with. The simulation model used in our work is only to illustrate and validate our approach.

3.4.2.1 Simulation Setup

We explain the details of the simulation setup here. For the origin-destination network, the locations of origins are the places which have been found to provide the lowest possible transit lead-time by a recent study in Chicago Consulting (2019). The locations of destinations are cities with the highest population. Because online marketplaces cannot stock all products in every origin, we set 50% of the products

as popular products, which are stocked in all origins and the rest of the products are stocked in each origin with uniform probability $p_{stock} = 50\%$. We focus on a network with $I = 3, J = 20, P = 30$, while also applying the same procedure to generate networks with different values of I, J, and P. In the simulation, each cycle has $T = 12500J$ periods. All inventories arrive before period 1. In each period, there is exactly one customer order arrival. We assume that there are 5 days over each cycle, and therefore, there are $2500J$ periods in each day. On each day, for each activated truck route, there is an assigned truck which departs from the origin at the end of the day. We set the truck capacity to be $\gamma = 1500$, because each truck can load around 30 pallets and each pallet has around 50 boxes in reality.

In the simulation, we implement two fulfillment policies. The first is a myopic fulfillment policy. For every customer order of product p in destination j, it first searches for the origins that have the inventory of product p and have a truck assigned between there and the destination j that has not reached its capacity. Then it selects one such origin i with the shortest distance l_{ij} and uses the corresponding truck to deliver the order. If no such origin exists, then the policy selects the origin i such that it has the inventory and the lowest 3PL unit price c_{ij}, and uses the 3PL to deliver the order. We note that a similar myopic fulfillment policy is implemented by a large American-based retailer (e.g., Acimovic and Graves 2014). Even though the myopic policy is pretty simple, we still need to rely on the simulation model to evaluate its performance as this policy needs to repetitively decide the fulfillment decision for every incoming customer order based on the current status of the network. The second is a probabilistic fulfillment policy developed by Jasin and Sinha (2015), where the solution to a deterministic linear program is used to decide the probability distribution of origins to fulfill the demand of each product p in each destination j.

While it is natural to create a dataset by randomly generate origin-destination

network configurations and then simulate them using fulfillment policies, the real-world dataset for networks configurations are not generated randomly. To verify the effectiveness of our approach more faithfully, we follow a data generation procedure that closely resembles the incumbent policy for changing network configurations from the online marketplace we worked with. This procedure generates network configurations by making incremental changes to the network: we first solve a deterministic network design problem with the random demand being replaced by its mean value. In cycle 0, we simulate the solution of the deterministic network design problem. In cycle n where $1 \leq n \leq N$, we perform the following process: for each edge (i, j) (in the network) from the previous cycle, we compute χ_{ij}^{n-1}, the truck shipment cost per package from the simulation of cycle $n-1$. To construct the network in the cycle n, we initially set it to be the network from cycle $n-1$ minus any edge (i, j) where χ_{ij}^{n-1} is higher than c_{ij}, i.e., removing edges where the truck shipment is more expensive than third-party shipment in terms of per-unit cost from the previous cycle. We rank the destinations by their percentages of third-party shipment from the previous cycle, defined as $\sum_{i\in\mathcal{I},p\in\mathcal{P}} u_{ijp}^{n-1}(\mathcal{Y}, \boldsymbol{\alpha}) / \sum_{p\in\mathcal{P}} d_{jp}T$, from the largest to smallest. For each of the highest ranked destinations, we activate a potential edge incident to each of those destinations with the lowest fixed cost, until either the size of the current network is K, or every destination has activated a new arc, or the remaining destinations' percentages of third-party shipment are too small such that adding another route would be significantly underutilized.

Throughout the chapter, we refer to the procedure described above as the *incumbent network configuration policy*, and use it as the data generation procedure. Next, we explain how components of the dataset S_N are recorded.

- $\boldsymbol{\alpha}^n$: We assume that the demand from destination j in cycle n of product p, d_{jp}^n, are identical across all products. Recall $\breve{d}_j^n = \sum_{p\in\mathcal{P}} d_{jp}^n$, then we have

$\sum_{j\in\mathcal{J}} \breve{d}_j^n = 1$ and $d_{jp}^n = \breve{d}_j^n/|P|$. For destination j, we let $\hat{d}_j$ be the proportion of the total population of cities in $\mathcal{J}$. To incorporate randomness, we set $\breve{d}_j^n = \hat{d}_j \cdot \omega_j^n, j = 1, \ldots, J-1$ where ω_j^n is independent and normally distributed with mean 1 and standard deviation 0.4, and $\breve{d}_J^n = 1 - \sum_{j=1,\ldots,J-1} \breve{d}_j^n$ (We discard any random instance with any negative d_j). For simplicity, we assume that the online marketplace has the same inventory levels in every cycle, i.e., $\boldsymbol{s}^1 = \ldots = \boldsymbol{s}^{N+1} = \boldsymbol{s}$, where $N+1$ is the current cycle we need to predict. For every product p, its inventory level in origin i is set to $s_{ip} = T \sum_{j:\theta_{ijp}=1} \hat{d}_j/|P|$ where θ_{ijp} is a binary variable to indicate if origin i is the closest origin with an inventory of product p to destination j. We assume the cost parameters $\{(\kappa_{ij}, c_{ij}) : i \in \mathcal{I}, j \in \mathcal{J}\}$ are static. The fixed charge of an in-house truck shipment is estimated by using the National average flatbed rate of \$2.14 per mile, and the assumption that each truck travels 5 times in one cycle. We also assume there is only one 3PL courier, UPS, and each package contains only 1 product with weight 2 pounds. We let c_{ij} be $8.759 + 0.846 + 0.001082 l_{ij}$ dollar per package, which is used to approximate the UPS rate by Jasin and Sinha (2015).

- $\boldsymbol{u}^n(\mathcal{Y}^n, \boldsymbol{\alpha}^n)$: The flow response is recorded after simulation completes for one cycle.

3.4.2.2 Constructing and Selecting the Predictive Model

Next, we numerically illustrate the data-driven prediction framework with the synthetic dataset. First, we use a 5-fold nested cross-valuation to determine the machine learning method and its hyper-parameters for each predictive model on a dataset with $N = 50$ cycles. Table 3.2 shows the average and standard error of MAPE in the

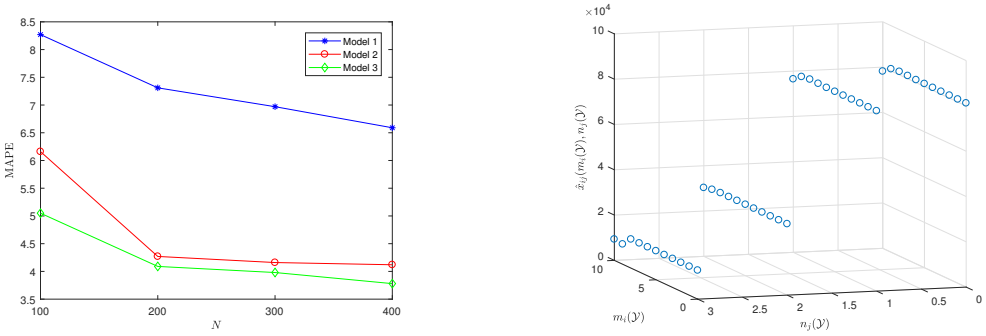
Table 3.2: Average and Standard Error of MAPE on Flow Response in Nested Cross-validation.

Myopic Fulfillment Policy						
Machine Learning Method	Average of MAPE			Standard Error of MAPE		
	Model 1	Model 2	Model 3	Model 1	Model 2	Model 3
Lasso	12.08	8.48	8.96	0.31	1.77	1.32
Ridge Regression	10.87	8.68	8.96	0.74	1.91	1.32
Decision Tree Regression	10.81	6.97	8.15	1.32	1.06	1.18
Random Forest Regression	8.27	**6.16**	**5.05**	1.93	0.93	1.25
SVR with RBF kernel	8.53	6.17	7.66	0.93	1.88	1.68
Probabilistic Fulfillment Policy						
Machine Learning Method	Average of MAPE			Standard Error of MAPE		
	Model 1	Model 2	Model 3	Model 1	Model 2	Model 3
Lasso	11.42	8.71	8.42	3.78	1.50	0.93
Ridge Regression	9.64	8.69	8.42	2.81	1.46	0.93
Decision Tree Regression	10.28	6.79	7.64	1.53	0.94	0.35
Random Forest Regression	7.56	**5.28**	**4.11**	1.11	0.40	0.51
SVR with RBF kernel	7.55	5.97	6.45	1.79	1.16	1.03

nested cross-validation on the flow response prediction from various machine learning methods with tuned hyper-parameters for Models 1 through 3. We choose Model 3 with random forest regression because it has the lowest mean of MAPE and a low standard error of MAPE.

Our study suggests that Model 1 is outperformed by the two proposed prediction models, Model 2 and 3. This verifies our intuition that the new prediction models improve the prediction accuracy. Notice that our observations of the three models are consistent regardless of the selected fulfillment policy, which implies the robustness of the new prediction model. We also tested the robustness of using network-decomposition when the networks in the dataset are randomly generated. We observe a similar result in Appendix B.5.

In addition, we tested the robustness of these observations by varying the sample size N, and the results are shown in Figure 3.1(a). We find that Model 2 and 3 are already more accurate when N is small, and increasing sample size N makes all three



(a) Impact of Sample Size on Predictive Model.

(b) An Example of the Estimator from Model 3.

Figure 3.1: Plots of Predictive Models

models more accurate, with Model 1 improving the fastest. Thus, Model 2 and 3 are more suitable when data is limited.

We can also have a pre-processing stage for Model 3: we first predict the edge-level flow response for all possible inputs of $(m_i(\mathcal{Y}),\ n_j(\mathcal{Y}),\ y_{ij})$ on each edge, and then store the results in computer memory. Such pre-processing is infeasible for Models 1 and 2 as their input domains are much larger, requiring a large computer memory space to store the predicted results, leaving fewer computing resources for optimization.

Finally, we select Model 3 with random forest regression as the predictive model, and train it again on the entire dataset to get the final predictive model of our framework. Figure 3.1(b) plots an example of an estimator trained from Model 3. The plot illustrates the estimated flow response for some activated edge (i, j) with different possible in-degrees $n_j(\mathcal{Y})$ and out-degrees $m_i(\mathcal{Y})$, while fixing all other input features. The figure shows that, in general, the edge-level predicted flow response is decreasing in $(m_i(\mathcal{Y}), n_j(\mathcal{Y}))$. This is because higher in-degrees of i or out-degrees of j should result in more product allocation to in-house truck shipment, leaving less volume for the 3PL carrier. Also, we note that the predicted shortfall cost lacks

general structural properties such as discrete convexity, which is consistent with our observations of the flow response generated via simulation.

3.5 Network Design with Predicted Flow Response

We next focus on the modified version of the network design problem $\boldsymbol{P}$ in (3.3), which replaces the unknown flow response by the predicted flow response via Model 3 with random forest regression.

To emphasize the dependence of the prediction on its input features, the predicted flow response associated with edge (i, j) given edge-level network configuration feature $Y_{ij}(\mathcal{Y})$ and edge-level network environment feature $\boldsymbol{\alpha}_{ij}$ in the current cycle is denoted by $\hat{x}_{ij}(m_i(\mathcal{Y}), n_j(\mathcal{Y}), y_{ij}) \equiv \hat{x}(Y_{ij}(\mathcal{Y}), \boldsymbol{\alpha}_{ij})$.

The transformed optimization problem is then to minimize the predicted network cost $\hat{C}(\mathcal{Y})$ as follows:

$$\hat{\boldsymbol{P}}: \min_{\mathcal{Y} \subseteq \mathcal{G}} \hat{C}(\mathcal{Y}) = \sum_{(i,j) \in \mathcal{Y}} \kappa_{ij} + \sum_{(i,j) \in \mathcal{G}} \hat{x}_{ij}(m_i(\mathcal{Y}), n_j(\mathcal{Y}), y_{ij}) \qquad (3.9)$$
$$s.t. \quad |\mathcal{Y}| \leq K.$$

It is in general computationally intractable to solve $\hat{\boldsymbol{P}}$ exactly. Although the unknown flow response can be predicted, the prediction function itself usually does not have a simple functional form. Instead, most of the common machine learning methods are either black-box models (e.g., neural network), or have a complex functional form (e.g., decision tree and random forest).

Remark 2. *It is worth noting that if Model 1 constructed from ridge regression or LASSO were chosen as the flow response predictor, then the above optimization problem would be easy to solve, because the objective function is linear with respect*

to each edge and we can choose the K edges with the lowest objective coefficients. However, we find the resulting network configurations usually incur a higher cost in a numerical study because the linear type prediction fails to capture the complex interrelationships among the activated edges.

We can reformulate Problem $\hat{\boldsymbol{P}}$ in (3.9) as a MILP. The details of this formulation is described in Appendix B.4. For companies with a small-size network, the problem can be solved in tolerable computational time. However, for major online marketplaces with more than 100 origins and 100 destinations, such as Amazon, the MILP would involve more than 100 million binary variables and constraints, which cannot be solved using off-the-shelf solvers. In this case, a more scalable approach is necessary.

3.5.1 c-Supermodularity

To develop a scalable optimization algorithm, we consider another approach, leveraging on the algorithms in submodular maximization. Assume the objective function in Problem $\hat{\boldsymbol{P}}$ is c-supermodular, a notion that generalizes and relaxes supermodularity, we can identify algorithms with theoretical approximation guarantees (see Section 3.6).

For any element ϕ and set X, let $X+\phi \equiv X \cup \{\phi\}$, and $X-\phi \equiv X \backslash \{\phi\}$. Denote by $f_\phi(X) = f(X+\phi) - f(X)$ the marginal changes of adding an element ϕ to X. Then

Definition 1. *A function $f : 2^{\mathcal{G}} \to \mathbb{R}$ is called c-supermodular (or c-submodular), if for every set X, Y with $X \subset Y \subset \mathcal{G}$, element $\phi \notin Y$, and $c \geq 0$, we have $f_\phi(X) \leq f_\phi(Y) + c$ (or $f_\phi(X) + c \geq f_\phi(Y)$). If $c = 0$, then f is supermodular (or submodular).*

Definition 2. *Similarly, a function f is called c-monotone on set $\mathcal{G}$ if for every set $S \subset \mathcal{G}$, element $\phi \in \mathcal{G} \backslash S$, and $c \geq 0$, we have $f_\phi(S) \geq -c$. If $c = 0$, f is monotone.*

We make the following assumption in the remainder of the chapter and show there exists an upper bound of c.

Assumption 1. *The objective function in Problem $\hat{\boldsymbol{P}}$ in (3.9) is c-supermodular.*

Assumption 1 implies that the marginal reduction in network cost by activating one more edge decreases when the number of activated edges increases, up to an additive error term of c.

Corollary 1. *There exists an upper bound on c such that Assumption 1 holds. The expression of the upper bound is provided in Appendix B.1.1.*

The theoretical performance guarantee (in the next section) can be loose if the value of c is large. Nevertheless, we find that the theoretical analysis for c-supermodular minimization problems enables us to design an algorithm that is effective in the numerical test. To provide some motivation for why Assumption 1 and c-supermodular optimization analysis is helpful, in the next Corollary 2, we introduce a stylized example where $c = 0$.

Corollary 2. *Assume the network has only 1 destination, unlimited truck capacity for all edges. If orders can be fulfilled after all demands are realized, then the objective function in Problem $\boldsymbol{P}$ in (3.3) is supermodular.*

Corollary 2 makes two restrictive assumptions, that there is only one destination and orders are fulfilled after the demand is known (i.e., offline optimization). For the offline assumption, we note that some effective online fulfillment policies proposed in the literature have performances close to the offline optimal solutions when the length

of cycle T is large, see, e.g., Jasin and Sinha (2015), Acimovic and Graves (2014). This intuitively suggests that the network cost under effective fulfillment policies exhibit similar (approximate) supermodularies. In general, a deeper understanding of the supermodularities of the network costs in more general settings is an important future research direction.

3.6 A Hybrid and Scalable Algorithm

In this section, we aim to develop an efficient algorithm to solve Problem $\hat{\boldsymbol{P}}$ in (3.9), a c-supermodular minimization problem. We first transform the problem into a non-negative c-submodular maximization problem by replacing the original objective function by the difference between a constant that upper-bounds the predicted network cost and the actual predicted network cost. Unfortunately, the new objective function is not monotone, so we cannot apply existing algorithms that require monotonicity. Instead, we find it more convenient to study the following general set maximization problem, which includes our problem as a special case:

$$\boldsymbol{P^c}: \max_{S\subseteq\mathcal{G}} f(S) \qquad s.t.\ |S| \leq K, \tag{3.10}$$

where $f(S)$ is non-negative, c-submodular, and non-monotone. That is, we aim to find a subset S of the ground set $\mathcal{G}$, with size no larger than K, that maximizes f. We use S^* to denote the optimal set. Correspondingly, throughout this section, we use the terms of elements and sets instead of edges and networks.

In Section 3.6.1, we adapt several existing algorithms from the submodular maximization literature as benchmarks. All these algorithms are "bottom-up" algorithms in the following sense: Starting from an empty set, we sequentially add an element in

each iteration to bring some improvement of the objective value in expectation, until K elements have been added so that the cardinality constraint is binding. In Section 3.6.2, we develop a scalable algorithm. This algorithm is a "top-down" algorithm in the following sense: First, we use an unconstrained algorithm to generate a solution (a set) that has a high objective value but may not be feasible, i.e., may contain more than K elements. Then, we search for a subset of this initial set until a feasible subset is reached.

Each of these algorithms will be analyzed for its performance guarantee and complexity. Similar to the submodular maximization literature, we define complexity as the number of the objective function (oracle) evaluations. In our context, a function evaluation means using Model 3 to compute the predicted network cost for a given network configuration. We say an algorithm is linear time if it has complexity $O(G)$.

3.6.1 Benchmarks: Bottom-Up Algorithms

We adapt three algorithms from the submodular maximization literature to our c-submodular maximization problem as benchmarks. The first is Deterministic Greedy, which is the classic algorithm and is most intuitive. It starts at the empty set and adds elements greedily until K elements are added. In each iteration, approximately G elements are evaluated for the marginal changes brought on the objective value. Thus, the complexity of this algorithm is $O(GK)$. For a monotone submodular function, the objective of Deterministic Greedy is known to be at least $(1-1/e)f(S^*)$ (Nemhauser et al. 1978), but there is no known theoretical guarantee in the literature for Deterministic Greedy with a non-monotone objective function.

The second benchmark is Random Greedy, which follows a similar bottom-up approach as Deterministic Greedy. The difference is that in each iteration, it ran-

Algorithm 2: Deterministic Greedy.

Initialize: $S_0 \leftarrow \emptyset$. **for** $k = 1$ *to* K **do**
  $\phi_k \leftarrow \arg\max_{\phi \in \mathcal{G} \setminus S_{k-1}} f_\phi(S_{k-1})$. $S_k \leftarrow S_{k-1} + \phi_k$.
end
return S_K.

Algorithm 3: Random Greedy.

Initialize: $S_0 \leftarrow \emptyset$. **for** $k = 1$ *to* K **do**
  Let $M_k = \arg\max\{\sum_{\phi \in S} f_\phi(S_{k-1}) | S \subseteq \mathcal{G} \setminus S_{k-1}, |S| = K\}$. Let ϕ_k be uniformly chosen from M_k. $S_k \leftarrow S_{k-1} + \phi_k$.
end
return S_K.

domly adds one element from the K best elements. Its complexity is still $O(GK)$. It is guaranteed to achieve an objective value of at least $f(S^*)/e$ for non-monotone submodular maximization. This theoretical guarantee can be extended to our c-submodular maximization problem, as in Theorem 1.

Theorem 1. *(adapted from Theorem 1.3 from Buchbinder et al. (2014)) Let f be a non-negative non-monotone c-submodular function. Random Greedy has the following performance guarantee:*

$$\mathsf{E}[f(S_K)] \quad \geq \quad \frac{1}{e} f(S^*) - (G + K)c$$

with $O(GK)$ function evaluations.

While the above two algorithms are intuitive, they are slow from the perspective of our application as they both need to evaluate the impact of approximately G elements in each iteration, and G can be large. For this reason, we also consider a faster benchmark: Stochastic Greedy from Mirzasoleiman et al. (2015). It works similarly as Deterministic Greedy. The difference is that instead of choosing the best element among all that are remaining, it first randomly samples s elements from the

Algorithm 4: Stochastic Greedy.

Initialize: $S_0 \leftarrow \emptyset$. **for** $k = 1$ *to* K **do**
  $M_k \leftarrow$ a subset obtained by randomly selecting s elements from $\mathcal{G} \backslash S_{k-1}$.
  $\phi_k \leftarrow \arg\max_{\phi \in M_k} f_\phi(S_{k-1})$. $S_k \leftarrow S_{k-1} + \phi_k$.
end
return S_K.

remaining, then chooses the best element from this sample in each iteration. If we set $s = (G/K)\log(1/\epsilon)$, then this algorithm has the complexity of $O(G\log(1/\epsilon))$ and achieves an objective value at least $(1 - 1/e - \epsilon)f(S^*)$ for monotone submodular maximization. However, like Deterministic Greedy, there is no known theoretical performance guarantee if the objective function is non-monotone.

3.6.2 A Scalable, Top-Down Algorithm

Because our focus is on large-scale e-commerce transportation networks, we are interested in developing a fast algorithm which is scalable in network size. The re-design of the network configuration is not a trivial task. Once the network configuration is decided, it needs to be evaluated by different interested parties, and there will be feedback on how the proposed network configuration needs to be adjusted. Then the middle-mile manager needs to rerun the algorithm by incorporating the feedback. Fast algorithm can thus help speed up this cycle of refining the network configuration.

We present a top-down algorithm, which builds on Double Greedy introduced by Buchbinder et al. (2015). Double Greedy earns its name because it works with two sets in each iteration. In the initial step, it sets $X_0 = \emptyset$ and $Y_0 = \mathcal{G} = \{\phi_1, \phi_2, ..., \phi_G\}$. In iteration k, it determines whether to add ϕ_k to X_{k-1} while maintaining Y_{k-1} as is or to remove ϕ_k from Y_{k-1} while maintaining X_{k-1} as is. As a result, $X_G = Y_G$ after the last iteration G as both sets agree on every element of $\mathcal{G}$. The objective of the algorithm then becomes maximizing $f(X_G) + f(Y_G) = 2f(X_G)$. This is done

by smartly making the adding or removing decision in each iteration to maximize $f(X_k) - f(X_{k-1}) + f(Y_k) - f(Y_{k-1}) = [f(X_k) + f(Y_k)] - [f(X_{k-1}) + f(Y_{k-1})]$. This algorithm is scalable with complexity $O(G)$: there are G iterations, and each iteration k requires two function evaluations.

Algorithm 5: Hybrid.

Step 1: Double Greedy
Initialize: $X_0 \leftarrow \emptyset, Y_0 \leftarrow \mathcal{G}$, $\phi_k \leftarrow k$-th element of $\mathcal{G}$. **for** $k = 1$ *to* G **do**
  $a_k \leftarrow f(X_{k-1} + \phi_k) - f(X_{k-1})$. $b_k \leftarrow f(Y_{k-1} - \phi_k) - f(Y_{k-1})$. **if** $a_k < 0$ *and* $b_k < 0$ **then**
    $p_k \leftarrow 1/2$.
  else if $a_k = b_k = 0$ **then**
    $p_k \leftarrow 1$.
  else
    $p_k = \max\{a_k, 0\}/(\max\{a_k, 0\} + \max\{b_k, 0\})$.
  end
  with probability p_k **do**: $X_k \leftarrow X_{k-1} + \phi_k, Y_k \leftarrow Y_{k-1}$. **else** (with the complement probability $1 - p_k$) **do**: $X_k \leftarrow X_{k-1}, Y_k \leftarrow Y_{k-1} - \phi_k$.
end
Step 2: Set Reduction
if $|X_G| \leq K$ **then**
  Let $X_r = X_G$ and skip step 3.
else
  Initialize: $S_0 \leftarrow X_G$, $\phi_k \leftarrow k$-th element of S_0. **for** $k = 1$ *to* $|S_0|$ **do**
    if $f(S_{k-1}) - f(S_{k-1} - \phi_k) < 0$ **then**
      $S_k \leftarrow S_{k-1} - \phi_k$.
    end
  end
  Let $X' = S_{|S_0|}$.
end
Step 3: Stochastic Greedy
if $|X'| \leq K$ **then**
  Let $X_r = X'$.
else
  We apply Stochastic Greedy on the ground set X' with the input $S'_0 \leftarrow \emptyset$ and output S'_K. Let $X_r = S'_K$.
end
return X_r.

Our scalable algorithm, Hybrid, has three steps, including Double Greedy, a set

reduction process, and Stochastic Greedy. Step 1 applies Double Greedy to find a good starting set X_G with a known performance guarantee. Here, different from Buchbinder et al. (2015), we need to consider a new scenario where neither action will lead to improvement because our objective function is not strictly submodular. In this scenario, we randomly choose an action from adding and removing (See details in Lemma 2). After step 1, X_G may not satisfy the cardinality constraint. In that case, the algorithm proceeds to step 2, where we introduce a novel process to remove some "bad" elements from X_G, meaning that removing elements that can improve the objective function. This step can guarantee function f to be sufficiently close to a monotone function on the remaining set (see details in Lemma 3). It transforms a difficult optimization problem with non-monotone objective function into an easier problem with the almost monotone objective function. The second step has complexity $O(G)$ because it considers the removal of at most G elements. In step 3, we treat X' as the ground set and apply Stochastic Greedy to find a feasible set with a good performance guarantee (See details in Lemma 5).

By setting $s = (G/K)\log(1/\epsilon)$, step 3 has $O(G\log(1/\epsilon))$ function evaluations because it has K iterations and each iteration involves $(G/K)\log(1/\epsilon)$ function evaluations. As the complexities of step 2 and 3 are linear in G, Hybrid is also scalable, with complexity $O(G\log(1/\epsilon))$.

The following theorem summarizes the theoretical results for this algorithm.

Theorem 2. *Let f be a non-negative non-monotone c-submodular function. Set $s = (G/K)\log(1/\epsilon)$. Hybrid has the following performance guarantee:*

$$\mathsf{E}[f(X_r)] \geq \frac{K}{2(G+K)}\left(1-\frac{1}{e}-\epsilon\right)f(S^*) - \frac{3GK}{2(G+K)}\left(1-\frac{1}{e}-\epsilon\right)c - (2-\epsilon)Kc,$$

with $O(G\log(1/\epsilon))$ function evaluations for any fixed ϵ.

Table 3.3: Approximation Guarantee and Complexity of Algorithms Assuming $K = \lambda IJ$.

Algorithm	**Approximation Guarantee**	**Complexity**
Deterministic Greedy	–	$O(GK) = O(I^2J^2\lambda)$
Random Greedy	$\frac{1}{e}f(S^*) - (G+K)c$	$O(GK) = O(I^2J^2\lambda)$
Stochastic Greedy	–	$O\big(G\log\frac{1}{\epsilon}\big) = O\big(IJ\log\frac{1}{\epsilon}\big)$
Hybrid	$\frac{K}{2(G+K)}\left(1-\frac{1}{e}-\epsilon\right)f(S^*)$ $-\frac{3GK}{2(G+K)}\left(1-\frac{1}{e}-\epsilon\right)c-(2-\epsilon)Kc$	$O\big(G\log\frac{1}{\epsilon}\big) = O\big(IJ\log\frac{1}{\epsilon}\big)$

Remark 3. *Random Greedy, Stochastic Greedy and Hybrid are all randomized algorithms. As a result, the output from such algorithms is also random. In practice, one can first run the randomized algorithm for few times to get several candidate network configurations, then evaluate their predicted network costs using the predictive model (or to simulate the network costs using the simulation model if possible). The network configuration with the lowest cost is used as the final result.*

3.6.3 Algorithm Comparison

We now compare the theoretical guarantees and complexities of algorithms considered in Section 3.6.1 and 3.6.2. Table 3.3 provides a summary. To have a better understanding of the complexity in the context of network design for online marketplaces, we assume $K = \lambda IJ$ where λ is the proportion of all potential edges that can be connected.

In terms of the complexity, Hybrid and Stochastic Greedy are on the same order, while the complexity of Deterministic Greedy and Random Greedy are on a much higher order. In terms of the theoretical performance guarantee, we assume c is small and the negative constant term in performance guarantee is negligible. Ran-

dom Greedy has a better performance guarantee than Hybrid. There is no known performance guarantee for both Deterministic Greedy and Stochastic Greedy when the objective function is non-monotone. Notice here we use worst-case analysis to derive the performance guarantee. An algorithm with a worse performance guarantee does not necessarily imply it works badly in a numerical study.

Remark 4. *Notice that the fastest algorithm above still takes $O(G)$ function evaluations, which may still be intolerable for a large network. In practice, however, the demand of a delivery station is usually served by fulfillment centers in the same region. Therefore, we can partition the network into subnetworks based on regions $\{(\mathcal{I}_k, \mathcal{J}_k)\}$, where $\sum_k \mathcal{I}_k = \mathcal{I}$, $\sum_k \mathcal{J}_k = \mathcal{J}$ and $|\mathcal{I}_k| = I_k$ and $|\mathcal{J}_k| = J_k$ for all k. The complexity of solving all sub-problems is $O(\sum_k I_k J_k)$, which is much smaller than $O(G)$.*

3.7 Numerical Study

In this section, we compare the effectiveness of the different algorithms presented in the previous section in both small and large networks and demonstrate that Hybrid is scalable.

The numerical study is tested in Python 3.7 on a 4.00 GHz Intel i7 CPU. We set $I = 3$ for all networks in this section but vary J in different tests. Model 3 with random forest regression is used as our predictive model. We also numerically tested Model 1 and 2 and find they lead to network configurations with higher costs. For each J, the synthetic dataset is generated using the procedure described in Section 3.4.2.1 under the myopic fulfillment policy. While not presented in the chapter, we also tested the probabilistic fulfillment policy and the results are similar.

We use the incumbent network configuration policy for both generating a dataset

and benchmarking against our algorithm. More specifically, we use the first 80 cycles ($N = 80$) to construct the predictive model. For the last 20 cycles, we use all algorithms introduced in Section 3.6 together with our predictive model, to generate network configurations for each cycle. We then compare the cost of the network configurations generated with our algorithm with the network cost under the incumbent network configuration policy for the last 20 cycles.

Small Systems: We consider a small network with $I = 3, J = 4, K = 6$ here. The purpose of selecting a small network is that we can find the optimal network configuration via enumeration. We introduce the two performance measures to compare the different algorithms in small systems. Let $\mathcal{Y}_n^a$ denote the derived network configuration by a specific algorithm a in cycle n, its predicted network cost $\hat{C}(\mathcal{Y}_n^a)$ is computed using the predictive model. For the same configuration, we compute the *actual network cost*, denoted by $\check{C}(\mathcal{Y}_n^a)$, using the average flow response generated via ten simulation runs, and each run adopts a different realization of the demand sequence. The first is how good the algorithm is in solving a supermodular minimization problem, measured by the average relative gap Gap_p between the predicted network cost $\hat{C}(\mathcal{Y}_n^a)$ and the minimum predicted network cost $\hat{C}_*^n$ obtained by solving Problem $\boldsymbol{P}_{\text{MILP}}$ in (B.62). The second is how good the derived network configuration is if it is implemented in reality, measured by the average relative gap Gap_a between the actual network cost $\check{C}(\mathcal{Y}_n^a)$ and the minimum actual network cost $\check{C}_*^n$ obtained by enumerating all possible network configurations and simulating their flow responses. In other words, we use the following two measures

$$Gap_p\% = \sum_{n=81}^{100} \frac{\hat{C}(\mathcal{Y}_n^a) - \hat{C}_*^n}{\hat{C}_*^n}, \quad Gap_a\% = \sum_{n=81}^{100} \frac{\check{C}(\mathcal{Y}_n^a) - \check{C}_*^n}{\check{C}_*^n}, \tag{3.11}$$

and a lower value implies better performance.

Table 3.4: Performance of Selected Algorithms on Small and Large Networks

(a) Small Network with $I = 3, J = 4$.

Performance Measure		$Gap_p\%$	$Gap_a\%$
Deterministic Greedy		2.60	3.30
Random Greedy, $s = 2$		2.89	2.91
Stochastic Greedy	$s = 3$	10.95	11.29
	$s = 4$	5.81	6.19
	$s = 5$	3.29	3.58
Hybrid	$s = 3$	0.23	0.58
	$s = 4$	1.58	0.51
	$s = 5$	1.21	1.13
Incumbent Policy		13.73	12.74
FCTP		6.93	3.94

(b) Large Network with $I = 10, J = 10$.

Performance Measure		$Gap_p^{SQ}\%$	$Gap_a^{SQ}\%$
Deterministic Greedy		2.60	-3.33
Random Greedy, $s = 10$		0.29	5.86
Stochastic Greedy	$s = 20$	-2.93	-1.01
	$s = 30$	-6.79	-1.83
	$s = 40$	-8.25	-7.83
Hybrid	$s = 20$	-5.12	-1.42
	$s = 30$	-6.71	-2.55
	$s = 40$	-8.97	-4.62
FCTP		7.31	0.61

For Stochastic Greedy and Hybrid, we consider sample size $s \in \{2, 4, 6, 8, 10\}$. For randomized algorithms including Random Greedy, Stochastic Greedy, and Hybrid, we run each algorithm 100 times and use its average performance for comparison.

Table 3.4(a) shows the result. Before going through the details, we verify that the predictions are accurate: the average MAPE of flow response among all developed network configurations is around 8.90%. Due to the accurate prediction, we first observe that a lower Gap_p usually leads to a lower Gap_a. This is intuitive as a low predicted network cost implies a low actual network cost, given an accurate estimator. As a result, we can compare Gap_p and Gap_a of each algorithm at the

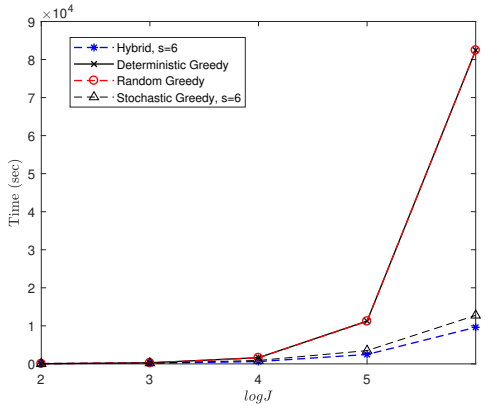


Figure 3.3: Computational Times of Selected Algorithms with $I = 3$.

same time. Deterministic Greedy generally has the best performance, and Random Greedy has the worst performance in the majority of the instances. Between Hybrid and Stochastic Greedy with identical s, Hybrid is slightly better. We also observe that increasing s improves the performance of both Stochastic Greedy and Hybrid as these two algorithms are likely to include better elements from random samples with a larger size. Deterministic Greedy, Stochastic Greedy ($s \geq 8$) and Hybrid ($s \geq 6$) are better than the incumbent network configuration policy.

Large System: We consider a larger sized problem with $I = 10, J = 10, K = 25$. Due to the scale of the system, it is difficult to obtain the value of the minimum predicted network cost $\hat{C}^n_*$ and the minimum actual network cost $\check{C}^n_*$. Thus, for the larger system, we use the following two different performance measures:

$$Gap^{SQ}_p\% = \sum_{n=81}^{100} \frac{\hat{C}(\mathcal{Y}^a_n) - \hat{C}(\mathcal{Y}^{SQ}_n)}{\hat{C}(\mathcal{Y}^{SQ}_n)}, \quad Gap^{SQ}_a\% = \sum_{n=81}^{100} \frac{\check{C}(\mathcal{Y}^a_n) - \check{C}(\mathcal{Y}^{SQ}_n)}{\check{C}(\mathcal{Y}^{SQ}_n)} \quad (3.12)$$

where $\hat{C}(\mathcal{Y}^{SQ}_n)$ and $C^a(\mathcal{Y}^{SQ}_n)$ are the predicted cost and actual cost of the network derived by the incumbent network configuration policy.

Table 3.4(b) summarizes the results, which is generally consistent with our observations in Table 3.4(a).

Computation Time: Another important performance metric of the algorithm is computation complexity. For Stochastic Greedy and Hybrid, we set s to be 6. We fix $I = 3$, consider $J \in \{4, 8, 16, 32, 64\}$, and let $K = 0.8IJ$.

In Figure 3.3, we first observe that the time of the linear-time algorithms are not strictly linear in G. The reason is that Model 3 requires the prediction to be made at every arc. As a result, the time of making a single prediction increases for increasing network size. Among all algorithms considered, Hybrid is the fastest for large J, while Stochastic Greedy is slower than Hybrid. Deterministic Greedy and Random Greedy are the slowest. This shows that scalable algorithms can significantly reduce the computation time.

The Performance of Deterministic Approximation: A natural question is whether the classic deterministic network models can be leveraged to provide a good heuristic network configuration, instead of using the predictive model. To test this idea, we optimize the network where the flow response is approximated by the optimal fulfillment policy when the demand is replaced by its mean value. This lead to a fixed-charge capacitated multicommodity network design (CMND) problem with a cardinality constraint. However, this problem is still not tractable in reality for two reasons: first, the scale of the problem is extremely large due to the scale of products offered by the online retailer. For example, a typical Walmart supercenter offers 150,000 SKUs, and Amazon even sold more than 500 million SKUs in 2018 (ScrapeHero 2018); second, the CMND problem requires the demand prediction of every product, but the prediction can be inaccurate for most long-tail products.

We can address these two issues by ignoring the product label, then the problem becomes a fixed-charge transportation problem (FCTP) with a cardinality constraint. This new problem is an NP-hard problem, but it has a much smaller size than the CMND problem, and it is much easier to predict the aggregated demand over all

Table 3.5: Performance of Selected Algorithms with Different SKU-FC Inventory Assignments.

SKU-FC Inventory Assignment	High		Low	
Performance Measure	$Gap_p\%$	$Gap_a\%$	$Gap_p\%$	$Gap_a\%$
Deterministic Greedy	2.26	24.11	0.10	2.74
Random Greedy	34.76	61.05	29.68	29.76
Stochastic Greedy, $s = 8$	7.32	33.39	1.95	4.93
Hybrid, $s = 8$	5.86	29.10	1.26	3.85
Incumbent Policy	9.14	36.69	17.14	19.65
FCTP	13.42	2.46	12.35	10.69

the products. The derived optimal network configuration can be used as a heuristic solution, denoted by FCTP.

To test the performance of this heuristic, we consider the same setting as in Table 3.4(a). Recall the setting considers a high SKU-FC inventory assignment: half of the products are stocked in every FC, and the rest of the products are stocked in 50% of the FCs. In this setting, the SKU-FC inventory assignment is high, as each FC holds roughly 75% of the SKUs. We also introduce a new scenario of low SKU-FC inventory assignment: each product is stocked in only one of the FCs. We continue to use Gap_p and Gap_a as the performance measures, and the results are summarized in Table 3.5: FCTP is the best when the SKU-FC inventory assignment is high; Hybrid and Deterministic Greedy are the best when the SKU-FC inventory assignment is low. In reality, the SKU-FC inventory assignment is typically low, as most of the long-tail products are stocked in a few FCs Anderson (2006). This partially explains why the online retailer we worked with relies on the incumbent policy to modify their network configuration and also demonstrates the potential value of our approach of using Hybrid or Deterministic Greedy policies.

3.8 Conclusion and Discussion

To stay competitive and sustain growth, online marketplaces constantly seek solutions to optimize their transportation networks. However, computing the exact flow response from the existing fulfillment policy can be very difficult. One of our main contributions is to identify a class of real-world network design problems where the network flow and shortfall, determined by a different decision entity within the organization, are unknown. We develop a data-driven approach to predict the flow response using network-decomposition, as firms often only have a very small number of samples on the performances of networks. The network design problem with unknown flow response can then be approximated as a network design problem with predicted flow response. One common challenge in data-driven optimization is the computation complexity. To address this issue, we identify our problem as a c-supermodular minimization problem and develop a scalable algorithm which is linear in network size. This is especially important to an online marketplace with a large-scale network.

We next discuss some potential directions to extend our study. In our predictive model, we considered a simple set of features to predict the shortfall cost. Given the data collected by some online marketplaces, the size of $\boldsymbol{\alpha}$ can be larger than what we considered in this chapter. For a dataset with a large number of features, an efficient and effective machine learning method needs to be developed. Some promising methods include representation learning (e.g., Bengio et al. 2013) and deep learning (e.g., LeCun et al. 2015).

Chapter 4

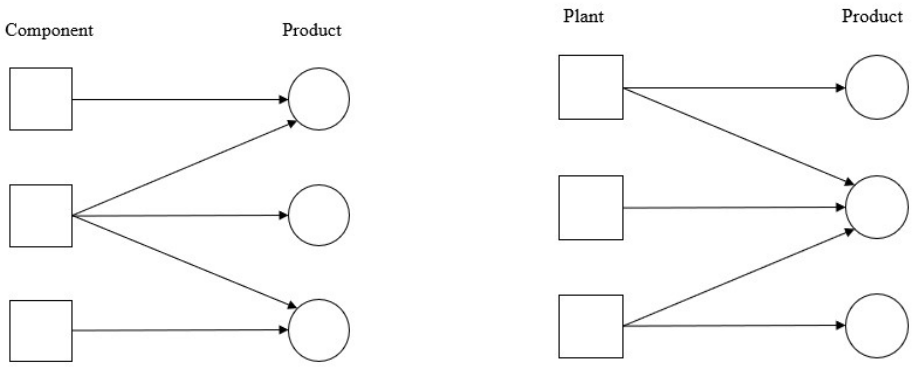
A Graph Neural Network Approach for Supply Chain Systems with Graphical Structures

4.1 Introduction

In modern times, large operational networks are ubiquitous in supply chains. For instance, most retailers manage a distribution network connecting factories, warehouses, and stores, such as Walmart, Amazon, Target, or any supermarket chains. Manufacturers also manage complex materials and production networks within and cross their factories. Examples include:

- **Assemble-to-order (ATO) systems**. Many computer manufacturers, such as Lenovo and Dell Computers, assemble computers from component inventories according to realized customer orders. Each product requires a different set of components, called the bill of material; and several products may share one or a few common component(s). Thus, the assembly process comprises a network structure defined by the bill of material of the final products, as shown in Figure 4.1(a)). This manufacturing strategy is known as ATO. It eliminates unwanted final products and exploits the risk-pooling benefit through component commonality, thus enabling mass customization.

- **Process flexibility networks**. Many manufacturers, including large automobile manufacturers such as Ford and General Motors, produce multiple products



(a) ATO System (b) Process Flexibility Network

Figure 4.1: Illustration of Materials and Production Networks

from multiple plants (or production lines). A plant that can produce multiple products is called a flexible plant, whereas a plant that only produces one product is called a dedicated plant. A more flexible plant is more expensive but has the ability to help the company respond to demand fluctuations. Given a set of products with random demands and a set of plants with fixed capacities, process flexibility is defined as the product-plant links indicating each plant's capability to produce the products. We call such a network a process flexibility network (see Figure 4.1(b)).

As companies are constantly looking for ideas to improve the design and planning of their operations, they need to evaluate the operational performances of many alternative networks, such as the supplier base, number of warehouses and stores, shipping schedule between warehouses and stores, the number of components and products, and the degree of commonality. Given any operational policies such as a transportation plan, a product mix, an inventory policy, a network is evaluated by its operational performance, like the long-run average inventory cost in the ATO system, the average profit in the process flexibility system, and the average transportation cost in the distribution network.

The performance evaluation of any given network and operational policy first requires representing the network by a graph, where one typically uses nodes to represent entities such as customers, facilities, and suppliers, and uses edges to represent the exchange relationship between those entities, as in Figure 4.1. To avoid ambiguity, in this chapter, we use the terminology *network* to denote the real-world supply chain network and *graph* to denote the abstract data structure. Also, we define the operational performance of the supply chain network as the graph-level operational performance. Our objective is to develop an efficient and effective ML model to predict the graph-level operational performance.

4.1.1 ML on the Graph

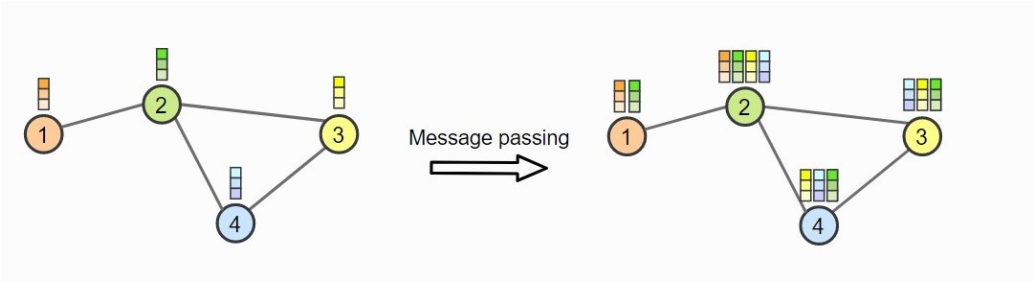
Traditionally, classic operations research tools (e.g., stochastic processes and simulation) have been used for exact or approximate evaluations of graph-level operational performances. These models can often become computationally expensive or intractable because they involve the evaluation of high-dimensional joint probabilities. In addition, for analytical tractability, these models usually have to leverage the special structure of the network and rely on restrictive assumptions and simplifications. For example, many studies of ATO systems assume a stationary Poisson process of customer demand and a simple allocation policy (e.g., the first-come-first-served rule), while these assumptions may deviate significantly from the practice.

The recent developments in machine learning (ML) have provided some promising tools for overcoming some limitations mentioned above. The ML approach is fundamentally different from the classic operation research tools, as it predicts the operational permanence based on the observed data from the complex system dynamics instead of a simplified mathematical model of the system dynamics. As such, it is assumption-free. Thanks to the availability of massive volumes of data and the

advancement in CPU and GPU processing, the ML model can generate significantly more accurate and faster predictions compared to twenty years ago, and has already achieved success in several operational management applications. For example, Liu et al. (2021) apply an ML model to predict the expected delay of the drivers in the order assignment because the drivers' behaviors are difficult to simulate. Ban and Rudin (2018) use an ML model to directly predict the optimal decision in the newsvendor problem, which does not assume any distributional knowledge of the demand.

When the learning target is the graph-level operational performance, the standard approach in the literature is to define graph-level features as inputs and then apply an ML model, such as multi-layer perceptron (MLP), to construct a graph-level predictive model. The simplest way of defining the graph-level features is to flatten the adjacency matrix as a vector, concatenate it with other graph-level attributes. Another way is to use the adjacency matrix itself and then apply the convolutional neural network method (Chan et al. 2022).

However, this approach is not permutation invariant, which means the prediction result is dependent on the arbitrary ordering of the nodes. Thus, the resulting predictive model is not robust as an arbitrary change in the ordering of the nodes can lead to a completely different prediction. It also requires careful and hand-engineered statistics and measures to design the graph-level features. Designing the graph-level features is ad-hoc and time-consuming, and the designed features are inflexible (Hamilton 2020).



Note: figure copied from Lippe P (2022) *UvA Deep Learning Tutorials*. https://uvadlc-notebooks.readthedocs.io/.

Figure 4.2: Illustration of Message Passing Scheme in GNN

4.1.2 Graph Neural Network

Fortunately, there is a new ML model, Graph Neural Network (GNN), which has the potential to address the above limitations. The idea of the GNN model is to generate node *embeddings*, which are vectors to represent the information of a node and its neighborhood nodes in a low dimensional space. GNNs follow a *message passing* scheme to update the node embedding. This scheme is shown in Figure 4.2, where each node has a vector to represent the current node embedding. The initial value of the node embedding is the node-level features, such as the demand information of each product and the supply information of each component in ATOs. Through message passing, each node updates its embedding by aggregating (such as summing) the embeddings from its topological neighborhood nodes. The final node embedding is then transformed into the prediction of the GNN model. GNNs directly use the graphical structure to compute the node embeddings and capture the dependency between nodes in the graphical structure. GNNs are also permutation invariant as the aggregation of the neighborhood node information is not affected by the ordering of the nodes. Also, the initial node-level features are often easier to define compared to the graph-level features.

GNN has achieved great success in applications such as social networks, citation

networks, biochemical networks, etc. For example, Uber Eats uses GNN on a graph with users, restaurants and foods and predicts the rank of restaurants and foods for every user. They observe a performance improvement of over 20% in different performance metrics in an experiment of implementing GNN (Jain et al. 2019).

Surprisingly, GNN has not been applied to supply chain networks. This may be because the GNN model is a new tool, and customization is necessary. Another reason may be that early applications of GNN focus on a learning setting which is *transductive*, meaning that the predictive model has access to a single and fixed graph, and it only needs to predict certain unknown nodes. While the learning setting for operational performances is mostly *inductive*, meaning that the predictive model has access to several graphs with known outputs (i.e., operational performances), and needs to learn a rule which generalizes to graphs with unknown outputs.

In this paper, we seek to understand the effectiveness of the GNN model as a prediction tool for supply chain systems with graphical structures, Below we summarize our main approaches and contributions.

4.1.3 Main Contributions

Introducing GNN to supply chain systems. To the best of our knowledge, our work is the first in the operations management literature to discuss the application of GNN in supply chains. We show how to formalize the supply chain dataset in the form of graph-structured data, using ATO systems and process flexibility networks as an illustration. We also provide detailed implementation of the GNN algorithms which tailors to these two systems and demonstrate their effectiveness.

Customization of the GNN Model. For applications of the GNN model to supply chain systems, we discover that two customized approaches are more effective.

One is the *decompose-then-aggregate prediction approach*. Another is the *edge-node graph transformation method*. We specify them below.

- *Decompose-then-aggregate prediction approach.* When using GNN to predict graph-level performances, the standard approach in the GNN literature is to use graph-level inputs. Deviate from this common practice, our research takes two decompose-then-aggregate prediction approaches – the node-level and edge-level approaches, even though the ultimate output of the GNN model is the prediction of the graph-level performance. This is motivated by our observation that the performance of a large operational network can often be measured by aggregating the node-level or edge-level operational performances.

  Specifically, the node-level approach takes node-level features as inputs to make predictions of node-level performances, and then aggregates these predictions together to derive the prediction of the graph-level performance. Here, the way of aggregating node-level predictions needs to be consistent with the way of aggregating node-level performances for the graph-level performance. For instance, in an ATO system, this approach first predicts node-level performances, which are the long-run average inventory holding costs at every component and back-ordering costs at every product. Because the graph-level performance is the long-run average total inventory cost, which is the sum of the node-level performances, we obtain the prediction of the graph-level performance by summing the node-level predictions.

  We adopt the GraphSAGE models proposed by Hamilton et al. (2017) as the base GNN model, as GraphSAGE is particularly simple and effective for graphs that have rich node attribute information.

  Similarly, the edge-level approach takes edge-level features as inputs to make

prediction of edge-level performances, and then aggregate these predictions together to derive the prediction of the graph-level performance. For instance, in a process flexibility network, this approach first predicts the edge-level performances, which are the average profits at every plant-product edge. Because the graph-level performance is the average total profits, which is the sum of the edge-level performances, we obtain the prediction of the graph-level performance by summing the edge-level predictions.

- *Edge-node graph transformation method.* For edge-level prediction, we introduce a graph transformation approach so that the GraphSAGE model on the transformed graph is designed for directly making edge-level predictions. This is different from the common approach of transforming the node-level outputs from GNNs into edge-level predictions. The GraphSAGE model only utilizes the information of the neighborhood nodes, while the GraphSAGE model on the transformed graph can utilize the information of the neighborhood edges.

Numerical Experiments and Insights. We design a numerical experiment to compare our tailored GNN model with the edge-node graph transformation against several benchmarks, including the base GNN model and other traditional ML approaches, such as the bag of nodes and edges, convolutional neural network and random forest. In particular, we use ATO systems to illustrate the scenario where the evaluation of the operational performance can be node-level, and use process flexibility systems to illustrate the scenario where the evaluation of the operational performance can be edge-level.

We find the base GNN model to be accurate for node-level prediction, while our GNN model with the proposed edge-node graph transformation to be beneficial for edge-level prediction. Through comparison of different ML models, we also find that

ignoring the graphical structure results in a substantial loss in prediction accuracy.

We also compare the performance of the graph-level prediction. Although some ML applications, including GNNs, can directly predict the graph-level performance, our study reveals that for several supply chain applications, it is more accurate to adopt the decompose-then-aggregate prediction approach.

4.2 Literature Review

Our work is related to the literature on the GNN and applying the ML model to the supply chain system.

GNN. The concept of the GNN was first proposed in Gori et al. (2005) and Scarselli et al. (2008). The GNN model learns the embedding of individual nodes via aggregating information from its topological neighborhood nodes. In this process, both the structural information about the graph and the features of the neighborhood nodes are explicitly captured. The advantage of the GNNs is that it can leverage node features compared to other embedding methods such as matrix factorization (Shen et al. 2018, Yang et al. 2018) and random walks (Perozzi et al. 2014). There are many popular GNN variants, such as GCN (Kipf and Welling 2016), GAN (Veličković et al. 2018) and GraphSAGE (Hamilton et al. 2017). We refer the reader to Wu et al. (2020) and Zhou et al. (2020) for a detailed review of GNNs.

While there is a rich literature on the GNN model, the edge-level predictions are less studied. The standard approach to derive the embedding of an edge is to concatenate the embeddings of two adjacent nodes. However, it ignores the essential edge features from the neighborhood edges. Gong and Cheng (2019) introduce an attention mechanism to explore edge features, but it is designed for node-level prediction, i.e., the edge information is used to improve the prediction accuracy of the

nodes embeddings. We proposed an edge-node graph transformation approach where each edge uses the information from the neighboring edges to improve the prediction of its edge embedding.

ML application in supply chain systems. The application of ML in the evaluation of supply chain systems has seen rapid development. Ang et al. (2016) propose a Q-Lasso method to predict the wait time in the emergency department. Shang et al. (2017) use Bayesian statistics to predict the transport risk, defined as the deviation of the actual arrival time from the planned arrival time, in cargo logistics. Cui et al. (2018) use different ML models with social media information to predict daily sales for an online retailer. Guo et al. (2021) use the regression tree to predict passengers' connection times at an airport. Salari et al. (2022) use the regression tree to predict order delivery time in online retailing. Chan et al. (2022) study the prediction of the expected maximum network flow in the process flexibility network. They follow the traditional approach where graph features are first defined and then a standard ML model is applied. They use the adjacency matrix as the graph-level features for a convolutional neural network since the adjacency matrix can be considered as grid-like inputs, but they do not consider other attributes of the graph. Chen et al. (2020) study the prediction of the expected third-party shipment cost in the e-commerce fulfillment network. They define edge features include the node degree at the origin and destination of the edge, then use the random forest model to predict the shipment cost at each edge. Recently, there have been works which directly learn the optimal decision in the supply chain systems. Ban and Rudin (2018) study the feature-based newsvendor problem and demonstrate the benefits of using the empirical risk minimization approach to solve the newsvendor problem with feature data. Bertsimas and Kallus (2020) study the learning of the optimal decision of a conditional stochastic optimization problem with imperfect observations.

Bravo and Shaposhnik (2020) propose a framework for mining the optimal policies in several operational problems. Wei et al. (2021) use reinforcement learning to identify the optimal solution to the flexibility design problems. Alternatively, Elmachtoub and Grigas (2017) and Elmachtoub et al. (2020) consider a "predict, then optimize" framework to construct loss functions which minimize the decision error instead of the prediction error.

However, these works do not explicitly utilize the graphical structure in learning. Our work fills this gap by focusing on the GNN method which considers the node attributes and uses the structure of the graph to improve learning. We show that GNN has a better performance compared to the traditional ML approaches in predicting the operational performances of the supply chain systems.

4.3 Model Setting and Preliminaries

In this section, we first define the notation of graphs and provide several examples. Then we introduce the idea of the GNN model.

4.3.1 Graph Data Structure

For the supply chain networks considered in this paper, we assume that their graphical representations are bipartite. Specifically, each graph representation contains nodes belong to two categories: supply nodes which offer inventory, suppliers, service, etc., and demand nodes represent customer orders. The graph is denoted by $G(V_s, V_d, E)$ with a set of supply nodes V_s, demand nodes V_d, and edges E. We also define set of nodes $V = V_s \cup V_d$. The size of the supply nodes and demand nodes are m and n, respectively.

For a given supply chain network with graphical representation G, we denote its

operational performance by $o(G)$, which we also refer to as the graph-level performance. For example, the operational performance can be the average cost over time in an ATO system. In many supply chain systems, the graph-level performance $o(G)$ can be easily decomposed into operational performances of individual nodes $o(v)$ or edges $o(i, j)$. For example, the operational performance of an ATO system is the sum of the average inventory holding cost on components and the average backorder cost on products.

The nodes in the graph have their own characteristics, which are summarized in node features x_v. We also assume nodes have homogeneous node features. Even if the nodes may have different attributes categories, we can let node features contain all attributes categories and we set the attribute to 0 if the corresponding category does not apply to the node.

Next, we describe the graph data structure for the two supply chain systems we study.

ATO system. V_s is the set of components and V_d is the set of products. E is the bill of the materials, which specifies the configuration of the components for each product. The product j follows a Poisson process with rate λ_j. The unfulfilled order of product j has a backorder cost rate b_j and the holding inventory of component i has a holding cost rate h_i. The component i has a lead time L_i. We consider a system following a first-come-first-served allocation rule and a constant base-stock policy for inventory replenishment. Vector $\mathbf{s} = \{s_1, \ldots, s_m\}$ is the vector of the base-stock levels. During the period of the observation, $\bar{I}_i$ is the average number of on-hand inventory of component i, and $\bar{B}^j$ is the average number of backorder of product j.

The graph-level operational performance, $o(G)$, is the long-run average total in-

ventory cost

$$\sum_{i \in V_s} h_i \bar{I}_i + \sum_{j \in V_d} b_j \bar{B}^j. \tag{4.1}$$

Note that the long-run average inventory holding cost $h_i \bar{I}_i$ on component i and backorder cost $b_j \bar{B}^j$ on product j are the observed node-level operational performance $o(v)$. In this case, $o(G) = \sum_{v \in \{V_s \cup V_d\}} o(v)$.

The node features x_i at supply node i contain attributes such as $L_i, s_i, h_i, \sum_{(i,j) \in S} \lambda_j$. The node features x_j at demand node j contain attributes such as $b_j, \lambda_j, \sum_{(i,j) \in S} s_i$.

Process flexibility network. V_s is the set of supply plants and V_d is the set of product demands. E is the network design. While the plant has fixed capacity C, demand of product have Q scenarios. The profit margin fulfilling product demand j from plant i is p_{ij}. The production starts after the demand is realized.

The graph-level performance, $o(G)$, is the average total profit among all Q scenarios. To compute their values, we first solve the following problem

$$\begin{aligned} \max_{f_{ij}^q, (i,j) \in S, q = 1, \ldots, Q} \quad & \frac{1}{Q} \sum_{(i,j) \in S, q = 1, \ldots, Q} f_{ij}^q p_{ij} \\ s.t. \quad & \sum_{i \in V_s} f_{ij}^q \leq d_j^q, j \in V_d, q = 1, \ldots, Q \\ & \sum_{j \in V_d} f_{ij}^q \leq s_i, i \in V_s, q = 1, \ldots, Q \\ & f_{ij}^q \leq y_{ij} C, f_{ij}^q \geq 0, (i,j) \in E, q = 1, \ldots, Q. \end{aligned} \tag{4.2}$$

Note that we can split the profits among plants and products

$$\begin{aligned} o(i) &= \alpha \sum_{j \in V_d} f_{ij}, i \in V_s \\ o(j) &= (1-\alpha) \sum_{i \in V_s} f_{ij}, j \in V_d \end{aligned}$$

where α is the split factor. In this case, $o(G) = \sum_{v \in \{V_s \cup V_d\}} o(v)$.

However, the value of the node-level performance is dependent on our approach to split the profits between plants and products. It is more intuitive to focus on the edge-level performance $o(i,j)$, which is the average profit on each arc, $\frac{1}{Q} \sum_{q=1,\ldots,Q} f_{ij}^q p_{ij}$. This edge-level performance is clearly defined and we have $o(G) = \sum_{(i,j) \in E} o(i,j)$. The node features x_i at supply plant i contain attributes such as $c_i, \sum_{(i,j) \in S,q} d_j^q$. The node features x_j at demand node j contain attributes such as $\sum_q d_j^q, \sum_{(i,j) \in S} c_i$.

4.3.2 GNN model

GNN utilizes a message passing scheme to exploit the graphical structure. We first introduce the message passing scheme by assuming that the model has already been trained and that the parameters are fixed.

Algorithm 6 introduces how the node embeddings are generated in the message passing scheme. The vector message of node v, h_v^k, is initialized as x_v at the beginning of iteration 0, and then the messages are exchanged between nodes and updated. During iteration k, each node v aggregates the embeddings from its neighborhood $N(v)$ to generate $h_{N(v)}^k$. Then both $h_{N(v)}^k$ and the previous embedding h_v^{k-1} are used to update the new embedding h_v^k. After K iterations, h_v^K is the final embedding for the node.

Here, the number of iterations K is the number of times that the messages are

Algorithm 6: Message Passing Scheme

Inputs: Graph $G(V, E)$; input features $\{x_v, v \in V\}$; the number of iterations K, aggregate function AGG, activation function σ, neighborhood function $N : v \to 2^V$, weight matrix W

Output: $h_v^K, v \in V$

$h_v^0 \leftarrow x_v, v \in V$

for $k = 1$ *to* K **do**

  for $v \in V$ **do**

    $h_{N(v)}^k \leftarrow \text{AGG}(h_u^{k-1}, u \in N(v))$

    $h_v^k \leftarrow \sigma(W \cdot \text{CONCAT}(h_v^{k-1}, h_{N(v)}^k))$

  end

end

passed. The aggregate function AGG decides how the node embeddings are aggregated. The activation function σ is useful for learning non-linear relationships. Some common activation functions are relu, sigmoid, tanh; and some common aggregate functions are mean, pool, gcn and lstm. The neighborhood function $N(\cdot)$ returns the nodes connected to a given node. We assume the choices of these parameters are fixed. The weight matrix W is assumed to be trained and thus are also known values. We provide more details about how to decide these values in the next subsection.

Note that the activation function does not require the ordering of the nodes in the neighborhood, so GNNs are permutation invariant.

The quantity h_v^K is the embedding of node v. Recall the operational performances can be edge-level in the process flexibility network. This is inconsistent with the node-level embedding. A common approach to address this issue is to define the edge embedding by concatenating the node embeddings and edge features $x_{(i,j)}$ if they exists

$$h_{(i,j)}^K \equiv (h_i^K, h_j^K, x_{(i,j)}) \tag{4.3}$$

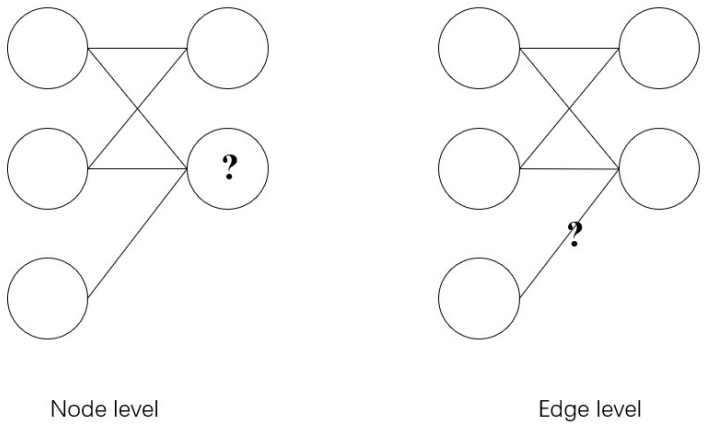


Figure 4.3: Illustration of the Node-level and Edge-level Prediction

Note that the embedding is still a vector and we need to further transform it to the final prediction. The transformation is dependent on the prediction task. Figure 4.3 shows node-level and edge-level predictions. We next explain the design of another layer for transforming the embedding to final predictions at different levels.

Node-level predictions. We use a multi-layer perceptron (MLP) to construct the final predictions. We consider such an MLP in Figure 4.4. Inside the MLP, the first layer is the input layer, which corresponds to $h(v) \equiv h_v^K$. Each node, $h_d(v), d = 1, \ldots, D$, represents one feature of $h(v)$. The second layer is a hidden layer with a prescribed size L. The third layer is the output layer with a single node. Between the input and the hidden layer, there is a weight matrix U. Between the hidden and output layer, there is a weight matrix U'. There are 2 activation functions $\phi_1(\cdot)$ and $\phi_2(\cdot)$. We assume the activation functions are known and the weight matrices U and U' are already trained. For $v \in V$, we define the transformation of the final

embedding as

$$u_l(v) = \phi_1\Big(\sum_{d=1}^{D} U_{dl} h_d(v)\Big), l = 1, \ldots, L, \tag{4.4}$$

$$p(v) = \phi_2\Big(\sum_{l=1}^{L} U'_l u_l(v)\Big). \tag{4.5}$$

The final output from the GNN model is $p(v)$ on each node $v \in V$.

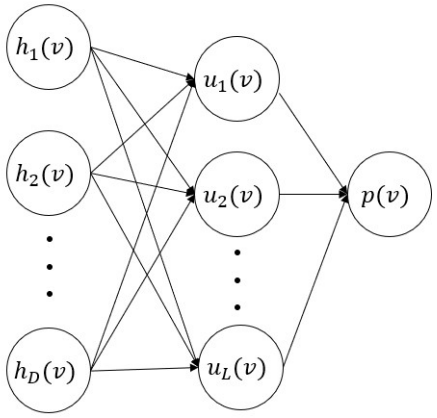


Figure 4.4: Illustration of MLP for Node-level Prediction.

Edge-level predictions. We construct a similar MLP for the final prediction, which is shown in Figure 4.5. The first layer in the MLP corresponds to $h(i,j) \equiv h^K_{(i,j)}$. Each node, $h_d(i,j), d = 1, \ldots, D$, represents one feature of $h(i,j)$. For $(i,j) \in E$, we define the transformation of the final embedding as

$$u_l(i,j) = \phi_1\Big(\sum_{d=1}^{D} U_{dl} h_d(i,j)\Big), l = 1, \ldots, L, \tag{4.6}$$

$$p(i,j) = \phi_2\Big(\sum_{l=1}^{L} U'_l u_l(i,j)\Big). \tag{4.7}$$

The final output from the GNN model is $p(i,j)$ on each edge $(i,j) \in E$.

Graph-level predictions. We first apply a sum pooling layer to get a graph-

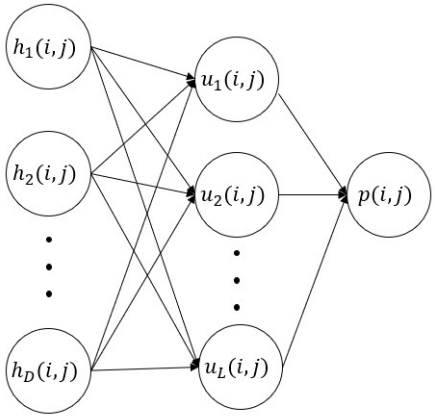


Figure 4.5: Illustration of MLP for Edge-level Prediction.

level embedding,

$$h_G^K \equiv \sum_{v \in V} h_v^K. \tag{4.8}$$

The use of the sum pooling layer is because the graph-level performance in supply chain systems can usually be calculated as the sum of the performances from individual edge and node. It is not necessary to consider more complex approach to compute the graph embedding.

We then construct a similar MLP for the final prediction, which is shown in Figure 4.6. The first layer in the MLP corresponds to $h(G) \equiv h_G^K$. Each node, $h_d(G), d = 1, \ldots, D$, represents one feature of $h(G)$. We define the transformation of the final embedding as

$$u_l(G) = \phi_1\left(\sum_{d=1}^{D} U_{dl} h_d(G)\right), l = 1, \ldots, L, \tag{4.9}$$

$$p(G) = \phi_2\left(\sum_{l=1}^{L} U_l' u_l(G)\right). \tag{4.10}$$

The final output from the GNN model is $p(G)$ on graph G.

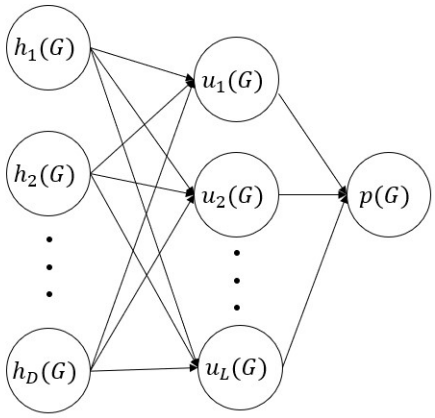


Figure 4.6: Illustration of MLP for Graph-level Prediction.

Next, we illustrate the prediction process in the GNN model using different examples. For simplification, we consider the update of the node embedding being simply the mean value of the previous node embedding and the mean embeddings from the neighborhood. Furthermore, we consider only 1 iteration in the message passing scheme. For the node-level prediction, we directly use the results of the message passing, the node embedding, as the prediction output. For the edge-level prediction, we directly use the mean value of the edge embedding as the prediction output.

We first introduce two examples commonly used by the ML community. We note that the training setting for the first two examples is transductive, meaning that the learning algorithm is performed within a single and fixed graph.

Transductive, citation network, node-level. We consider a simplified citation network (see Figure 4.7). Each node represents an academic paper and each edge represents the paper-paper citation relationship. We assume the green nodes are papers published in an ML journal and orange nodes are papers published in all

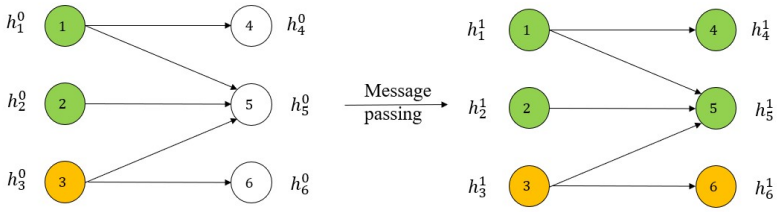


Figure 4.7: Example of GNN: Node-level Prediction in Citation Network

other journals. Everything is known except the categories of papers 4, 5, 6, which are the predicted targets. The initial embedding can be a binary value to indicate whether "machine learning" appears in the keywords. Here, $\mathbf{h}^0 = [1, 1, 0, 1, 1, 0]$.

In message passing, we update the final embeddings of node 4, 5, 6 and have

$$h_4^1 = 0.5 \cdot h_4^0 + 0.5 \cdot h_1^0 = 1, \tag{4.11}$$

$$h_5^1 = 0.5 \cdot h_5^0 + 0.5 \cdot \frac{h_1^0 + h_2^0 + h_3^0}{3} = 0.83, \tag{4.12}$$

$$h_6^1 = 0.5 \cdot h_6^0 + 0.5 \cdot h_3^0 = 0, \tag{4.13}$$

which is $\mathbf{h}^1 = [1, 1, 0, 1, 0.83, 0]$.

We set a classification rule: for each node, if the value of the final node embedding is above 0.5, then the paper is an ML paper; otherwise, it is not an ML paper. Then nodes 4 and 5 are classified as papers published in ML journals, while node 6 is a paper published in some other journal.

Transductive, citation network, edge-level. The next example is based on the same context as the previous example. Here, the goal is to predict whether one paper has any citation relationship with another paper. Everything is known except the citation between paper 2 and paper 5, which is the predicted target. Here, $\mathbf{h}^0 = [1, 1, 0, 1, 1, 0]$, and similarly we have $\mathbf{h}^1 = [1, 1, 0, 1, 0.75, 0]$.

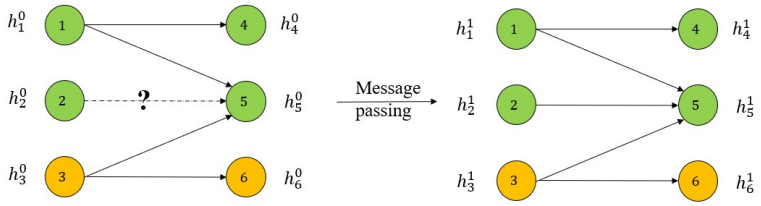


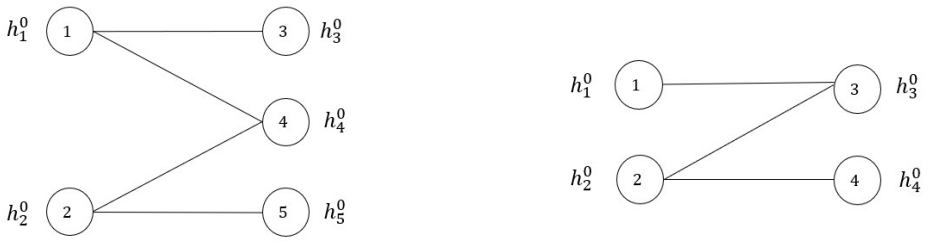
Figure 4.8: Example of GNN: Edge-level Prediction in Citation Network

We set a classification rule: for each edge, if the mean value of the final node embeddings of the adjacent nodes is above 0.5, then we predict there is a citation relationship; otherwise, there is no citation relationship. Thus, we predict paper 2 has a citation relationship with paper 5.

Next, we identify two examples of applying GNN in supply chain systems. We note that in our examples, the training setting is inductive, meaning that the learning algorithm is performed to deal with multiple graphs.

Inductive, ATO system, node-level. For example, Figure 4.9(a) use new graph (also for the next graph) can be seen as an ATO system where the task is to predict the average inventory cost at each node. Different from the classification tasks considered in the GNN literature, the prediction of the node-level operational performances is often a regression problem.

Inductive, process flexibility system, edge-level. Figure 4.9(b) can be seen as a process flexibility network where the task is to predict the average profit at each edge. Different from the link prediction tasks considered in the GNN literature, the prediction of the edge-level operational performances is often a regression problem.



(a) ATO System

(b) Process Flexibility Network

Figure 4.9: Examples of GNN: ATO and Process Flexibility System

4.3.3 Learning the parameters of GNN

In the previous subsection, we assume all parameters in GNN are known. Here, we discuss how to learn these parameters.

For machine learning models, there are two types of parameters: model parameters and hyperparameters. Model parameters are part of the model. They are the parameters which can be learned by solving an optimization problem to minimize the loss function, which measures the gap between the observed data and predictions. They include the weight matrix W in message passing and weight matrix U and U' in MLP. Hyperparameters are external to the model, They include the number of iterations K, choice of the aggregate function AGG and activation function σ in message passing, the size of the hidden layer L in MLP. They are manually set before the model starts training.

To optimize the model parameters, we first construct a loss function as the objective function of the learning problem. We then apply a standard optimization method, such as gradient descent, to minimize the loss function.

For example, consider learning model parameters for the predictive model of node-

level predictions, the loss function on a single graph is defined as

$$\sum_{v \in V} (o(v) - p(v))^2, \tag{4.14}$$

where $o(v)$ is the observed operational performance on node v. We then apply gradient descent to solve the following problem.

$$\min_{W,U,U'} \sum_{v \in V} (o(v) - p(v))^2. \tag{4.15}$$

We first initialize the model parameters by default values, then repeat the following process: (i) We follow the steps described in the previous subsection to derive the final predictions $p(v)$ on each node, and this step is denoted as forward propagation. Then we are able to evaluate the loss function and its value implies how to adjust the model parameters. (ii) We compute the current gradient of the loss function and use it to update the model parameters. This step is known as back propagation. The forward and backward propagation are repeated until the termination condition is met, and we then get the optimized model parameters.

Similarity, we can define the loss function for other levels of predictions. For edge-level predictions, recall the observed operational performance on edge (i, j) is $o(i, j)$. Then the loss function is defined as

$$\sum_{(i,j) \in E} (o(i, j) - p(i, j))^2. \tag{4.16}$$

For graph-level prediction, recall the observed operational performance on graph G is $o(G)$, then the loss function is defined as

$$(o(G) - p(G))^2. \tag{4.17}$$

For hyperparameters, they are usually tuned via cross-validation. We manually compare the model performance under different choices of the hyperparameters. We have introduced a detailed nested cross-validation framework in Appendix C.3.

4.4 Customization of GNN Model for Supply Chain Networks

For the standard GNN model, we find that the loss function used for the graph-level prediction only utilizes the observation of graph-level performance. However, we can observe node-level or edge-level operational performances in many supply chain systems. Therefore, it is interesting to incorporate these observations to improve the prediction accuracy of the GNN model. Another finding is that the common approach to obtain the final edge embeddings in the GNN model is to aggregate the final embeddings of adjacent nodes and the edge features. This approach only utilizes the features of the predicted edge. In reality, the neighborhood edges also contain rich information. In fact, the edge features are usually more informative than the node features because it is straightforward to define the edge features by combining the features of the adjacent nodes. The question to ask is how to utilize the neighborhood edges information to update the edge embeddings. We propose an edge-node graph transformation approach to address this issue.

4.4.1 Decompose-then-aggregate Prediction Approach

Recall that for graph-level prediction, the standard approach is to create a prediction $p(G)$ for the graph-level performance $o(G)$ and then minimize the squared loss between these two values. We can also observe $o(v)$ on each node for supply chain systems such as ATOs, and $o(i, j)$ on each edge for systems such as process flexibility

networks.

We propose to use GNN for the node-level prediction and adopt the node-level loss function if we have node-level observations,

$$\sum_{v \in V} (o(v) - p(v))^2, \tag{4.18}$$

and use GNN for the edge-level prediction and adopt the edge-level loss function if we have edge-level observations,

$$\sum_{(i,j) \in E} (o(i,j) - p(i,j))^2. \tag{4.19}$$

4.4.2 Edge-Node Graph Transformation

Motivated by message-passing algorithms proposed for weighted matching and min-cost flow problems (see e.g., Bayati et al. 2008, Gamarnik et al. 2012), we propose an edge-node graph transformation which converts the edges in the original graph as nodes in the transformed graph. When applying GNN on the transformed graph, learning the information from the neighborhood node in the transformed graph is equivalent to learning the information from the neighborhood edges in the original graph. Therefore, this transformation allows us to exploit the information on the neighborhood edges and get the embeddings of the edges in the original graph.

We next introduce the transformation approach defined in Algorithm 7. For each edge (i, j), we convert it into two nodes $j \to i$ and $i \to j$ in the transformed graph. The edges in the transformed graph is defined by the following rule

$$j \to i \text{ connects to } \; i' \to j' \text{ if } i' = i;$$

Note that edges in the transformed graph is directed. This is to reflect the difference between neighborhood edges. Some edges are neighbors because they share a common supply node, while others share a common demand node.

The idea of transformation is illustrated using a 2 by 3 network in Figure 4.10. For example, edge $(1,3)$ in the original graph has two neighboring edges $(1,4)$ and $(2,3)$. Its corresponding node in the transformed graph are $3 \to 1$ and $1 \to 3$. $3 \to 1$ has neighborhood node $2 \to 3$ and $1 \to 4$; $1 \to 3$ has neighborhood node $3 \to 2$ and $4 \to 1$.

We denote GNN on this transformed graph as GNN-TD. We also consider adjusting Algorithm 7 by considering edges as undirected. Note that if we restrict the predictions at (i,j) and (j,i) as identical, then it is equivalent to the line graph transformation shown in Appendix C.1. We denote GNN on the undirected transformed graph as GNN-TU.

Algorithm 7: Transformation Approach

Inputs: Graph $G(V,E)$
Output: Graph $G(V',E')$
$V' = \{j \to i, i \to j, (i,j) \in E\}, E' = \emptyset$
for $(i,j) \in E$ **do**
  for $(i',j') \in E$ **do**
    if $(i,j) \neq (i',j')$ & $i = i'$ **then**
      Add $(j \to i, i' \to j')$ into E'
    end
  end
end

4.4.3 Customized Loss Function

The transformed graph raises a new issue in constructing the loss function. The loss function we consider for GNNs is the mean square error of the node-level predictions for all nodes in a graph.

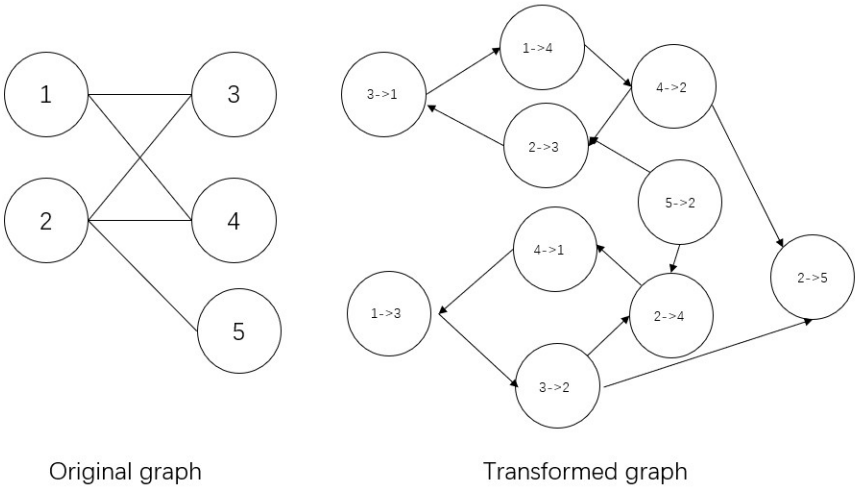


Figure 4.10: Illustration of the Edge-node Graph Transformation.

However, the predictions in the transformed graph are not unique. For example, edge (i, j) in the original graph has two corresponding nodes $j \to i$ and $i \to j$ in the transformed node set V' and we denote them as v_1 and v_2. In some sense, the model is overparameterized as there are many sets of values of $p(v_1)$ and $p(v_2)$ that minimizes the overall loss. Thus, we add a regularization to penalize the difference between $p(v_1)$ and $p(v_2)$. We define the terms related to (i, j) in the loss function as

$$(o(i,j) - \bar{p})^2 + \theta \frac{(p(v_1) - \bar{p})^2 + (p(v_2) - \bar{p})^2}{2} \tag{4.20}$$

where

$$\bar{p} = \frac{p(v_1) + p(v_2)}{2} \tag{4.21}$$

For example, for the observed edge-level statistic $o(1, 3)$ in the original graph from Figure 4.10, node $3 \to 1$ and node $1 \to 3$ in the transformed graph generate predictions $p(3 \to 1)$ and $p(1 \to 3)$. Then in the loss function, the terms related to

the observation $o(1,3)$ is

$$(o(1,3)-\bar{p})^2+\theta\frac{\big(p(3\to 1)-\bar{p}\big)^2+\big(p(1\to 3)-\bar{p}\big)^2}{2}$$

The second term is a regularization term and θ is a hyperparameter which controls the amount of penalty associated with the discrepancy between $p(v_1)$ and $p(v_2)$. The purpose of this regularization is to reduce the gap between $p(v_1)$ and $p(v_2)$ as a larger gap increases the value of the loss function.

4.5 Numerical Tests

In this section, we compare the effectiveness of our proposed GNN-TD, GNN-TU, GNN-N and GNN-E against different benchmark ML models, including the bag of nodes or bag of edges (denoted by BAG), convolutional neural network (denoted by CNN) and random forest with node-level and edge-level predictions (denoted by RF). We also consider the base GNN model which directly makes graph-level prediction (denoted by GNN-G). The benchmark models are introduced in details in Appendix C.2. The comparison is conducted on synthetic datasets generated from ATO systems and process flexibility networks. We report the node-level predictions in ATO systems and edge-level predictions in process flexibility networks. The graph-level predictions are reported in both ATO and process flexibility systems.

The above ML models are different in the level of the prediction (Table 4.1). GNN-G, CNN and BAG only work for the graph-level prediction and both their input features are graph-level. The rest of the ML models can first predict both the node-level or edge-level performances, and the aggregate the predictions for the corresponding graph-level prediction. In particular, GNN-TD, GNN-TU, GNN-E can

Table 4.1: ML Method and its Input and Output

	Method	Explanation	Decompose-then-aggregate	Transformed Graph
Our Approach	GNN-TD	GNN on directed transformed graph	Yes	Yes
	GNN-TU	GNN on undirected transformed graph	Yes	Yes
	GNN-N	Node-level GraphSage	Yes	No
	GNN-E	Edge-level GraphSage	Yes	No
Benchmark	GNN-G	Graph-level GraphSage	No	No
	RF	Random Forest	Yes	No
	BAG	Bag of nodes or bag of edges	No	No
	CNN	Convolutional neural network	No	No

Table 4.2: Datasets for Supply Chain Systems

Supply Chain Systems	Datasets		
ATO system	10x10	8x16	12x12
Process flexibility network	Obermeyer	JG	Rand10x10
	Rand6x43	Rand17x17	Rand12x21

work for edge-level prediction. GNN-N can work for node-level prediction. RF can work for both node-level and edge-level prediction.

4.5.1 Experiment Setting

We use a 5-fold nested cross-validation approach to conduct the comparison. The nested cross-validation follows a version proposed by Errica et al. (2019) and is explained in details in Appendix C.3. We consider an inductive learning setting that each dataset contains 100 graphs. All these numerical studies are tested in Python 3.7 on a 4.00 GHz Intel i7 CPU.

Data. We consider two different applications: ATO system and process flexibility network. The objective of the test on the ATO system is to compare ML models for node-level prediction. The objective of the test on the process flexibility network is to compare ML models for edge-level prediction. For each application, we consider several datasets in Table 4.2. The detailed generation of the dataset is introduced in Appendix C.4.

Hyperparameters. For hyperparameter tuning, we have a large hyperparameter

space. To avoid the exhaustive search over the entire space, we perform a randomized search cross-validation by evaluating 40 uniformly random points in the space.

We introduce the hyperparameters and its selection range below. These ranges are common values for GNN.

- Related to message passing: The number of iteration in message passing is selected from $\{0, 1, 2\}$. We select the size of the hidden layer in MLP, L, from $\{32, 64, 128\}$. For aggregate function, we select from {mean, pool, gcn}. Activation function is selected from {relu, sigmoid}.

- Related to optimization: To apply standard gradient descent, we use the learning rate to control how quickly the model is adapted to the problem. We apply a learning decay rate strategy which is time-based decay with an initial learning rate of $1e$-2 , reduce factor 0.8, and patience value 40 for all GNN type models. The batch size is the number of graphs we use in one epoch to train the GNN model, which is selected from $\{1, 2, 4, 8\}$. The number of epochs is the number of times our model go through the entire dataset, which is set as 300.

- Related to regularization: To reduce overfitting and improving the generalization, we use dropout regularization and the dropout rate is selected from $\{0, 0.1, 0.2\}$, and consider L2 regularization where the coefficient is selected from $\{0, 1e\text{-}1, 1e\text{-}2\}$. For the regularization to reduce overfitting on our transformed graph, θ is selected from $\{, 0, 1, 2\}$.

Performance measure. The performance measure is the average weighted mean absolute percentage error (WMAPE) for graph-level, node-level and edge-level predictions across graphs in the test folds. To reduce the fluctuations in our reporting, we use the average prediction over the last 20 epochs as the final prediction.

Table 4.3: Node-level Prediction Performance Comparison (WMAPE, %) for ATO System

Datasets	Our Approach	Benchmark
	GNN-N	RF
10x10	**6.53**	7.63
8x16	**9.51**	11.24
12x12	**6.77**	8.23
Average	**7.60**	9.03

4.5.2 Results

We discuss the comparison results for the two applications separately.

ATO system. The node-level comparison results are shown in Table 4.3 and the graph-level comparison results are shown in Table 4.4.

From Table 4.3, we observe the performance of different ML models for node-level predictions for ATO systems. GNN-N has a consistently lower average WMAPE (7.60%) than RF (9.03%). This is consistent with its superior performance in other contexts of node-level predictions from the ML literature.

Table 4.4 reveals the differences among ML models for the graph-level prediction. We find that models following the decompose-then-aggregate prediction approach (GNN-N with 2.36% and RF with 3.33%) have a consistently lower average WMAPE than predicting the graph-level performance (GNN-G with 3.52%, CNN with 6.32% and BAG with 6.05%). This implies the benefits of starting with node-level predictions.

Process flexibility network. From Table 4.5, we compare the ML models for edge-level predictions in process flexibility systems. GNN-TD has the lowest average WMAPE (28.00%). It is more accurate than GNN-TU for 4 out of 6 datasets, which shows that the transformed graph improves the prediction accuracy of its undirected counterpart. Note that GNN-E (41.90%) is not doing well here. This

Table 4.4: Graph-level Prediction Performance Comparison (WMAPE, %) for ATO System

Datasets	Our Approach	Benchmarks			
	GNN-N	GNN-G	RF	CNN	BAG
10x10	**2.21**	3.01	2.98	7.14	5.54
8x16	**2.75**	5.35	3.75	6.49	7.75
12x12	**2.13**	2.20	3.27	5.34	4.86
Average	**2.36**	3.52	3.33	6.32	6.05

may be because it requires an extra step to transform the node-level embeddings to edge-level embeddings, which adds more complexity to the training of the GNN model.

Table 4.6 shows the corresponding comparison for the graph-level prediction. We find that models following the decompose-then-aggregate prediction approach (GNN-TD, GNN-TU, GNN-E and RF) is more accurate than directly predicting the graph-level performance (GNN-G, CNN and BAG), except the average WMAPE for GNN-E (16.79%) is between CNN (17.68%) and BAG (14.90%) and larger than GNN-G (11.34%). The improvement in the average WMAPE from GNN-G to GNN-TD (5.10%) and GNN-TU (5.88%) also highlights the benefits of the consideration of the information of the neighborhood edges. The reason for GNN-E being less accurate than GNN-G may be still because the extra transformation in GNN-E. We also observe that GNN-TD has better performance than GNN-TU for 5 out of 6 datasets, which again shows the importance of considering the direction in the transformed graph.

In summary, we observe that it is better to follow the decompose-then-aggregate prediction approach, compared to directly predicting the graph-level performance. The base GNN model is the most accurate for node-level predictions. For edge-level predictions, the base GNN model does not perform well, but our proposed GNN-TD and GNN-TU are consistently more accurate than other ML models. These results

Table 4.5: Edge-level Prediction Performance Comparison (WMAPE) for Process Flexibility Network

Datasets	Our Approaches			Benchmark
	GNN-TD	GNN-TU	GNN-E	RF
Obermeyer	44.19	**40.80**	53.97	52.51
JG	**24.69**	25.19	39.47	47.64
Rand10x10	**21.80**	22.23	30.88	25.56
Rand6x43	28.96	31.31	45.23	**25.65**
Rand17x17	**21.64**	21.12	43.31	29.76
Rand12x21	**26.70**	28.02	30.37	38.56
Average	**28.00**	28.11	35.25	41.90

Table 4.6: Graph-level Prediction Performance Comparison (WMAPE, %) for Process Flexibility Network

Datasets	Our Appraoches			Benchmarks			
	GNN-TD	GNN-TU	GNN-E	GNN-G	RF	CNN	BAG
Obermeyer	8.43	**7.83**	19.74	9.62	7.87	14.34	13.73
JG	**4.04**	4.94	17.81	4.88	5.24	16.62	6.85
Rand10x10	**4.88**	4.95	8.72	14.94	6.86	27.50	19.63
Rand6x43	4.89	6.56	19.63	10.75	**4.16**	12.26	14.96
Rand17x17	**4.32**	4.55	24.19	14.12	5.93	17.35	16.35
Rand12x21	**4.05**	6.46	10.66	13.75	5.62	18.01	17.87
Average	**5.10**	5.88	16.79	11.34	5.95	17.68	14.90

highlight the importance of utilizing the graphical structure and the neighborhood edge information in learning and avoid the extra transformation from node embeddings to edge embeddings.

4.6 Discussion and Conclusion

The evaluation of the graphical structure in supply chain systems is important for designing and planning the system. However, the research on how to use the GNN model in operational performance prediction is limited. We investigate how to adjust the GNN model in predicting operational performances in supply chain systems with an underlying graphical structure. We show the power of GNN is great in predicting node-level prediction, but further customization is needed for edge-level prediction. Therefore, we propose an edge-node graph transformation approach to utilize the edge features from the neighborhood edges and directly predicting the edge-level performance. We show these changes greatly improve the accuracy of the GNN model in edge-level prediction.

We next discuss some potential directions to extend our study. In this paper, we only consider ATO systems and process flexibility networks. We hope to steer some of the GNN research towards other supply chain systems. In our application of the GNN model, we assume every node and edge has equal weight, which may not be true in reality. For example, some warehouse nodes are large and they should have more influence in their neighborhoods. It is interesting to consider how to apply the GAN model to deal with the weights.

Chapter 5

Conclusion

This dissertation develops theory and data-driven tools for the design and performance prediction of supply chain systems with graphical structures. It consists of three essays. The first essay discusses the design of effective inventory policies for continuous-time Assemble-to-Order (ATO) systems with non-identical replenishment lead times. For a simple ATO system, we establish an asymptotically optimal policy when demand volume is high. Building on the insights from this analysis, we further develop simple and effective policies to implement. In particular, we propose an assembly decomposition for simple systems and a distribution decomposition for general systems to construct two different heuristic policies and numerically show that they outperform several benchmarks in the literature. The second essay develops a predict-then-optimize approach for e-commerce transportation network design in which the decision-maker for the network design does not know the network flow decision for any given network, because the latter is made by a separate entity and difficult to model. We first develop a network flow predictor for any given network decision. We find that first predicting edge-level shipment flow and then aggregating the predictions together is more effective than predicting the total shipment flow on the entire network. We then develop a linear-time approximation algorithm to compute the optimal network design with a performance guarantee. The third essay adopts a more advanced machine learning model, the GNN model, to predict operational performance in supply chain systems by leveraging their graphical structures. Even though GNN can make a direct prediction of the graph-level (i.e., the supply

chain system) performance, we find it is more accurate to use the decompose-then-aggregate prediction approach for predicting the system performance. For edge-level prediction, we also propose a novel graph transformation approach which significantly improves the prediction accuracy.

Appendix A

Appendix for Chapter 2

A.1 Appendix: Proofs of the Lemmas and Theorems

A.1.1 Proof of Proposition 2

To prove Proposition 2 (and Theorem 1), we need the following lemma related to the linear program properties. Consider a linear program problem:

$$\begin{aligned}
\theta(z_1, z_2) &= \min_{x_i, y^K \geq 0} \left\{ b^1 y^1 + b^{12} y^{12} + b^2 y^2 + h_1 x_1 + h_2 x_2 \right\} && \text{(A.1)} \\
\text{s.t.} \quad & x_1 = z_1 + y^1 + y^{12} \geq 0; && \text{(A.2)} \\
& x_2 = z_2 + y^2 + y^{12} \geq 0; && \text{(A.3)} \\
& y^1, y^2, y^{12} \geq 0. && \text{(A.4)}
\end{aligned}$$

Lemma 1. *$\theta(z_1, z_2)$ given by* (A.1)-(A.4) *is Lipschitz continuous and convex in (z_1, z_2). Moreover, the Lipschitz continuity coefficient depends only on b^1, b^2, b^{12}, h_1 and h_2.*

Proof. First note that (A.1)-(A.4) are equivalent to

$$\begin{aligned}
\theta(z_1, z_2) &= \min_{y^K \geq 0} \left\{ (h_1 + b^1) y^1 + (h_1 + h_2 + b^{12}) y^{12} + (h_2 + b^2) y^2 + h_1 z_1 + h_2 z_2 \right\} && \text{(A.5)} \\
\text{s.t.} \quad & y^1 + y^{12} \geq z_1^-; && \text{(A.6)} \\
& y^2 + y^{12} \geq z_2^-; && \text{(A.7)} \\
& y^1, y^2, y^{12} \geq 0. && \text{(A.8)}
\end{aligned}$$

The Lipschitz continuity of $\theta(z_1, z_2)$ directly follows from Proposition 1. For the Lipschitz continuity of the solutions of the more general linear program, see Mangasarian and Shiau (1987).

Now consider the convexity. Its proof may be hidden in some book or paper, for completeness, we provide a proof here. For $\alpha, \beta \in [0, 1]$ with $\alpha + \beta = 1$, to prove

$$\theta(\alpha z_1 + \beta \hat{z}_1, \alpha z_2 + \beta \hat{z}_2) \leq \alpha\theta(z_1, z_2) + \beta\theta(\hat{z}_1, \hat{z}_2), \tag{A.9}$$

let (y_*^1, y_*^2, y_*^{12}) and $(\hat{y}_*^1, \hat{y}_*^2, \hat{y}_*^{12})$ be the solutions of (A.5)-(A.8) corresponding to (z_1, z_2) and $(\hat{z}_1, \hat{z}_2)$, respectively. Then from

$$\begin{aligned} y_*^1 + y_*^{12} \geq z_1^-, \quad y_*^2 + y_*^{12} \geq z_2^-; \\ \hat{y}_*^1 + \hat{y}_*^{12} \geq \hat{z}_1^-, \quad \hat{y}_*^2 + \hat{y}_*^{12} \geq \hat{z}_2^-, \end{aligned}$$

we have that

$$\begin{aligned} \alpha y_*^1 + \beta \hat{y}_*^1 + \alpha y_*^{12} + \beta \hat{y}_*^{12} \geq \alpha z_1^- + \beta \hat{z}_1^- \geq (\alpha z_1 + \beta \hat{z}_1)^-; \\ \alpha y_*^2 + \beta \hat{y}_*^2 + \alpha y_*^{12} + \beta \hat{y}_*^{12} \geq \alpha z_2^- + \beta \hat{z}_2^- \geq (\alpha z_2 + \beta \hat{z}_2)^-. \end{aligned}$$

Thus we know that $(\alpha y_*^1 + \beta \hat{y}_*^1, \alpha y_*^2 + \beta \hat{y}_*^2, \alpha y_*^{12} + \beta \hat{y}_*^{12})$ is a feasible solution for $(\alpha z_1 + \beta \hat{z}_1, \alpha z_2 + \beta \hat{z}_2)$. This, by noticing

$$\begin{aligned} \alpha\theta(z_1, z_2) = (h_1 + b^1)\alpha y_*^1 + (h_1 + h_2 + b^{12})\alpha y_*^{12} + (h_2 + b^2)\alpha y_*^2 + h_1\alpha z_1 + h_2\alpha z_2; \\ \beta\theta(\hat{z}_1, \hat{z}_2) = (h_1 + b^1)\beta \hat{y}_*^1 + (h_1 + h_2 + b^{12})\beta \hat{y}_*^{12} + (h_2 + b^2)\beta \hat{y}_*^2 + h_1\beta \hat{z}_1 + h_2\beta \hat{z}_2, \end{aligned}$$

implies that (A.9) holds. Hence, we have the convexity. □

Proof of Proposition 2: Consider a relaxed linear program of (2.19)-(2.22) by dropping the constrain (2.22):

$$\min_{B^K \geq 0} \left\{ b^1 B^1 + b^{12} B^{12} + b^2 B^2 + h_1 I_1 + h_2 I_2 \right\} \tag{A.10}$$

$$\text{s.t.} \quad I_1 = I_1^n(0) + O_1^n(\ell\delta^n - L_1) - \mathcal{D}^{1,n}(\ell\delta^n) - \mathcal{D}^{12,n}(\ell\delta^n) + B^1 + B^{12} \geq 0, \tag{A.11}$$

$$I_2 = I_2^n(0) + O_2^n(\ell\delta^n - L_2) - \mathcal{D}^{2,n}(\ell\delta^n) - \mathcal{D}^{12,n}(\ell\delta^n) + B^2 + B^{12} \geq 0 \tag{A.12}$$

Let $\{(\hat{I}_i^n(\ell\delta^n), \hat{B}^{K,n}(\ell\delta^n), i \in \mathcal{I}, K \in \mathcal{K}) : 0 \leq \ell \leq \lfloor \frac{T}{\delta^n} \rfloor + 1\}$ be the above solution. Then from (A.1)-(A.4), for $\ell = 0, \ldots, \lfloor \frac{T}{\delta^n} \rfloor + 1$,

$$\theta\Big(I_1^n(0) + O_1^n(\ell\delta^n - L_1) - \mathcal{D}^{1,n}(\ell\delta^n) - \mathcal{D}^{12,n}(\ell\delta^n), I_2^n(0) + O_2^n(\ell\delta^n - L_2) \tag{A.13}$$

$$-\mathcal{D}^{2,n}(\ell\delta^n) - \mathcal{D}^{12,n}(\ell\delta^n) \Big)$$

$$= \varphi\Big(I_1^n(0) + O_1^n(\ell\delta^n - L_1) - \mathcal{D}^{1,n}(\ell\delta^n) - \mathcal{D}^{12,n}(\ell\delta^n), I_2^n(0) + O_2^n(\ell\delta^n - L_2) \tag{A.14}$$

$$-\mathcal{D}^{2,n}(\ell\delta^n) - \mathcal{D}^{12,n}(\ell\delta^n) \Big)$$

$$= \varphi\Big(I_1^n(0) + O_1^n(\ell\delta^n - L_1) - \mathcal{D}_1^n(\ell\delta^n), I_2^n(0) + O_2^n(\ell\delta^n - L_2) - \mathcal{D}_2^n(\ell\delta^n) \Big) \tag{A.15}$$

For $t \in [\ell\delta^n, (\ell+1)\delta^n)$, by Lemma 1, there exists a $C > 0$ such that

$$\Big| \varphi\Big(I_1^n(0) + O_1^n(t - L_1) - \mathcal{D}_1^n(t), I_2^n(0) + O_2^n(t - L_2) - \mathcal{D}_2^n(t) \Big)$$

$$-\varphi\Big(I_1^n(0) + O_1^n(\ell\delta^n - L_1) - \mathcal{D}_1^n(\ell\delta^n), I_2^n(0) + O_2^n(\ell\delta^n - L_2) - \mathcal{D}_2^n(\ell\delta^n) \Big) \Big|$$

$$\leq C \cdot \sum_{i \in \mathcal{I}} \Big(\Big| O_i^n(t - L_i) - O_i^n(\ell\delta^n - L_i) \Big| + \Big| \mathcal{D}_i^n(t) - \mathcal{D}_i^n(\ell\delta^n) \Big| \Big). \tag{A.16}$$

By the stationarity and independent increments of $\{(\mathcal{D}^{K,n}(t), K \in \mathcal{K}), t \geq 0\}$ and

$\mathsf{E}\left[\mathcal{D}^{K,n}\left(\frac{1}{n}\right)\right]^5 < \infty$ for $K \in \mathcal{K}$, we know that

$$\frac{1}{\sigma^K\sqrt{n}}\Big(\mathcal{D}^{K,n}(t) - \lambda^K nt\Big) \Longrightarrow \mathcal{N}^K(t) \ \text{ on } D[0,\infty).$$

Here $\{\mathcal{N}^K(t), t \geq 0\}$ is a standard Brownian motion. With the help of the Skorokhod theorem, further we have that with probability one,

$$\lim_{n\to\infty} \frac{1}{\sigma^K\sqrt{n}}\Big(\mathcal{D}^{K,n}(t) - \lambda^K nt\Big) = \mathcal{N}^K(t). \tag{A.17}$$

This and the continuity of the sample path of $\{(B^K(t), K \in \mathcal{K}), t \geq 0\}$ imply that for $t \in [\ell\delta^n, (\ell+1)\delta^n)$,

$$\begin{aligned}
\frac{1}{\sqrt{n}}\Big|\mathcal{D}_i^n(t) - \mathcal{D}_i^n(\ell\delta^n)\Big| &= \frac{1}{\sqrt{n}}\Big|\mathcal{D}_i^n(t) - \mathcal{D}_i^n(\ell\delta^n) - \lambda_i n(t-\ell\delta^n) + \lambda_i n(t-\ell\delta^n)\Big| \\
&\leq \Big|\frac{1}{\sqrt{n}}\Big(\mathcal{D}_i^n(t) - \lambda_i nt\Big) - \frac{1}{\sqrt{n}}\Big(\mathcal{D}_i^n(\ell\delta^n) - \lambda_i n\ell\delta^n\Big)\Big| \\
&\quad + \lambda_i\sqrt{n}\delta^n \\
&\to 0 \ \text{ with probability one.}
\end{aligned} \tag{A.18}$$

Hence, it follows from (A.16) that for $t \in [\ell\delta^n, (\ell+1)\delta^n)$,

$$\begin{aligned}
&\frac{1}{\sqrt{n}}\Big|\varphi\Big(I_1^n(0) + O_1^n(t-L_1) - \mathcal{D}_1^n(t), I_2^n(0) + O_2^n(t-L_2) - \mathcal{D}_2^n(t)\Big) \\
&\quad -\varphi\Big(I_1^n(0) + O_1^n(\ell\delta^n - L_1) - \mathcal{D}_1^n(\ell\delta^n), I_2^n(0) + O_2^n(\ell\delta^n - L_2) - \mathcal{D}_2^n(\ell\delta^n)\Big)\Big| \\
&\to 0 \ \text{ with probability one.}
\end{aligned} \tag{A.19}$$

If define

$$\begin{aligned}\hat{I}_i^n(t) &= \hat{I}_i^n(\ell\delta^n) \text{ for } i \in \mathcal{I} \text{ and } t \in [\ell\delta^n, (\ell+1)\delta^n);\\ \hat{B}^{K,n}(t) &= \hat{B}^{K,n}(\ell\delta^n) \text{ for } K \in \mathcal{K} \text{ and } t \in [\ell\delta^n, (\ell+1)\delta^n),\end{aligned}$$

then by (A.15) and (A.19),

$$\begin{aligned}&\limsup_{n\to\infty} \frac{1}{\sqrt{n}}\frac{1}{T}\mathsf{E}\int_0^T \Big(\sum_{i\in\mathcal{I}} h_i \cdot \hat{I}_i^n(t) + \sum_{K\in\mathcal{K}} b^K \cdot \hat{B}^{K,n}(t)\Big)\mathsf{d}t\\ &= \limsup_{n\to\infty} \frac{1}{\sqrt{n}}\frac{1}{T}\mathsf{E}\sum_{\ell=0}^{\lfloor \frac{T}{\delta^n}\rfloor}\int_{\ell\delta^n}^{(\ell+1)\delta^n} \varphi\Big(I_1^n(0) + O_1^n(\ell\delta^n - L_1) - \mathcal{D}_1^n(\ell\delta^n), I_2^n(0)\\ &\quad + O_2^n(\ell\delta^n - L_2) - \mathcal{D}_2^n(\ell\delta^n)\Big)\mathsf{d}t\\ &= \limsup_{n\to\infty} \frac{1}{\sqrt{n}}\frac{1}{T}\mathsf{E}\sum_{\ell=0}^{\lfloor \frac{T}{\delta^n}\rfloor}\int_{\ell\delta^n}^{(\ell+1)\delta^n} \varphi\Big(I_1^n(0) + O_1^n(t - L_1) - \mathcal{D}_1^n(t), I_2^n(0)\\ &\quad + O_2^n(t - L_2) - \mathcal{D}_2^n(t)\Big)\mathsf{d}t\\ &= \limsup_{n\to\infty} \frac{1}{\sqrt{n}}\frac{1}{T}\mathsf{E}\int_0^T \varphi\Big(I_1^n(0) + O_1^n(t - L_1) - \mathcal{D}_1^n(t), I_2^n(0) + O_2^n(t - L_2)\\ &\quad - \mathcal{D}_2^n(t)\Big)\mathsf{d}t.\end{aligned}$$

To get the proposition proved, it suffices to show that

$$\begin{aligned}&\limsup_{n\to\infty} \frac{1}{\sqrt{n}}\frac{1}{T}\mathsf{E}\int_0^T \Big(\sum_{i\in\mathcal{I}} h_i \cdot I_i^n(t, \pi_*^n) + \sum_{K\in\mathcal{K}} b^K \cdot B^{K,n}(t, \pi_*^n)\Big)\mathsf{d}t\\ &\quad = \limsup_{n\to\infty} \frac{1}{\sqrt{n}}\frac{1}{T}\mathsf{E}\int_0^T \Big(\sum_{i\in\mathcal{I}} h_i \cdot \hat{I}_i^n(t) + \sum_{K\in\mathcal{K}} b^K \cdot \hat{B}^{K,n}(t)\Big)\mathsf{d}t. \qquad \text{(A.20)}\end{aligned}$$

By again the stationarity and independent increments of $\{(\mathcal{D}^{K,n}(t), K \in \mathcal{K}), t \geq 0\}$ and $\mathsf{E}\left[\mathcal{D}^{K,n}\left(\frac{1}{n}\right)\right]^5 < \infty$ for $K \in \mathcal{K}$, we know that, by Ata and Kumar (2005), for

any finite constant $\alpha > 0$, there exists a constant $\beta > 0$ such that

$$\mathsf{Pr}\Big(\max_{0\le\ell\le\lfloor T/\delta^n\rfloor+1}\max_{K\in\mathcal{K}}\Big|\mathcal{D}^{K,n}((\ell+1)\delta^n)-\mathcal{D}^{K,n}(\ell\delta^n)-\lambda^K n\delta^n\Big|<\alpha n^{1/3}\Big)\ge 1-\beta n^{-1/6}.$$

By the Lipschitz continuity of $\hat{I}_i^n(t)$ and $\hat{B}^{K,n}(t)$ given by (A.10)-(A.12) from Lemma 1, following the proof of Proposition 4.1 in Plambeck and Ward (2006), we can show that there exists a constant $\beta > 0$ such that

$$\mathsf{Pr}\left(\hat{B}^{K,n}(\ell\delta^n)=B^{K,n}(\ell\delta^n,\pi_*^n),\ell=1,\cdots,\left\lfloor\frac{T}{\delta^n}\right\rfloor\right)\ge 1-\beta n^{-1/6}. \tag{A.21}$$

Then (A.20) directly follows from (A.21). □

A.1.2 Proof of Theorem 1

We first establish the following lemma:

Lemma 2. *The function $\eta_1(\cdot)$ defined by (2.31) is increasing and $\mathsf{d}\eta_1(y)/\mathsf{d}y \le 1$.*

Proof. Let $\psi(u,v)$ represent the density function of bivariate normal (Θ_1,Θ_2). Then from Proposition 1, we can obtain the closed-form expression for $\varphi(y_1-\Theta_1, y_2-\Theta_2)$. Here we present the proof for case $c^2 \vee c^1 < c^{12} < c^1 + c^2$ and other cases can be analogously analyzed. Note that

$$\begin{aligned}\varphi(y_1-\Theta_1,y_2-\Theta_2) &= h_1\cdot(y_1-\Theta_1)+h_2\cdot(y_2-\Theta_2)\\ &\quad +c^{12}\cdot\Big((y_1-\Theta_1)^-\wedge(y_2-\Theta_2)^-\Big)\\ &\quad +c^1\cdot\Big((y_1-\Theta_1)^--(y_2-\Theta_2)^-\Big)^+\\ &\quad +c^2\cdot\Big((y_2-\Theta_2)^--(y_1-\Theta_1)^-\Big)^+.\end{aligned}$$

A straightforward calculation yields that

$$\begin{aligned}
\frac{\partial^2 H(y_1,y_2)}{\partial y_1^2} &= c^{12}\Big(\int_{y_2}^{\infty}\psi(y_1,v)\mathsf{d}v-\int_{y_1}^{\infty}\phi(u,u-(y_1-y_2))\mathsf{d}u\Big)\\
&\quad +c^{1}\Big(\int_{-\infty}^{y_2}\psi(y_1,v)\mathsf{d}v+\int_{y_1}^{\infty}\phi(u,u-(y_1-y_2))\mathsf{d}u\Big)\\
&\quad +c^{2}\Big(-\int_{y_2}^{\infty}\psi(y_1,v)\mathsf{d}v+\int_{y_2}^{\infty}\phi(v+(y_1-y_2),v)\mathsf{d}v\Big); \qquad \text{(A.22)}\\
\frac{\partial^2 H(y_1,y_2)}{\partial y_1\partial y_2} &= c^{12}\int_{y_1}^{\infty}\phi(u,u-(y_1-y_2))\mathsf{d}u-c^{1}\int_{y_1}^{\infty}\phi(u,u-(y_1-y_2))\mathsf{d}u\\
&\quad -c^{2}\int_{y_2}^{\infty}\phi(v+(y_1-y_2),v)\mathsf{d}v.
\end{aligned}$$

By

$$\int_{y_2}^{\infty}\phi(v+(y_1-y_2),v)\mathsf{d}v=\int_{y_1}^{\infty}\phi(u,u-(y_1-y_2))\mathsf{d}u,$$

we have

$$\begin{aligned}
\frac{\partial^2 H(y_1,y_2)}{\partial y_1\partial y_2} &= c^{12}\int_{y_1}^{\infty}\phi(u,u-(y_1-y_2))\mathsf{d}u-c^{1}\int_{y_1}^{\infty}\phi(u,u-(y_1-y_2))\mathsf{d}u\\
&\quad -c^{2}\int_{y_1}^{\infty}\phi(u,u-(y_1-y_2))\mathsf{d}u\\
&= (c^{12}-c^{1}-c^{2})\int_{y_1}^{\infty}\phi(u,u-(y_1-y_2))\mathsf{d}u<0. \qquad \text{(A.23)}
\end{aligned}$$

Using (A.22),

$$\begin{aligned}
\frac{\partial^2 H(y_1,y_2)}{\partial y_1^2} &= -(c^{12}-c^{1}-c^{2})\int_{y_1}^{\infty}\phi(u,u-(y_1-y_2))\mathsf{d}u+c^{1}\int_{-\infty}^{y_2}\psi(y_1,v)\mathsf{d}v\\
&\quad +(c^{12}-c^{2})\int_{y_2}^{\infty}\psi(y_1,v)\mathsf{d}v. \qquad \text{(A.24)}
\end{aligned}$$

It follows from (A.23)-(A.24) and the positivity of

$$c^1 \int_{-\infty}^{y_2} \psi(y_1, v)\mathsf{d}v + (c^{12} - c^2) \int_{y_2}^{\infty} \psi(y_1, v)\mathsf{d}v$$

that

$$\frac{\mathsf{d}\eta_1(y)}{\mathsf{d}y} = -\frac{\partial^2 H(y_1, y_2)}{\partial y_1 \partial y_2} \Big/ \frac{\partial^2 H(y_1, y_2)}{\partial y_1^2} < 1,$$

which gives the lemma. □

Proof of Theorem 1*:* First we prove that the PRP allocation rule and the CBS-type replenishment policy $(IP_1^n(t), IP_2^n(t))$ defined by (2.34)-(2.35) belongs to $\mathcal{A}^n$. To that end, it suffices to show that for small $\varepsilon > 0$ if

$$\eta_2 + \sqrt{n}\lambda_2\Delta - \frac{\mathcal{D}^{12,n}(t-\Delta, t]}{\sqrt{n}} > \eta_2 + \sqrt{n}\lambda_2\Delta - \frac{\mathcal{D}^{12,n}(t+\varepsilon-\Delta, t+\varepsilon]}{\sqrt{n}},$$

then

$$\begin{aligned}
&\sqrt{n} \cdot \eta_1\Big(\eta_2 + \sqrt{n}\lambda_2\Delta - \frac{\mathcal{D}^{12,n}(t-\Delta, t]}{\sqrt{n}}\Big) - \Big(\mathcal{D}^{12,n}(t+\varepsilon-\Delta, t+\varepsilon] \\
&\quad -\mathcal{D}^{12,n}(t-\Delta, t]\Big) \\
&\leq \sqrt{n} \cdot \eta_1\Big(\eta_2 + \sqrt{n}\lambda_2\Delta - \frac{\mathcal{D}^{12,n}(t+\varepsilon-\Delta, t+\varepsilon]}{\sqrt{n}}\Big). \qquad \text{(A.25)}
\end{aligned}$$

This directly follows from Lemma 2.

Now we establish that for any $\pi^n \in \mathcal{A}^n$ and $T > 0$,

$$\limsup_{n\to\infty} \frac{1}{\sqrt{n}} \frac{1}{T} \mathsf{E} \int_0^T \Big(\sum_{i\in\mathcal{I}} h_i \cdot I_i^n(t, \pi_*^n) + \sum_{K\in\mathcal{K}} b^K \cdot B^{K,n}(t, \pi_*^n) \Big) \mathsf{d}t$$
$$\leq \limsup_{n\to\infty} \frac{1}{\sqrt{n}} \frac{1}{T} \mathsf{E} \int_0^T \Big(\sum_{i\in\mathcal{I}} h_i \cdot I_i^n(t, \pi^n) + \sum_{K\in\mathcal{K}} b^K \cdot B^{K,n}(t, \pi^n) \Big) \mathsf{d}t. \quad \text{(A.26)}$$

In view of Proposition 2, for $t \geq L + \Delta$,

$$\begin{aligned}
&\lim_{n\to\infty} \frac{1}{\sqrt{n}} \mathsf{E} \Big(\sum_{i\in\mathcal{I}} h_i \cdot I_i^n(t, \pi_*^n) + \sum_{K\in\mathcal{K}} b^K \cdot B^{K,n}(t, \pi_*^n) \Big) \\
&= \lim_{n\to\infty} \mathsf{E}\varphi \Big(\eta_1 \Big(\eta_2 - \frac{\mathcal{D}^{12,n}(t-L-\Delta, t-L] - n\lambda_2\Delta}{\sqrt{n}} \Big) - \frac{\mathcal{D}_1^n(t-L, t] - n\lambda_1 L}{\sqrt{n}}; \\
&\qquad\qquad \eta_2 - \frac{\mathcal{D}_2^n(t-L-\Delta, t-L] - n\lambda_2\Delta}{\sqrt{n}} - \frac{\mathcal{D}_2^n(t-L, t] - n\lambda_2}{\sqrt{n}} \Big) \\
&= H_2(\eta_2). \qquad \text{(A.27)}
\end{aligned}$$

On the other hand, for any $\pi^n \in \mathcal{A}^n$, define

$$\begin{aligned}
IP_1^\infty(t - L_1) &= \liminf_{n\to\infty} \frac{IP_1^n(t - L_1, \pi^n) - \lambda_1 n L_1}{\sqrt{n}}; \\
IP_2^\infty(t - L_2) &= \liminf_{n\to\infty} \frac{IP_2^n(t - L_2, \pi^n) - \lambda_2 n L_2}{\sqrt{n}}; \\
\widehat{\mathcal{D}}_1^n(t - L_1, t] &= \frac{\mathcal{D}_1^n(t - L_1, t] - \lambda_1 n L_1}{\sqrt{n}}; \\
\widehat{\mathcal{D}}_2^n(t - L_1, t] &= \frac{\mathcal{D}_2^n(t - L_1, t] - \lambda_2 n L_1}{\sqrt{n}}; \\
\widehat{\mathcal{D}}_2^n(t - L_2, t - L_1] &= \frac{\mathcal{D}_2^n(t - L_2, t - L_1] - \lambda_2 n \Delta}{\sqrt{n}}.
\end{aligned}$$

Then for $t \geq L + \Delta$,

$$
\begin{aligned}
&\liminf_{n\to\infty} \frac{1}{\sqrt{n}} \mathsf{E}\Big(\sum_{i\in\mathcal{I}} h_i \cdot I_i^n(t,\pi^n) + \sum_{K\in\mathcal{K}} b^K \cdot B^{K,n}(t,\pi^n) \Big) \\
&\geq \liminf_{n\to\infty} \mathsf{E}\varphi\Big(\frac{IP_1^n(t-L_1,\pi^n) - \lambda_1 n L_1}{\sqrt{n}} - \widehat{\mathcal{D}}_1^n(t-L_1,t], \\
&\qquad\qquad \frac{IP_2^n(t-L_2,\pi^n) - \lambda_2 n L_2}{\sqrt{n}} - \widehat{\mathcal{D}}_2^n(t-L_2,t-L_1] - \widehat{\mathcal{D}}_2^n(t-L_1,t] \Big) \\
&= \limsup_{n\to\infty} \mathsf{E}\Big\{ \mathsf{E}\Big(\mathsf{E}\Big[\varphi\Big(\frac{IP_1^n(t-L_1,\pi^n) - \lambda_1 n L_1}{\sqrt{n}} - \widehat{\mathcal{D}}_1^n(t-L_1,t], \\
&\qquad \frac{IP_2^n(t-L_2,\pi^n) - \lambda_2 n L_2}{\sqrt{n}} - \widehat{\mathcal{D}}_2^n(t-L_2,t-L_1] - \widehat{\mathcal{D}}_2^n(t-L_1,t] \Big) \Big| \mathcal{F}_{t-L}^n \Big] \Big| \\
&\qquad \mathcal{F}_{t-L-\Delta}^n \Big) \Big\} \\
&\geq \mathsf{E}\Big\{ \mathsf{E}\Big(\mathsf{E}\Big[\varphi\Big(IP_1^\infty(t-L) - \Theta^1(t) - \Theta^{12}(t), IP_2^\infty(t-L-\Delta) \\
&\qquad\qquad -\Theta^2(t) - \Theta^{12}(t) - \Delta\Theta^{12}(t-L) - \Delta\Theta^2(t-L) \Big) \Big| \mathcal{F}_{t-L}^\infty \Big] \Big| \mathcal{F}_{t-L-\Delta}^\infty \Big) \Big\} \\
&\geq H_2(\eta_2). \qquad \text{(A.28)}
\end{aligned}
$$

Hence, (A.26) directly follows from (A.27)-(A.28). □

A.1.3 Proof of Theorem 3

We first look at the sample-path lower bound (2.14) conditional on $\{N_i(t,\pi), i \in \mathcal{I}\}$,

$$
\begin{aligned}
\min_{y^K, K\in\mathcal{K}} \quad & \sum_{K\in\mathcal{K}} c^K \cdot y^K \qquad \text{(A.29)} \\
\text{s.t.} \quad & y^K \geq 0, \ K \in \mathcal{K}; \\
& \sum_{K\in\mathcal{K}_i} y^K \geq [N_i(t,\pi)]^-, \ i \in \mathcal{I},
\end{aligned}
$$

and its dual problem

$$\begin{aligned} \max_{v_i, i\in\mathcal{I}} \quad & \sum_{i\in\mathcal{I}} v_i \cdot [N_i(t,\pi)]^- \\ \text{s.t.} \quad & v_i \geq 0, \quad i \in \mathcal{I}; \\ & \sum_{i\in K} v_i \leq c^K, \quad K \in \mathcal{K}. \end{aligned} \tag{A.30}$$

Denote the optimal solution to (A.29) and (A.30) as $\{y^{K*}, K \in \mathcal{K}\}$ and $\{v_i^*, i \in \mathcal{I}\}$, respectively. We first establish the lower bounds. Since $b^K \geq \sum_{i\in K} h_i$, $\sum_{i\in K} v_i^\ell = \sum_{i\in K} 2h_i \leq \sum_{i\in K} h_i + b^K = c^K, K \in \mathcal{K}$, and $v_i^\ell \geq 0, i \in \mathcal{I}$, i.e., v_i^ℓ is a feasible solution to the dual problem. By the weak duality, $\sum_{K\in\mathcal{K}} c^K \cdot y^{K*} \geq \sum_{i\in\mathcal{I}} v_i^\ell \cdot [N_i(t,\pi)]^-$. Therefore, we establish a lower bound of inventory cost

$$\begin{aligned} & \sum_{i\in\mathcal{I}} h_i \cdot N_i(t,\pi) + \sum_{K\in\mathcal{K}} c^K \cdot y^{K*} \\ & \quad \geq \sum_{i\in\mathcal{I}} h_i \cdot N_i(t,\pi) + \sum_{i\in\mathcal{I}} v_i^\ell \cdot [N_i(t,\pi)]^- \\ & \quad = \sum_{i\in\mathcal{I}} h_i \cdot [N_i(t,\pi)]^+ + \sum_{i\in\mathcal{I}} (v_i^\ell - h_i) \cdot [N_i(t,\pi)]^-, \end{aligned} \tag{A.31}$$

where the equation is from the fact that $N_i(t,\pi) = [N_i(t,\pi)]^+ - [N_i(t,\pi)]^-$.

Next, we establish the upper bounds. We have

$$\begin{aligned} & \sum_{i\in\mathcal{I}} h_i \cdot N_i(t,\pi) + \sum_{K\in\mathcal{K}} c^K \cdot y^{K*} \\ & \quad = \sum_{i\in\mathcal{I}} h_i \cdot N_i(t,\pi) + \sum_{i\in\mathcal{I}} v_i^* \cdot [N_i(t,\pi)]^- \\ & \quad \leq \sum_{i\in\mathcal{I}} h_i \cdot N_i(t,\pi) + \sum_{i\in\mathcal{I}} v_i^u \cdot [N_i(t,\pi)]^- \\ & \quad = \sum_{i\in\mathcal{I}} h_i \cdot [N_i(t,\pi)]^+ + \sum_{i\in\mathcal{I}} (v_i^u - h_i) \cdot [N_i(t,\pi)]^-, \end{aligned} \tag{A.32}$$

where the first equation follows from strong duality and the inequality follows from

$v_i^* \leq \min\{c^K,\ K \in \mathcal{K}_\rangle\} = \sqsubseteq_\rangle^\sqcap$ for any $i \in \mathcal{I}$.

Under the optimal IBS policy π_{IBS}^* with base-stock levels $(s_1^*, \ldots, s_m^*)$, we have

$$\begin{aligned}
\underline{C}(t, \pi_{\text{IBS}}^*) &= \sum_{i\in\mathcal{I}} h_i \cdot \mathsf{E}[s_i^* - \Theta_i] + \sum_{i\in\mathcal{I}} \mathsf{E}[v_i^* \cdot (s_i^* - \Theta_i)^-] \\
&\leq \sum_{i\in\mathcal{I}} h_i \cdot \mathsf{E}[s_i^u - \Theta_i] + \sum_{i\in\mathcal{I}} \mathsf{E}[v_i^* \cdot (s_i^u - \Theta_i)^-] \\
&\leq \sum_{i\in\mathcal{I}} h_i \cdot \mathsf{E}[s_i^u - \Theta_i] + \sum_{i\in\mathcal{I}} \mathsf{E}[v_i^u \cdot (s_i^u - \Theta_i)^-] \\
&= \min_{\mathbf{s}} C^u(\mathbf{s}) \qquad \text{(A.33)}
\end{aligned}$$

The first inequality is because $\{s_i^u, i \in \mathcal{I}\}$ is not the optimal IBS base-stock levels. The second inequality follows the limit version of inequality (A.32) in the nth system. We also have

$$\begin{aligned}
\underline{C}(t, \pi_{\text{IBS}}^*) &= \sum_{i\in\mathcal{I}} h_i \cdot \mathsf{E}[s_i^* - \Theta_i] + \sum_{i\in\mathcal{I}} \mathsf{E}[v_i^* \cdot (s_i^* - \Theta_i)^-] \\
&\geq \sum_{i\in\mathcal{I}} h_i \cdot \mathsf{E}[s_i^* - \Theta_i] + \sum_{i\in\mathcal{I}} \mathsf{E}[v_i^\ell \cdot (s_i^* - \Theta_i)^-] \\
&\geq \sum_{i\in\mathcal{I}} h_i \cdot \mathsf{E}[s_i^l - \Theta_i] + \sum_{i\in\mathcal{I}} \mathsf{E}[v_i^\ell \cdot (s_i^l - \Theta_i)^-] \\
&= \min_{\mathbf{s}} C^l(\mathbf{s}) \qquad \text{(A.34)}
\end{aligned}$$

The first inequality follows the limit version of inequality (A.31) in the nth system. The second inequality is because $\{s_i^l, i \in \mathcal{I}\}$ is the optimal newsvendor order quantities when we replace v_i^* by v_i^ℓ for every component-i.

Thus, by (A.33) and (A.34), one has $\underline{C}^* = \min_{\mathbf{s}} C^l(\mathbf{s}) \leq \mathsf{E}[\underline{C}(t, \pi_{\text{IBS}}^*)] \leq \min_{\mathbf{s}} C^u(\mathbf{s}) = \overline{C}^*$. □

A.1.4 Proof of Theorem 4

Obviously, $v_i^{NV} \in [v_i^{\ell}, v_i^u], i \in \mathcal{I}$, thus, $s_i^{\mathrm{NV}} \in [s_i^l, s_i^u], i \in \mathcal{I}$ and $C^{\mathrm{NV}} \in [\underline{C}^*, \overline{C}^*]$. $\square$

A.2 Sample-path Analysis of the General M- and W-system

Similar to the M-system, the sample-path analysis of the general M- and W-system depends on the cost structure, we present these analysis in following. To simplify the notation, we use $m+1$ to index the super product in the general M-system, and $1, \ldots, m-1$ to index the products in the general W-system. We also label product-$(m-1)$ be the product with the lowest value of the unit inventory cost in the general W-system, i.e., $c^{m-1} = \min\{c^i : i = 1, 2, \ldots, m-1\}$.

Proposition 1. (Sample-path Lower Bound for the General M-system) *If there exists an admissible policy $\widehat{\pi}$ such that the conditional sample-path lower bound given by the linear program* (2.11)-(2.13) *can be achieved for the general M-system, it has the following sample-path properties.*

(i) *If* $\sum_{K=1}^{m} c^K = c^{m+1}$, *then*

$$B^K(t, \hat{\pi}) + B^{m+1}(t, \hat{\pi}) = [N_K(t, \hat{\pi})]^-, \quad K = 1, 2, ..., m, \tag{A.1}$$

(ii) *If* $\sum_{K=1}^{m} c^K < c^{m+1}$, *then*

$$B^{m+1}(t, \hat{\pi}) = 0, \tag{A.2}$$

$$B^K(t, \hat{\pi}) = [N_K(t, \hat{\pi})]^-, \quad K = 1, 2, ..., m, \tag{A.3}$$

(iii) *If* $\min\{c^K, K = 1, 2, ..., m\} < c^{m+1} < \sum_{K=1}^{m} c^K$, *then*

$$B^{m+1}(t, \hat{\pi}) = \max_{J \in \mathcal{J}} \{\min_{K \in J} [N_K(t, \hat{\pi})]^-\}, \tag{A.4}$$

$$B^K(t, \hat{\pi}) = [B^{m+1}(t, \hat{\pi}) - [N_K(t, \hat{\pi})]^-]^-, \quad K = 1, 2, ..., m, \tag{A.5}$$

(iv) *If* $c^{m+1} \leq \min\{c^K, K = 1, 2, ..., m\}$, *then*

$$B^{m+1}(t, \hat{\pi}) = \max\{[N_K(t, \hat{\pi})]^-, K = 1, 2, ..., m\}, \tag{A.6}$$

$$B^K(t, \hat{\pi}) = 0, \quad K = 1, 2, ..., m. \tag{A.7}$$

Here $\mathcal{J}$ *denotes the set of all subset of products such that* $\sum_{K \in \mathcal{J}} c^K > c^{m+1}$.

Proposition 2. (Sample-path Lower Bound for the General W-system) *If there exists an admissible policy* $\widehat{\pi}$ *such that the conditional sample-path lower bound given by the linear program* (2.11)-(2.13) *can be achieved for the general W-system, it has the following sample-path properties.*

$$B^K(t, \hat{\pi}) = [N_K(t, \hat{\pi})]^- \quad K = 1, 2, ...m - 2, \tag{A.8}$$

$$B^{m-1}(t, \hat{\pi}) = \left[[N_m(t, \hat{\pi})]^- - \sum_{K=1}^{m-2} [N_K(t, \hat{\pi})]^- \right]^+ \vee [N_{m-1}(t, \hat{\pi})]^-. \tag{A.9}$$

A.3 Linear Programming Formulation for the M-system and the W-system

Here we present the linear programming formulation for the M-system under RBS policy with $c^{12} \leq c^2 \wedge c^1$ as an example, the M-system with other cost parameters and the W-system could be analyzed analogously.

Proof of (2.37). Replacing the expected value by sample average, one has

$$\begin{aligned}\min_{\boldsymbol{r}\in\mathbb{R}^5_+} \quad & \Bigg\{\frac{1}{Q}\sum_{\omega=1}^{Q}[h_1(r_1^1+\beta r_{1l}^{12}+(1-\beta)(r_{1u}^{12}-\delta^{12}(\omega))-d^1(\omega)-d^{12}(\omega))\\ & +h_2(r_2^2+r_2^{12}-\delta^2(\omega)-\delta^{12}(\omega)-d^2(\omega)-d^{12}(\omega))\\ & +c^{12}((r_2^2+r_2^{12}-\delta^2(\omega)-\delta^{12}(\omega)-d^2(\omega)-d^{12}(\omega))^-\vee[r_1^1+r_{1l}^{12}\wedge\\ & (r_{1u}^{12}-\delta^{12}(\omega))-d^1(\omega)-d^{12}(\omega)]^-)]\Bigg\}\end{aligned}$$

We make the following transformations, let

$$\begin{aligned}z_1(\omega)&=(r_1^1+r_{1l}^{12}\wedge(r_{1u}^{12}-\delta^{12}(\omega))-d^1(\omega)-d^{12}(\omega))^-,\\ z_2(\omega)&=(r_2^2+r_2^{12}-\delta^2(\omega)-\delta^{12}(\omega)-d^2(\omega)-d^{12}(\omega))^-,\\ z^{12}(\omega)&=z_1(\omega)\vee z_2(\omega).\end{aligned}$$

Ignoring the constant terms, the above minimization problem is equivalent to

$$\begin{aligned}\boldsymbol{P}_{\mathrm{RBS}}:\quad & \min_{\boldsymbol{r},\,\boldsymbol{z}}\ \tfrac{1}{Q}\textstyle\sum_{\omega=1}^{Q}\Big[h_1[r_1^1+\beta r_{1l}^{12}+(1-\beta)(r_{1u}^{12}-\delta^{12}(\omega))]\\ & +h_2(r_2^2+r_2^{12})+c^{12}z^{12}(\omega)\Big]\\ \text{s.t.}\quad & z_1(\omega)\ge d^1(\omega)+d^{12}(\omega)-r_1^1-r_{1l}^{12},\\ & z_1(\omega)\ge d^1(\omega)+d^{12}(\omega)-r_1^1-(r_{1u}^{12}-\delta^{12}(\omega)),\\ & z_2(\omega)\ge\delta^2(\omega)+\delta^{12}(\omega)+d^2(\omega)+d^{12}(\omega)-r_2^2-r_2^{12},\\ & z^{12}(\omega)\ge z_1(\omega),\\ & z^{12}(\omega)\ge z_2(\omega),\\ & \boldsymbol{r}\in\mathbb{R}^5_+,\\ & \boldsymbol{z}\in\mathbb{R}^{5Q}_+,\\ & \omega=1,2,\ldots,Q.\end{aligned}$$

Thus, we have proved (2.37). □

A.4 Difficulty in Extending the CBS Policy to the General System

We have characterized the asymptotically optimal CBS policy for the M-system. However, it is difficult to apply the same approach on general systems. To see this, we consider two specially structured systems: the general M-system and the general W-system, introduced in Section 3.

We first solve the linear program (2.11)-(2.13). The resulting sample-path properties are summarized in Appendix A.2. (2.11)-(2.13) are not difficult to solve for the general M- and W-system due to the special structure of the system. The challenge is, when we move to minimize the expected sample-path lower bound, we need to solve a multistage stochastic program similar to (2.28). This is to find the closed-form expressions for the minimizers like $\eta_1(\cdot)$ and η_2 in the M-system given by (2.29)-(2.33), that are used to characterize the asymptotically optimal policy. Moreover, without a closed-form expression for the minimizers, it would be almost impossible to verify the policy characterized by the minimizers is well coordinated, like what we do for the M-system by showing the derivative to be positive and less than one. We use the W-system with $c^{13} = c^{23}$ and $L_3 > L_2 > L_1$ as an example to intuitively illustrate why it is difficult, where component-1 and component-2 do not share any common product. Consider the inventory position for component-1 at time t: it needs to coordinate with the available inventory of component-2 and -3 at time $t+L_1$. Suppose there is an extreme case where there is no demand of product-13 during $(t-(L_3-L_1),t]$, while the demand for product-23 is extremely large during $(t-(L_2-L_1),t]$ and there will be no available inventory of the common component at time $t+L_1$. Then it is plausi-

ble to reduce the inventory position of component-1 to 0 as there is no component-3 available at time $t + L_1$ to assemble product-13. This is not always possible, so we cannot construct the CBS policy in this scenario. For the same reason, Lu et al. (2015) also mention that they cannot characterize the CBS policy in the W-system when the common component has the longest or intermediate leadtime.

A.5 Configuration of Numerical Examples

A.5.1 Detailed Numerical Result in Small System

The complete numerical setup and computation results for the M-system are illustrated in Table A.1, for the W-system are illustrated in Table A.2. The unit for holding cost and backorder cost is \$/day, for demand rate is unit/day, for leadtime is day. We fix $h_1 = 0.48, h_2 = 0.34, L_1 = 4, L_2 = 10$ in the M-system, $h_1 = 0.34, h_2 = 0.48, h_3 = 0.68, b^{13} = 1.5$ in the W-system.

A.5.2 Examples in the General System

The *PC assembly system* example is adopted from Cheng et al. (2002). We leave out 2 obsolete components of CD-ROM in the original example. The BOM of this system is illustrated in Table A.3. The yearly holding costs of the components are assumed to be 25% of the retailing price.

The *maintenance organization problem* is adopted from van Jaarsveld and Scheller-Wolf (2015). The spare parts used probability is displayed in Table A.4. A dash (-) means that the repair type never use that particular spare part.

Table A.1: Heuristic Performance in the M-system (with $h_1 = 0.48, h_2 = 0.34, L_1 = 4, L_2 = 10$)

b^1	b^2	b^{12}	λ^1	λ^2	λ^{12}	NV	CM	RBS
1.5	1.6	3.2	3	3	24	1.54	5.56	1.00
2.4	2.2	3.2	3	3	24	-0.48	2.25	-0.83
3.8	1.3	2.2	3	3	24	0.00	2.62	-0.32
1	3.8	1.2	3	3	24	0.29	8.89	-2.78
2.2	2.4	1	24	3	3	-0.05	13.13	0.29
1.5	1.6	3.2	24	3	3	-0.91	1.63	-0.09
2.4	2.2	3.2	24	3	3	-0.15	2.37	-0.59
3.8	1.3	2.2	24	3	3	0.00	9.38	1.13
1	3.8	1.2	24	3	3	0.88	10.36	3.24
2.2	2.4	1	3	24	3	2.45	28.71	-0.06
1.5	1.6	3.2	3	24	3	0.16	2.59	-0.93
2.4	2.2	3.2	3	24	3	-0.86	1.44	-1.21
3.8	1.3	2.2	3	24	3	0.29	2.61	0.76
1	3.8	1.2	3	24	3	3.16	36.11	-0.21
2.2	2.4	1	3	3	24	0.39	9.29	-4.90
3	3.2	6.4	3	3	24	0.92	4.25	0.92
4.8	4.4	6.4	3	3	24	-0.14	2.73	-0.19
7.6	2.6	4.4	3	3	24	-0.42	2.32	1.02
2	7.6	2.4	3	3	24	0.00	6.94	-0.54
4.4	4.8	2	24	3	3	0.08	13.36	0.71
3	3.2	6.4	24	3	3	0.18	2.70	0.18
4.8	4.4	6.4	24	3	3	0.00	2.54	0.13
7.6	2.6	4.4	24	3	3	-1.15	8.65	-0.19
2	7.6	2.4	24	3	3	0.00	10.41	0.50
4.4	4.8	2	3	24	3	2.16	26.36	0.06
3	3.2	6.4	3	24	3	0.59	2.17	0.59
4.8	4.4	6.4	3	24	3	0.00	2.47	0.55
7.6	2.6	4.4	3	24	3	0.11	2.46	0.32
2	7.6	2.4	3	24	3	3.12	33.47	-1.00
4.4	4.8	2	3	3	24	-0.11	7.24	-2.04

Table A.2: Algorithm Performance in the W-system (with $h_1 = 0.34, h_2 = 0.48, h_3 = 0.68, b^{13} = 1.5$)

b^{23}	L_1	L_2	L_3	λ^{13}	λ^{23}	NV	CM	RBS
2	4	8	8	15	5	0.53	16.57	-3.64
2	8	4	4	15	5	0.20	10.95	-1.95
4	8	4	12	15	5	0.22	15.90	-1.75
4	4	8	8	15	5	0.27	11.03	-1.24
4	8	4	4	15	5	0.44	4.58	-1.16
2	8	4	12	5	15	0.52	5.21	-1.28
2	4	8	8	5	15	0.65	4.96	-0.70
2	8	4	4	5	15	0.49	5.85	-0.56
4	8	4	12	5	15	0.34	6.60	0.72
4	4	8	8	5	15	0.80	8.65	-1.34
4	8	4	4	5	15	0.55	6.48	-2.22
2	8	4	12	15	5	1.00	6.36	0.41

Table A.3: The BOM of A Desktop Computers Assembly System

		Products						Retailing Price ($)	L
Components		1	2	3	4	5	6		
1	Shell	1	1	1	1	1	1	55	5
2	Shell (Common Parts 1)	1	1	1	1	1	1	114	8
3	Shell (Common Parts 2)	1	1	1	1	1	1	114	8
4	Processor 1	1	0	0	0	0	0	339	12
5	Processor 2	0	1	0	0	0	0	409	12
6	Processor 3	0	0	1	1	1	0	699	12
7	Processor 4	0	0	0	0	0	1	999	12
8	Memory	1	1	1	1	1	1	120	15
9	Hard drive 1	1	1	0	0	0	1	129	18
10	Hard drive 2	0	0	1	1	1	0	229	18
11	Hard drive (Common Parts)	1	1	1	1	1	1	108	8
12	Software Pre-load 1	1	1	1	0	1	0	114	4
13	Software Pre-load 2	0	0	0	1	0	0	114	4
14	Video Graphics Card	1	1	1	1	1	0	699	6
15	Ethernet Card	0	0	1	0	0	0	30	10

Table A.4: Spare Parts Used Probability in A Maintenance Organization

h	L	p_a	p_b	p_c	h	L	p_a	p_b	p_c	h	L	p_a	p_b	p_c
341	55	2.3%	-	-	56	21	4.7 %	5.7%	1.8%	12	5	2.3%	-	-
270	55	2.3%	2.9%	0%	52	7	7%	8.6%	6.4%	11	6	-	-	0.9%
270	55	-	-	8.3%	50	19	-	-	9.2%	11	5	-	2.9%	4.6%
249	41	4.7%	2.9%	4.6%	50	21	2.3%	2.9%	0.9%	10	5	2.3%	-	0.9%
240	41	2.3%	5.7%	2.8%	50	6	-	-	33%	10	6	4.7%	-	-
213	55	16.3%	5.7%	-	50	13	-	-	3.7%	9	5	2.3%	-	3.7%
175	34	2.3%	2.9%	0.9%	50	17	2.3%	-	-	8	5	-	-	1.8%
162	55	-	5.7%	-	50	19	18.6%	31.4%	20.22%	8	6	-	2.9%	-
156	45	7%	-	4.6%	46	21	-	2.9%	-	7	6	-	2.9%	-
156	31	-	-	0.9%	46	17	4.7%	-	4.6%	6	5	2.3%	2.9%	2.8%
128	28	-	-	0.9%	44	12	-	-	0.9%	6	7	-	-	0.9%
123	16	2.3%	2.9%	1.8%	43	12	4.7%	-	0.9%	6	5	-	2.9%	-
123	13	4.7%	-	2.8%	41	50	2.3%	8.6%	-	5	5	-	-	2.8%
105	37	-	-	1.8%	41	23	2.3%	8.6%	-	5	6	2.3%	2.9%	-
105	37	-	-	14.7%	41	3	4.7%	2.9%	5.5%	4	10	23.3%	2.9%	0.9%
105	17	-	-	0.9%	39	13	-	-	0.9%	4	6	4.7%	-	1.8%
101	3	7%	2.9%	-	39	12	-	2.9%	0.9%	4	5	-	-	2.8%
97	28	4.7%	5.7%	3.7%	39	7	11.6%	5.7%	10.1%	4	5	-	5.7%	4.6%
94	28	2.3%	-	5.5%	39	19	11.6%	5.7%	10.1%	4	10	4.7%	-	0.9%
90	26	2.3%	-	0.9%	39	12	20.9%	25.7%	18.3%	4	5	11.6%	17.1%	-
86	50	-	2.9%	-	39	3	-	-	1.8%	4	6	44.2%	45.7%	33%
83	23	-	-	1.8%	39	28	32.6%	42.9%	29.4%	3	6	2.3%	2.9%	-
83	23	-	-	3.7%	38	12	4.7%	8.6%	6.4%	3	6	-	2.9%	-
80	37	4.7%	11.4%	7.3%	36	13	-	2.9%	-	3	5	4.7%	8.6%	2.8%
77	12	-	-	0.9%	35	19	-	2.9%	-	3	17	4.7%	-	1.8%
77	50	-	2.9%	5.5%	35	6	2.3%	-	-	3	5	2.3%	-	-
77	26	-	-	0.9%	35	17	14%	-	-	3	6	-	2.9%	0.9%
74	16	-	-	1.8%	32	6	-	42.9%	-	3	6	4.7%	-	%0
74	7	-	2.9%	0.9%	32	3	-	-	1.8%	3	7	-	2.9%	0.9%
71	13	2.3%	8.6%	-	32	21	51.2%	62.9%	43.1%	3	7	2.3%	2.9%	0.9%
68	3	48.8%	57.1%	53.2%	31	19	11.6%	17.1%	8.3%	3	7	4.7%	-	0.9%
66	6	2.3%	-	-	31	12	11.6%	17.1%	8.3%	2	5	4.7%	-	1.8%
66	13	-	2.9%	-	31	6	14%	-	-	2	5	-	-	2.8%
63	26	48.8%	57.1%	48.6%	31	23	2.3%	-	0.9%	1	6	7%	8.6%	-
61	23	-	-	0.9%	31	16	-	2.9%	-	1	6	14%	22.9%	2.8%
58	12	2.3%	-	-	30	11	4.7%	-	-	1	5	7%	2.9%	3.7%
58	21	39.5%	-	-	27	10	-	-	0.9%					

Appendix B

Appendix for Chapter 3

B.1 Appendix: Proofs of the Lemmas and Theorems

B.1.1 Proof of Corollary 1

We want show that there exists a constant c such that the following inequality holds:

$$\left(\hat{C}(\mathcal{X}+(i',j'))-\hat{C}(\mathcal{X})\right)-\left(\hat{C}(\mathcal{Y}+(i',j'))-\hat{C}(\mathcal{Y})\right)\leq c \tag{B.1}$$

for any two network configurations $\mathcal{X},\mathcal{Y}$ such that $\mathcal{X}\subset\mathcal{Y}\subset\mathcal{G}$ and any edge $(i',j')\notin\mathcal{Y}$. We introduce x_{ij}, a binary indicator, where $x_{ij}=1$ if and only if $(i,j)\in\mathcal{X}$.

First, we expand $\hat{C}(\mathcal{X}+(i',j')$, the first term on LHS of (B.1).

$$\begin{aligned}\hat{C}(\mathcal{X}) &= \sum_{(i,j)\in\mathcal{X}}\kappa_{ij}+\sum_{(i,j)\in G}\hat{x}_{ij}(m_i(\mathcal{X}),n_j(\mathcal{X}),x_{ij})\\ \hat{C}(\mathcal{X}+(i',j')) &= \sum_{(i,j)\in\mathcal{X}+(i',j')}\kappa_{ij}+\sum_{(i,j)\in G}\hat{x}_{ij}(m_i(\mathcal{X}+(i',j')),n_j(\mathcal{X}+(i',j')),x_{ij})\end{aligned}$$

The effects of adding edge (i',j') into network configuration $\mathcal{X}$ on each edge of $\mathcal{G}$ is different. For edge (i',j'), both the in-degree at destination j' and the out-degree at origin i' increases from 0 to 1. For edges in $(i,j)\in G\backslash(i',j')$, we can further divide them into 3 categories: (a) edges (i,j'), $\forall i\in\mathcal{I}$: the in-degree at destination j' increases by 1; (b) edges (i',j), $\forall j\in\mathcal{J}$: the out-degree at origin i' increases by 1;

(c) edges (i,j), $\forall i \in \mathcal{I}\backslash i', \forall j \in \mathcal{J}\backslash j'$: neither the in-degree at destination j nor the out-degree at origin i changes.

$\hat{C}(\mathcal{X}+(i',j')) - \hat{C}(\mathcal{X})$ can then expressed as

$$\begin{aligned}
&\kappa_{i'j'} + \Big(\hat{x}_{i'j'}(m_i(\mathcal{X})+1, n_j(\mathcal{X})+1, 1) - \hat{x}_{i'j'}(m_i(\mathcal{X}), n_j(\mathcal{X}), 0)\Big) \\
&+\Big(\sum_{i\neq i'} \hat{x}_{ij'}(m_i(\mathcal{X}), n_{j'}(\mathcal{X})+1, x_{ij'}) - \sum_{i\neq i'} \hat{x}_{ij'}(m_i(\mathcal{X}), n_{j'}(\mathcal{X}), x_{ij'})\Big) \\
&+\Big(\sum_{j\neq j'} \hat{x}_{i'j}(m_{i'}(\mathcal{X})+1, n_j(\mathcal{X}), x_{i'j}) - \sum_{j\neq j'} \hat{x}_{i'j}(m_{i'}(\mathcal{X}), n_j(\mathcal{X}), x_{i'j})\Big). \quad \text{(B.2)}
\end{aligned}$$

It is upper bounded by:

$$\kappa_{i'j'} + \Delta_{12}^{max}\hat{x} + (I-1)\Delta_2^{max}\hat{x} + (J-1)\Delta_1^{max}\hat{x} \quad \text{(B.3)}$$

because

$$\begin{aligned}
&(i) \quad \hat{x}_{i'j'}(m_{i'}(\mathcal{X})+1, n_{j'}(\mathcal{X})+1, 1) - \hat{x}_{i'j'}(m_{i'}(\mathcal{X}), n_{j'}(\mathcal{X}), 0) \leq \Delta_{12}^{max}\hat{x} \\
&(ii) \quad \sum_{i\neq i'} \hat{x}_{ij'}(m_i(\mathcal{X}), n_{j'}(\mathcal{X})+1, x_{ij'}) - \sum_{i\neq i'} \hat{x}_{ij'}(m_i(\mathcal{X}), n_{j'}(\mathcal{X}), x_{ij'}) \leq (I-1)\Delta_2^{max}\hat{x} \\
&(iii) \quad \sum_{j\neq j'} \hat{x}_{i'j}(m_{i'}(\mathcal{X})+1, n_j(\mathcal{X}), x_{i'j}) - \sum_{j\neq j'} \hat{x}_{i'j}(m_{i'}(\mathcal{X}), n_j(\mathcal{X}), x_{i'j}) \leq (J-1)\Delta_1^{max}\hat{x}.
\end{aligned}$$

Similarly, we can find the lower bound for $\hat{C}(\mathcal{Y}+(i',j')) - \hat{C}(\mathcal{Y})$, the second term on LHS of (B.1), is

$$\kappa_{i'j'} + \Delta_{12}^{min}\hat{x} + (I-1)\Delta_2^{min}\hat{x} + (J-1)\Delta_1^{min}\hat{x} \quad \text{(B.4)}$$

In conclusion, we can have

$$c = \Delta_{12}^{max}\hat{x} - \Delta_{12}^{min}\hat{x} + (I-1)(\Delta_{2}^{max}\hat{x} - \Delta_{2}^{min}\hat{x}) + (J-1)(\Delta_{1}^{max}\hat{x} - \Delta_{1}^{min}\hat{x}). \quad \text{(B.5)}$$

B.1.2 Proof of Corollary 2

There is only one destination ($J = \{1\}$) in this setting. Recall that the expected shortfall is $\boldsymbol{u}(\mathcal{Y}, \boldsymbol{s}, \boldsymbol{d}) = \{u_{i1p}(\mathcal{Y}, \boldsymbol{s}, \boldsymbol{d}) : i \in \mathcal{I}, p \in \mathcal{P}\}$. The expected network cost given network configuration $\mathcal{Y}$ is

$$C(\mathcal{Y}, \boldsymbol{s}, \boldsymbol{d}) = \sum_{(i,1)\in\mathcal{Y}} \kappa_{i1} + \sum_{i\in\mathcal{I}, p\in\mathcal{P}} c_{i1} u_{i1p}(\mathcal{Y}, \boldsymbol{s}, \boldsymbol{d}) \quad \text{(B.6)}$$

Since the fulfillment decisions are made after the demands are realized, for each realized demand instance $\boldsymbol{d}'$, we use v'_{i1p} and u'_{i1p} to denote the optimal network flow and the optimal network shortfall of product p from origin i to destination 1. We can find the network flow and shortfall given a network configuration $\mathcal{Y}$ by solving the following network flow problem:

$$\begin{aligned} C^{\mathcal{Y}}(\boldsymbol{s}, \boldsymbol{d}') = \min_{\boldsymbol{u}', \boldsymbol{v}'} & \sum_{i\in\mathcal{I}, p\in\mathcal{P}} c_{i1} u'_{i1p} \quad \text{(B.7)} \\ s.t. \quad & \sum_{i\in\mathcal{I}} (u'_{i1p} + v'_{i1p}) = d'_{1p}, \ \forall p \in \mathcal{P}, \\ & u'_{i1p} + v'_{i1p} \le s_{ip}, \ \forall i \in \mathcal{I}, \forall p \in \mathcal{P}, \\ & v'_{i1p} = 0, \ \forall (i,1) \notin \mathcal{Y}, \forall p \in \mathcal{P}, \\ & u'_{i1p} \ge 0, v'_{i1p} \ge 0 \ \forall i \in \mathcal{I}, \forall p \in \mathcal{P}. \end{aligned}$$

Next, note the above problem can be decomposed into P subproblems where each subproblem corresponds to the network flow problem for product p. Second, each

subproblem can be formulated as a network flow problem, which can be equivalent formulated as a maximum weight circulation problem by consider maximizing the negative of the objective $\sum_{i\in\mathcal{I}} c_{i1}u'_{i1p}$.

Therefore, applying the result of Gale and Politof (1981), the flow variables u'_{i1p} on edges $\{(i,1)\}_{i\in\mathcal{I}}$ are *in parallel* with each other, i.e., heads or tails of the edges lie on a common node, implying that the edges are submodular with each other in each of the maximum weight circulation problems correspond to product p. This implies that the network flow problem $C^{\mathcal{Y}}(\boldsymbol{s},\boldsymbol{d}')$ is supermodular in $\mathcal{Y}$. Finally, note that $C(\mathcal{Y},\boldsymbol{s},\boldsymbol{d})$ is simply $\sum_{(i,1)\in\mathcal{Y}}\kappa_{i1}$ plus the expected value of $C^{\mathcal{Y}}(\boldsymbol{s},\boldsymbol{d}')$ over all instances of $\boldsymbol{d}'$. Thus, $C(\mathcal{Y},\boldsymbol{s},\boldsymbol{d})$ is also supermodular, or c-supermodular with $c=0$. □

B.1.3 Proof of Theorem 1

We need to prove Lemma 1 first before proving Theorem 1.

Lemma 1. *(adapted from Lemma 2.2 from Buchbinder et al. (2014)). Let f be a non-negative non-monotone c-submodular function. Denote by $\mathcal{G}(p)$ a random subset of $\mathcal{G}$, where each element appears with probability at most p, then we have*

$$\mathsf{E}[f(\mathcal{G}(p))] \geq (1-p)f(\emptyset) - Gc. \tag{B.8}$$

Proof. This proof is similar to Lemma 2.2 from Buchbinder et al. (2014) except that we use the property of c-submodularity instead of submodularity in the first inequality of (B.9).

Sort the elements of $\mathcal{G}$ in a non-decreasing order of probability to be in $\mathcal{G}(p)$: $a_1, a_2, \ldots, a_G$. In other words, for every pair $1 \leq k_1 \leq k_2 \leq G, Pr[a_{k_1} \in \mathcal{G}(p)] \geq Pr[a_{k_2} \in \mathcal{G}(p)]$. For $k = 1, \ldots, G$, we introduce an indicator function I_{a_k} where $I_{a_k} = 1$ if the event $a_k \in \mathcal{G}(p)$ happens and $I_{a_k} = 0$ if not, define probability p_k of

event $a_k \in \mathcal{G}(p)$ where $p_k = \mathsf{E}[I_{a_k}]$ and define set $\mathcal{G}_k = \{a_1, \ldots, a_k\}$. We have

$$\begin{aligned}
\mathsf{E}[f(\mathcal{G}(p))] &= E\left[f(\emptyset) + \sum_{k=1}^{G} I_{a_k} f_{a_k}(\mathcal{G}_{k-1} \cap \mathcal{G}(p))\right] \\
&\geq E\left[f(\emptyset) + \sum_{k=1}^{G} I_{a_k} f_{a_k}(\mathcal{G}_{k-1})\right] - Gc \\
&= f(\emptyset) + \sum_{k=1}^{G} \mathsf{E}[I_{a_k}] f_{a_k}(\mathcal{G}_{k-1}) - Gc \\
&= f(\emptyset) + \sum_{k=1}^{G} p_k f_{a_k}(\mathcal{G}_{k-1}) - Gc \\
&= (1-p_1) f(\emptyset) + \sum_{k=1}^{G-1} [p_k - p_{k+1}] f(\mathcal{G}_k) + p_G f(\mathcal{G}) - Gc \\
&\geq (1-p) f(\emptyset) - Gc. \qquad \text{(B.9)}
\end{aligned}$$

The first inequality follows from c-submodularity and $\mathcal{G}_{k-1} \cap \mathcal{G}(p)$ is a subset of $\mathcal{G}_{k-1}$. The second follows from the way elements ordered, $p_1 \geq p_2 \geq \ldots \geq p_{k-1} \geq p_k \geq \ldots \geq p_G$ and f is non-negative. $\square$

Proof of Theorem 1*:* This proof is similar to Theorem 1.3 from Buchbinder et al. (2014) except that we need to consider c-submodularity instead of submodularity in the last inequality of both (B.10) and (B.12).

Consider a set M'_k containing the elements of $S^* \backslash S_{k-1}$ plus enough dummy elements to make the size of M'_k exactly K. In every iteration, implicitly conditioned

on S_{k-1}, we have

$$\begin{aligned}
\mathsf{E}[f_{\phi_k}(S_{k-1})] &= \frac{1}{K}\sum_{\phi\in M_k} f_\phi(S_{k-1}) \\
&\geq \frac{1}{K}\sum_{\phi\in M'_k} f_\phi(S_{k-1}) \\
&= \frac{1}{K}\sum_{\phi\in S^*\backslash S_{k-1}} f_\phi(S_{k-1}) \\
&\geq \frac{1}{K}(f(S^*\cup S_{k-1}) - f(S_{k-1}) - Kc) \qquad \text{(B.10)}
\end{aligned}$$

The first inequality follows from definition of M_k and M'_k. The second inequality is from c-submodularity of f. Define a group of elements $b_1,\ldots,b_{K'}$ such that $\{b_1,b_2,\ldots,b_{K'}\} = S^*\backslash S_k$ for some $K' \leq K$, and set $N_0 = \{\emptyset\}, N_1 = \{b_1\}, N_2 = \{b_1,b_2\},\ldots,N_{K'} = \{b_1,\ldots,b_{K'}\}$. By c-submodularity we have

$$\begin{aligned}
f(S^*\cup S_k) - f(S_k) &= f_{b_1}(S_k\cup N_0) + f_{b_2}(S_k\cup N_1) + \ldots + f_{b_{K'}}(S'_k\cup N_{K'-1}) \\
&\leq f_{b_1}(S_k) + f_{b_2}(S_k) + \ldots + f_{b_{K'}}(S_k) + Kc \\
&= \sum_{\phi\in S^*\backslash S_k} f_\phi(S_k) + Kc \qquad \text{(B.11)}
\end{aligned}$$

Taking expectation of the last inequality in (B.10) over all S_{k-1}, we have

$$\begin{aligned}
\mathsf{E}[f_{\phi_k}(S_{k-1})] &\geq \frac{1}{K}(\mathsf{E}[f(S^*\cup S_{k-1})] - \mathsf{E}[f(S_{k-1})] - Kc) \\
&\geq \frac{1}{K}\bigg((1-\frac{1}{K})^{k-1}f(S^*) - \mathsf{E}[f(S_{k-1})] - Kc - Gc\bigg). \qquad \text{(B.12)}
\end{aligned}$$

The second inequality is because Lemma 1: We observe that function $g(S) = f(S\cup S^*)$ is still c-submodular, and each element of $\mathcal{G}\backslash S_{k-1}$ stays outside of S_k with probability at least $1-1/K$, implying each element belongs to S_k with probability

at most $1-(1-1/K)^k$ in iteration k.

Then we prove by induction that $\mathsf{E}[f(S_k)] \geq k/K(1-1/K)^{k-1}f(S^*) - Kc - Gc$. For $k=0$, it is true. Assume it holds for every $k' < k$, let us prove it for $k > 0$.

$$\begin{aligned}
\mathsf{E}[f(S_k)] &= \mathsf{E}[f(S_{k-1})] + \mathsf{E}[f_{\phi_k}(S_{k-1})] \\
&\geq \mathsf{E}[f(S_{k-1})] + \frac{1}{K}\left((1-\frac{1}{K})^{k-1}f(S^*) - \mathsf{E}[f(S_{k-1})] - Kc - Gc\right) \\
&= (1-\frac{1}{K})\mathsf{E}[f(S_{k-1})] + \frac{1}{K}(1-\frac{1}{K})^{k-1}f(S^*) - c - \frac{G}{K}c \\
&\geq (1-\frac{1}{K})\frac{k-1}{K}(1-\frac{1}{K})^{k-2}f(S^*) + \frac{1}{K}(1-\frac{1}{K})^{k-1}f(S^*) - Kc - Gc \\
&= \frac{k}{K}(1-\frac{1}{K})^{k-1}f(S^*) - Kc - Gc. \qquad \text{(B.13)}
\end{aligned}$$

In conclusion,

$$\begin{aligned}
\mathsf{E}[f(S_K)] &\geq \frac{K}{K}(1-\frac{1}{K})^{K-1}f(S^*) - (G+K)c \\
&\geq \frac{1}{e}f(S^*) - (G+K)c. \qquad \text{(B.14)}
\end{aligned}$$

There are K iterations, and every iteration requires G function evaluations, so the complexity of Random Greedy is $O(GK)$. $\square$

B.1.4 Proof of Theorem 2.

Before proceeding to the formal proof, we first give an outline. We consider $X_r = S'_K$ as S'_K has the worst performance guarantee among all possible outputs of Hybrid. First, we will show through Lemma 2 that the output of the Double Greedy step (Step 1), X_G, has the following performance guarantee:

$$\mathsf{E}[f(X_G)] \geq \frac{1}{2}f(S^*) - \frac{1}{2}Gc. \qquad \text{(B.15)}$$

We then show through Lemma 3 that function f is c-monotone on the output of the set reduction process (Step 2), X'. Then we consider a reduced problem of Problem $\boldsymbol{P^c}$ in (3.10):

$$\underline{\boldsymbol{P^c}}: \max_{S \subseteq X'} f(S) \qquad s.t.\ |S| \leq K, \tag{B.16}$$

and we use S^{**} to denote the optimal solution to this reduced problem. We can show through Lemma 5 that the output of the Stochastic Greedy step (Step 3), S'_K, has the following performance guarantee:

$$\mathsf{E}[f(S'_K)] \geq \left(1 - \frac{1}{e} - \epsilon\right) f(S^{**}) - (2 - \epsilon)Kc. \tag{B.17}$$

We link S^{**} to X' through Lemma 6:

$$f(S^{**}) \geq \frac{K}{G+K} f(X') - \frac{KG}{G+K} c. \tag{B.18}$$

Also, step 2 ensures $f(X') \geq f(X_G)$, and by Lemma 2, $\mathsf{E}[f(X_G)] > 1/2 f(S^*) - (1/2)Gc$, then we combine the above results and have:

$$\mathsf{E}[f(S'_K)] \geq \frac{K}{2(G+K)}\left(1 - \frac{1}{e} - \epsilon\right) f(S^*) - \frac{3GK}{2(G+K)}\left(1 - \frac{1}{e} - \epsilon\right) c - (2 - \epsilon)Kc.$$

Lemma 2. *(adapted from Theorem 1.2 from Buchbinder et al. (2015)) Double Greedy in step 1 of Hybrid returns a set X_G, which has the following performance guarantee:*

$$\mathsf{E}[f(X_G)] \geq \frac{1}{2} f(S^*) - \frac{1}{2} Gc. \tag{B.19}$$

Proof. This proof is similar to Theorem 1.2 from Buchbinder et al. (2015) except

that we need to consider c-submodularity instead of submodularity in showing the inequality (B.20) and consider an extra scenario where $a_k < 0, b_k < 0$ (case (iv)).

Define a hybrid solution $H_k = (S^* \cup X_k) \cap Y_k$ and always note that $\phi_k \in Y_{k-1}$, $\phi_k \notin X_{k-1}$ and $X_k \subseteq Y_k$. We want to show the below inequality always holds:

$$\mathsf{E}[f(H_{k-1}) - f(H_k)] \leq \frac{1}{2}\mathsf{E}[f(X_k) - f(X_{k-1}) + f(Y_k) - f(Y_{k-1})] + c. \tag{B.20}$$

Because hybrid solution $H_0 = S^*$ and $H_G = X_G = Y_G$, finding the inequality between the decreases in H_k and increases in X_k and Y_k in each iteration helps derive the performance guarantee:

Summing up (B.20) for every $1 \leq k \leq G$, we have

$$\begin{aligned} \mathsf{E}[f(H_0) - f(H_G)] &\leq \frac{1}{2}\mathsf{E}[f(X_G) - f(X_0) + f(Y_G) - f(Y_0)] + Gc \\ &\leq \frac{1}{2}\mathsf{E}[f(X_G) + f(Y_G)] + Gc. \end{aligned} \tag{B.21}$$

As $H_0 = S^*$ and $H_G = X_G = Y_G$, we have

$$\mathsf{E}[f(X_G)] \geq \frac{1}{2}f(S^*) - \frac{1}{2}Gc \tag{B.22}$$

To prove the inequality (B.20) always holds, we check four different cases. The proof of Theorem 1.2 from Buchbinder et al. (2015) only considers the first three cases because its function is submodular instead of c-submodular. Notice that it suffices to prove inequality (B.20) conditioned on $\{\phi_1, \ldots, \phi_{k-1}\}$. The rest of the proof implicitly assumes everything is conditioned on this event.

(i) $a_k \geq 0, b_k \leq 0$: In this case, $Y_k = Y_{k-1}$, $X_k = X_{k-1} + \phi_k$.

We first analyze LHS of inequality (B.20):

$$\mathsf{E}[f(H_{k-1}) - f(H_k)] = \mathsf{E}[f(H_{k-1}) - f(H_{k-1} + \phi_k)]. \tag{B.23}$$

If $\phi_k \in H_{k-1}$, then $H_{k-1} = H_{k-1} + \phi_k$ and $\mathsf{E}[f(H_{k-1}) - f(H_k)]$ is 0; otherwise, by c-submodularity and $H_{k-1} = (S^* \cup X_{k-1}) \cap Y_{k-1}$ is subset of $Y_{k-1} - \phi_k$, we have

$$\mathsf{E}[f(H_{k-1}) - f(H_{k-1} + \phi_k)] \leq f(Y_{k-1} - \phi_k) - f(Y_{k-1}) + c = b_k + c \leq c. \tag{B.24}$$

We then analyze RHS of inequality (B.20): $\frac{1}{2}\mathsf{E}[f(X_k) - f(X_{k-1}) + f(Y_k) - f(Y_{k-1})] + c = \frac{1}{2}a_k + c \geq c$. This implies that inequality (B.20) holds.

(ii) $a_k < 0, b_k \geq 0$: similar to (a), $Y_k = Y_{k-1} - \phi_k$, $X_k = X_{k-1}$ in this case.

We first analyze LHS of inequality (B.20):

$$\mathsf{E}[f(H_{k-1}) - f(H_k)] = \mathsf{E}[f(H_{k-1}) - f(H_{k-1} - \phi_k)]. \tag{B.25}$$

If $\phi_k \in H_{k-1}$, then by c-submodularity and X_{k-1} is a subset of $H_{k-1} - \phi_k = (S^* \cup X_{k-1}) \cap Y_{k-1} - \phi_k$,

$$\mathsf{E}[f(H_{k-1}) - f(H_{k-1} - \phi_k)] \leq \mathsf{E}[f(X_{k-1} + \phi_k) - f(X_{k-1})] = a_k + c \leq c. \tag{B.26}$$

Otherwise, $H_{k-1} = H_{k-1} - \phi_k$ and $\mathsf{E}[f(H_{k-1}) - f(H_k)]$ is 0.

We then analyze RHS of inequality (B.20) $\frac{1}{2}$: $\mathsf{E}[f(X_k) - f(X_{k-1}) + f(Y_k) - f(Y_{k-1})] + c = \frac{1}{2}b_k + c \geq c$. This implies that inequality (B.20) holds.

(iii) $a_k \geq 0, b_k > 0$: With probability $a_k/(a_k + b_k)$, $X_k = X_{k-1} + \phi_k, Y_k = Y_{k-1}$; with probability $b_k/(a_k + b_k)$, $X_k = X_{k-1}, Y_k = Y_{k-1} - \phi_k$.

We first analyze LHS of inequality (B.20):

$$\begin{aligned}\mathsf{E}[f(H_{k-1}) - f(H_k)] &= \frac{a_k}{a_k + b_k}[f(H_{k-1}) - f(H_{k-1} + \phi_k)] \\ &\quad + \frac{b_k}{a_k + b_k}[f(H_{k-1}) - f(H_{k-1} - \phi_k)] \\ &\leq \frac{a_k b_k}{a_k + b_k} + c \end{aligned} \tag{B.27}$$

The inequality holds by considering two cases. In the first case, $\phi_k \notin H_{k-1}$, then $H_{k-1} = H_{k-1} - \phi_k$ and the second term of the LHS of the inequality in (B.27) equals zero. Moreover, $H_{k-1} = (S^* \cup X_{k-1}) \cap Y_{k-1}$ is subset of $Y_{k-1} - \phi_k$ and therefore, by c-submodularity,

$$f(H_{k-1}) - f(H_{k-1} + \phi_k) \leq f(Y_{k-1} - \phi_k) - f(Y_{k-1}) + c = b_k + c. \tag{B.28}$$

In the second case, if $\phi_k \in H_{k-1}$, then $H_{k-1} = H_{k-1} + \phi_k$ and the first term of the LHS of the inequality in (B.27) equals zero. Moreover, X_{k-1} is a subset of $H_{k-1} - \phi_k = (S^* \cup X_{k-1}) \cap Y_{k-1} - \phi_k$ and therefore, by c-submodularity,

$$f(H_{k-1}) - f(H_{k-1} - \phi_k) \leq f(X_{k-1} + \phi_k) - f(X_{k-1}) + c = a_k + c. \tag{B.29}$$

We then analyze RHS of the inequality (B.20):

$$\begin{aligned} & \frac{1}{2}\mathsf{E}[f(X_k) - f(X_{k-1}) + f(Y_k) - f(Y_{k-1})] + c \\ = \; & \frac{a_k}{2(a_k + b_k)}\mathsf{E}[f(X_{k-1} + \phi_k) - f(X_{k-1}) + f(Y_k) - f(Y_{k-1})] \\ & + \frac{b_k}{2(a_k + b_k)}\mathsf{E}[f(X_k) - f(X_{k-1}) + f(Y_{k-1} - \phi_k) - f(Y_{k-1})] + c \\ = \; & \frac{1}{2}\frac{a_k^2 + b_k^2}{a_k + b_k} + c. \end{aligned} \tag{B.30}$$

Finally, the inequality (B.20) holds because $\frac{1}{2}\frac{a_k^2+b_k^2}{a_k+b_k} + c \geq \frac{a_k b_k}{a_k+b_k} + c$.

(iv) $a_k < 0, b_k < 0$: With probability $1/2$, $X_k = X_{k-1} + \phi_k, Y_k = Y_{k-1}$; with probability $1/2$, $X_k = X_{k-1}, Y_k = Y_{k-1} - \phi_k$.

We first analyze LHS of the inequality (B.20):

$$\mathsf{E}[f(H_{k-1}) - f(H_k)] \;=\; \frac{1}{2}[f(H_{k-1}) - f(H_{k-1} + \phi_k)] + \frac{1}{2}[f(H_{k-1}) - f(H_{k-1} - \phi_k)]. \tag{B.31}$$

We consider two cases. In the first case, $\phi_k \notin H_{k-1}$, then $H_{k-1} = H_{k-1} - \phi_k$ and the second term of the LHS of the inequality in (B.31) equals zero. Moreover, $H_{k-1} = (S^* \cup X_{k-1}) \cap Y_{k-1}$ is subset of $Y_{k-1} - \phi_k$ and therefore, by c-submodularity,

$$f(H_{k-1}) - f(H_{k-1} + \phi_k) \leq f(Y_{k-1} - \phi_k) - f(Y_{k-1}) + c = b_k + c. \tag{B.32}$$

In the second case, if $\phi_k \in H_{k-1}$, then $H_{k-1} = H_{k-1} + \phi_k$ and the first term of the LHS of the inequality in (B.31) equals zero. Moreover, X_{k-1} is a subset of $H_{k-1} - \phi_k = (S^* \cup X_{k-1}) \cap Y_{k-1} - \phi_k$ and therefore, by c-submodularity,

$$f(H_{k-1}) - f(H_{k-1} - \phi_k) \leq f(X_{k-1} + \phi_k) - f(X_{k-1}) + c = a_k + c. \tag{B.33}$$

Combine the result above:

$$\mathsf{E}[f(H_{k-1}) - f(H_k)] \;\leq\; \frac{1}{2}\max\{a_k + c, b_k + c\} \tag{B.34}$$

Then we analyze RHS of the inequality (B.20):

$$\begin{aligned} & \frac{1}{2}\mathsf{E}[f(X_k) - f(X_{k-1}) + f(Y_k) - f(Y_{k-1})] + c \\ = \; & \frac{1}{4}\mathsf{E}[f(X_{k-1} + \phi_k) - f(X_{k-1}) + f(Y_k) - f(Y_{k-1})] \\ & + \frac{1}{4}\mathsf{E}[f(X_k) - f(X_{k-1}) + f(Y_{k-1} - \phi_k) - f(Y_{k-1})] + c \\ = \; & \frac{1}{4}(a_k + b_k) + c. \end{aligned} \tag{B.35}$$

Also by c-submodularity and X_{k-1} is subset of $Y_{k-1} - \phi_k$, and the definition of a_k, b_k, we have

$$a_k + b_k = f(X_{k-1} + \phi_k) - f(X_{k-1}) + f(Y_{k-1} - \phi_k) - f(Y_{k-1}) \geq -c. \tag{B.36}$$

We find that RHS of the inequality (B.20) is $1/4(a_k + b_k) + c \geq c/2$, and LHS of the inequality is $1/2 \max\{a_k + c, b_k + c\} \leq c/2$. This implies that inequality (B.20) holds. □

Lemma 3. *The set reduction process in step 2 of Hybrid returns a set X' such that the objective function f is c-monotone.*

Proof. For any set $S \subset S_0$, element ϕ in $X' \backslash S$, we are going to show $f_\phi(S) \geq -c$. First, we observe that ϕ must also be an element of S_0 since $X' \subseteq S_0$. Suppose ϕ is k-th element of S_0, i.e., $\phi = \phi_k$, then we can use the property of c-submodularity to show

$$f(S + \phi_k) - f(S) + c \geq f(S_{k-1}) - f(S_{k-1} - \phi_k) \geq 0. \tag{B.37}$$

This is because $S \subset S_{k-1} - \phi_k$ and removing ϕ_k in iteration k does not improve the objective value. □

We next state a lemma, which will be used to prove Lemma 5.

Lemma 4. *(adapted from Lemma 2 from Mirzasoleiman et al. (2015)) Let* $s = (|X'|/K)\log(1/\epsilon)$. *Given a solution* S'_k, *then* $\mathsf{E}[f(S'_{k+1}) - f(S'_k)]$, *the expected gain from iteration* $k+1$ *in step 3 of Hybrid, is at least*

$$\frac{1-\epsilon}{K} \sum_{\phi \in S^{**} \setminus S'_k} (f_\phi(S'_k) + c) - c, \tag{B.38}$$

where S^{**} *is the optimal solution to Problem* $\underline{\boldsymbol{P}}^{\boldsymbol{c}}$ *in (B.16).*

Proof. This proof is similar to Lemma 2 from Mirzasoleiman et al. (2015) except that we use the property of c-monotonicity instead of monotonicity in showing the second inequality of (B.41).

First estimate the probability of $M_k \cap (S^{**} \setminus S'_k) \neq \emptyset$. As M_k is a set containing s random samples from $X' \setminus S'_k$, we have

$$Pr[M_k \cap (S^{**} \setminus S'_k) = \emptyset] = \left(1 - \frac{|S^{**} \setminus S'_k|}{|X' \setminus S'_k|}\right)^s \leq e^{-\frac{s}{|X'|}|S^{**} \setminus S'_k|}. \tag{B.39}$$

Then we have

$$\begin{aligned} Pr[M_k \cap (S^{**} \setminus S'_k) \neq \emptyset] &\geq 1 - e^{-\frac{s}{|X'|}|S^{**} \setminus S'_k|} \\ &\geq \left(1 - e^{-\frac{s}{|X'|}K}\right) \frac{|S^{**} \setminus S'_k|}{K} \\ &= (1-\epsilon)\frac{|S^{**} \setminus S'_k|}{K}, \end{aligned} \tag{B.40}$$

The second inequality is due to the concavity of function $g(x) = 1 - e^{-(sx)/|X'|}$: we have the property $g(x)/x \geq g(K)/K$ for $x \leq K$ and then substitute $x = |S^{**} \setminus S'_k| \leq K$. The last equality is because we set $s = (|X'|/K)\log(1/\epsilon)$.

As a result, we have

$$
\begin{aligned}
\mathsf{E}[f(S'_{k+1}) - f(S'_k)] \quad &\geq Pr[M_k \cap (S^{**} \backslash S'_k) \neq \emptyset] \times \frac{1}{|S^{**} \backslash S'_k|} \sum_{\phi \in S^{**} \backslash S'_k} f_\phi(S'_k) \\
&+ Pr[M_k \cap (S^{**} \backslash S'_k) = \emptyset] \times \frac{1}{|X' \backslash (S^{**} \backslash S'_k)|} \sum_{\phi \in X' \backslash (S^{**} \backslash S'_k)} f_\phi(S'_k) \\
&= Pr[M_k \cap (S^{**} \backslash S'_k) \neq \emptyset] \times \frac{1}{|S^{**} \backslash S'_k|} \sum_{\phi \in S^{**} \backslash S'_k} (f_\phi(S'_k) + c) \\
&+ Pr[M_k \cap (S^{**} \backslash S'_k) = \emptyset] \times \frac{1}{|X' \backslash (S^{**} \backslash S'_k)|} \sum_{\phi \in X' \backslash (S^{**} \backslash S'_k)} \\
&(f_\phi(S'_k) + c) - c \\
&\geq Pr[M_k \cap (S^{**} \backslash S'_k) \neq \emptyset] \times \frac{1}{|S^{**} \backslash S'_k|} \sum_{\phi \in S^{**} \backslash S'_k} \\
&(f_\phi(S'_k) + c) - c \\
&\geq \frac{1 - \epsilon}{K} \sum_{\phi \in S^{**} \backslash S'_k} (f_\phi(S'_k) + c) - c. \qquad \text{(B.41)}
\end{aligned}
$$

The first inequality is because we greedily select the added element, and M_k is equally likely to contain every element of X'. Thus, a uniformly random element of $M_k \cap (S^{**} \backslash S'_k)$ is actually a uniformly random element of $S^{**} \backslash S'_k$ if $M_k \cap (S^{**} \backslash S'_k) \neq \emptyset$. Similarly, a uniformly random element of M_k is a uniformly random element of $X' \backslash (S^{**} \backslash S'_k)$ if $M_k \cap (S^{**} \backslash S'_k) = \emptyset$. Also notice f is c-monotone, which implies $f_\phi(S'_k) + c \geq 0$ for every $\phi \in X'$. Then the second inequality is because we drop a non-negative term, and the third inequality holds due to (B.40). $\square$

Lemma 5. *Stochastic Greedy in step 3 of Hybrid returns a set S'_K, which has the following performance guarantee:*

$$
\mathsf{E}[f(S'_K)] \geq \left(1 - \frac{1}{e} - \epsilon\right) f(S^{**}) - (2 - \epsilon) K c, \qquad \text{(B.42)}
$$

where S^{**} is the optimal solution to Problem $\underline{\boldsymbol{P}}^{\boldsymbol{c}}$ in (B.16).

Proof. This proof follow similar steps of the proof of Theorem 1.3 from Buchbinder et al. (2014) except that in (B.43) where all three inequalities are different. In particular, we need to consider c-monotonicity instead of monotonicity in showing the first inequality, c-submodularity instead of submodularity in showing the second inequality, elements are added using Stochastic Greedy instead of Random Greedy in showing the last inequality.

In iteration k, we define elements $v_1, \ldots, v_{K'}$ such that $\{v_1, v_2, \ldots, v_{K'}\} = S^{**} \backslash S'_k$ for some $K' \leq K$, and sets $N_0 = \{\emptyset\}, N_1 = \{v_1\}, N_2 = \{v_1, v_2\}, \ldots, N_{K'} = \{v_1, \ldots, v_{K'}\}$. Implicitly conditioned on S'_{k-1}, we have

$$\begin{aligned} f(S^{**}) &\leq f(S^{**} \cup S'_{k-1}) + Kc \\ &= f(S'_{k-1}) + f_{v_1}(S'_{k-1} \cup N_0) + f_{v_2}(S'_{k-1} \cup N_1) + \ldots + f_{v_{K'}}(S'_{k-1} \cup N_{K'-1}) + Kc \\ &\leq f(S'_{k-1}) + \sum_{\phi \in S^{**} \backslash S'_{k-1}} (f_\phi(S'_{k-1}) + c) + Kc \\ &\leq f(S'_{k-1}) + \frac{K}{1-\epsilon} \mathsf{E}[f(S'_k) - f(S'_{k-1})] + Kc + \frac{K}{1-\epsilon} c. \end{aligned} \tag{B.43}$$

The first inequality follows from Lemma 3 and $|S'_{k-1}|$ is upper bounded by K. The second inequality follows from c-submodularity. The third inequality follows from Lemma 4.

We take expectation of the last inequality in (B.43) over all S'_{k-1}, multiply both sides of (B.43) by $(1-\epsilon)/K$, and rearrange this inequality, then we have:

$$\frac{1-\epsilon}{K}(f(S^{**}) - \mathsf{E}[f(S'_{k-1})]) \leq \mathsf{E}[f(S'_k) - f(S'_{k-1})] + (2-\epsilon)c \tag{B.44}$$

Let $\delta' f_k = f(S^{**}) - \mathsf{E}[f(S'_k)]$ and (B.44) becomes

$$\frac{1-\epsilon}{K}\delta' f_{k-1} \leq (\delta' f_{k-1} - \delta' f_k) + (2-\epsilon)c. \tag{B.45}$$

Rearrange (B.45), we have

$$\delta' f_k \leq \left(1 - \frac{1-\epsilon}{K}\right)(\delta' f_{k-1}) + (2-\epsilon)c \tag{B.46}$$

Hence,

$$\begin{aligned}
\delta' f_K &\leq \left((1 - \frac{1-\epsilon}{K}\right)\delta' f_{K-1} + (2-\epsilon)c \\
&\leq \left(1 - \frac{1-\epsilon}{K}\right)^2 \delta' f_{k-2} + \left(1 + (1 - \frac{1-\epsilon}{K})\right)(2-\epsilon)c \\
&\cdots \\
&\leq \left(1 - \frac{1-\epsilon}{K}\right)^K \delta' f_0 + \sum_{j=0}^{K-1}\left(1 - \frac{1-\epsilon}{K}\right)^j (2-\epsilon)c
\end{aligned} \tag{B.47}$$

Finally, we have

$$\mathsf{E}[f(S'_K)] \geq \left(1 - \frac{1}{e} - \epsilon\right) f(S^{**}) - (2-\epsilon)Kc. \tag{B.48}$$

This is because for $\epsilon \in [0,1]$

$$1 - \left(1 - \frac{1-\epsilon}{K}\right)^K \geq 1 - e^{-(1-\epsilon)} \geq 1 - \frac{1}{e} - \epsilon \tag{B.49}$$

and

$$-\sum_{k=0}^{K-1}\left(1 - \frac{1-\epsilon}{K}\right)^k \geq -K \tag{B.50}$$

In order to have a meaningful performance guarantee, we restrict $\epsilon \in (0, 1-1/e]$.

□

Lemma 6. *Step 2 of Hybrid returns a set X' such that*

$$f(S^{**}) \geq \frac{K}{G+K} f(X') - \frac{GK}{G+K} c, \tag{B.51}$$

where S^{**} is the optimal solution to Problem $\underline{\boldsymbol{P}}^{\boldsymbol{c}}$ in (B.16).

Proof. This inequality obviously holds when $|X'| \leq K$. Thus, we only consider $|X'| > K$ in this proof. First, we split X' into L disjoint subsets ($L = \lceil \frac{|X'|}{K} \rceil$): $\{A_1, \ldots, A_L\}$ where $A_1, \ldots, A_{L-1}$ are sets with size equal to K and A_L is a set with size equal to $|X'| \mod K$. We have

$$\begin{aligned} f(X') &= f(A_1 + X' \backslash A_1) \\ &\leq f(A_1) + f(X' \backslash A_1) + Kc \\ &\leq f(A_1) + f(A_2) + f(X' \backslash (A_1 + A_2)) + 2Kc \\ &\ldots \\ &\leq f(A_1) + \ldots + f(A_L) + (L-1)Kc \end{aligned} \tag{B.52}$$

We explain only the first inequality in (B.52) as the rest of the inequalities can be shown similarly. Suppose $A_1 = \{v_1, \ldots, v_K\}$. Using c-submodularity and $\emptyset \subset X' \backslash A_1$, we have

$$f_{v_1}(X' \backslash A_1) \leq f_{v_1}(\emptyset) + c, \tag{B.53}$$

Similarly:

$$f_{v_2}((X'\backslash A_1) + v_1) \quad \leq \quad f_{v_2}(v_1) + c, \tag{B.54}$$

$$\dots$$

$$f_{v_K}(X'\backslash v_K) \quad \leq \quad f_{v_K}(A_1\backslash v_K) + c, \tag{B.55}$$

Aggregate all these inequalities together, we have

$$f(X') - f(X'\backslash A_1) \quad \leq \quad f(A_1) - f(\emptyset) + Kc. \tag{B.56}$$

As $f(\emptyset)$ is non-negative, we have shown the first inequality in (B.52).

Next, we analyze the link between elements $A_1, \dots, A_L$ to S^{**}. As S^{**} is the optimal solution to Problem $\underline{\boldsymbol{P}}^{\boldsymbol{c}}$ in (B.16), then we have

$$f(A_k) \leq f(S^{**}),\ k = 1, \dots, L. \tag{B.57}$$

Combining these inequalities with the last inequality in (B.52), we have

$$\begin{aligned} f(X') &\leq Lf(S^{**}) + (L-1)Kc \\ &= \lceil \frac{|X'|}{K} \rceil f(S^{**}) + \lfloor \frac{|X'|}{K} \rfloor Kc \end{aligned} \tag{B.58}$$

As $|X'| \leq G$, we have

$$f(X') \quad \leq \quad \frac{G+K}{K} f(S^{**}) + Gc, \tag{B.59}$$

or equivalently

$$f(S^{**}) \geq \frac{K}{G+K} f(X') - \frac{GK}{G+K} c. \quad \square \tag{B.60}$$

Proof of Theorem 2*:* We have proved Lemma 2, 3 and 5, then the performance guarantee of Hybrid is at least $(K/2(G+K))(1-1/e-\epsilon)f(S^*)-((3GK)/(2(K+G)))(1-1/e-\epsilon)Kc-(2-\epsilon)Kc$, which is shown in the outline of the proof. For complexity, Double Greedy in step 1 requires $2G$ function evaluations. The set reduction process in step 2 requires at most G function evaluations. If Stochastic Greedy is executed in step 3, it requires at most $G\log(1/\epsilon)$ function evaluations. Thus, the complexity of Hybrid is $O(G\log(1/\epsilon))$. $\square$

B.2 Generalization to More than One Truck per Lane

We can generalize our problem to the scenario where there are multiple trucks in one arc. Suppose each edge can have at most U trucks. We change our decision set to $\mathcal{G} = \{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_U\}$ where $\mathcal{G}_1 = \{(i_1, j_1) : i_1 \in \mathcal{I}, j_1 \in \mathcal{J}\}, \mathcal{G}_2 = \{(i_2, j_2) : i_2 \in \mathcal{I}, j_2 \in \mathcal{J}\}, \ldots, \mathcal{G}_U = \{(i_U, j_U) : i_U \in \mathcal{I}, j_U \in \mathcal{J}\}$.

Then the general problem of network design with unknown flow response is

$$\begin{aligned} \min_{\mathcal{Y} \subseteq \mathcal{G} = \{\mathcal{G}_\infty, \mathcal{G}_\in, \ldots, \mathcal{G}_\mathcal{U}\}} C(\mathcal{Y}, \boldsymbol{s}, \boldsymbol{d}) \quad &= \quad \sum_{(i,j) \in \mathcal{Y}} \kappa_{ij} + \sum_{(i,j) \in \mathcal{G}, p \in \mathcal{P}} c_{ij} u_{ijp}(\mathcal{Y}, \boldsymbol{s}, \boldsymbol{d}) \\ s.t. \qquad & |\mathcal{Y}| \leq K. \end{aligned}$$

Similarly, the general problem of network design with predicted flow response is

$$\begin{aligned} \min_{\mathcal{Y}\subseteq\mathcal{G}=\{\mathcal{G}_\infty,\mathcal{G}_\in,\ldots,\mathcal{G}_\mathcal{U}\}} \hat{C}(\mathcal{Y}) &= \sum_{(i,j)\in\mathcal{Y}} \kappa_{ij} + \sum_{(i,j)\in\mathcal{G}} \hat{x}_{ij}(m_i(\mathcal{Y}), n_j(\mathcal{Y}), y_{ij}) \\ s.t. \quad & |\mathcal{Y}| \leq K. \end{aligned}$$

We can still use the same algorithm to solve this problem. For this problem the size of the ground set is linear in U.

B.3 Machine Learning Methods and Nested Cross-Validation

B.3.1 Common Machine Learning Methods

First, we describe the machine learning methods we used to build each predictive model. The first two, ridge regression and LASSO, assume a linear relationship between the features and outcome. The remaining three, decision tree regression, random forest regression, and support vector regression (SVR) with radial basis function (RBF), can handle non-linear relationships. Each of these methods has the model parameters, which are optimized via a training algorithm, and the hyper-parameters, which are set prior to the training. We use a training dataset of size N and Model 1 to illustrate how each machine learning method works.

- Ridge regression. This prediction has the form of $\hat{C}_2(\mathcal{Y}, \boldsymbol{\alpha}) = \beta_0 + \sum_{ij} \beta_{j+(i-1)J} y_{ij} + \sum_i \beta_{IJ+i} \breve{s}_i + \sum_i \beta_{IJ+I+j} \breve{d}_j$, where $\boldsymbol{\beta} = (\beta_0, \ldots, \beta_{IJ+I+J})$ are the model parameters. To derive the model parameters using the training dataset, for any

given hyper-parameter λ, we solve

$$\min_{\boldsymbol{\beta}} \Bigg\{ \sum_{n=1}^{N} \Bigg(C_2^n(\mathcal{Y}^n) - \beta_0 - \sum_{ij} \beta_{j+(i-1)J} \mathcal{Y}_{ij}^n - \sum_{i} \beta_{IJ+i} \breve{s}_i^n - \sum_{i} \beta_{IJ+I+j} \breve{d}_j^n \Bigg)^2 + \lambda \sum_{i=0}^{IJ+I+J} \beta_i^2 \Bigg\}. \tag{B.61}$$

Here, the ℓ_2-norm regularization makes the value of $\boldsymbol{\beta}$ smaller and helps prevent overfitting. This method reduces the model complexity from that under ordinary least squares. Ridge regression performs well if the response C_2 is impacted by the majority of the features.

- LASSO. This prediction has the same form and procedure as ridge regression. The only difference is that, the ℓ_2-norm term $\sum_{i=0}^{IJ+I+J} \beta_i^2$ in (B.61) is replaced by the ℓ_1-norm term $\sum_{i=0}^{IJ+I+J} |\beta_i|$,

  which shrinks the less important features' coefficients to 0. As a result, LASSO performs well if there are only a small number of features that impact the final response.

- Decision tree regression. This prediction has the form of a tree structure, as illustrated in Figure B.1. Each internal node (excluding the terminal nodes) represents a test on one splitting feature f' with threshold s', where f' and s' are the model parameters. The branch following the internal node represents the outcome of the test. The tree is equivalent to a series of such tests and the input feature space is recursively partitioned into a number of rectangular subspaces. Each terminal node (or leaf) represents a subspace. For an observation falling into a specific terminal node, the predicted flow response equals the mean response value of the observations at that node from the training

set. In Figure B.1, for example, $\breve{d}_1$ is the splitting feature f' and threshold 1 is the value of s' in the first internal node; an observation $(\mathcal{Y}, \boldsymbol{\alpha})$ with $\breve{d}_1$ larger than (less or equal to) threshold 1 is assigned to the left (right) branch following the first internal node; the first terminal node represents a subspace $R_1 = \{(\mathcal{Y}, \boldsymbol{\alpha}) : \breve{d}_1 > \text{threshold } 1, \breve{s}_2 > \text{threshold } 2, y_{12} > \text{threshold } 3\}$, and the predicted flow response equals $[\sum_{(\mathcal{Y}^n, \boldsymbol{\alpha}^n) \in R_1} C_2^n(\mathcal{Y}^n, \boldsymbol{\alpha}^n)]/|\{n : (\mathcal{Y}^n, \boldsymbol{\alpha}^n) \in R_1\}|$.

Some common hyper-parameters in decision tree regression include the maximum depth of the decision tree D and the minimum number of samples required to split an internal node S. Given any D and S, this method searches for a splitting feature f_1' and a threshold s_1' to divide the feature space, and then split it further on each branch. We stop splitting an internal node when either it reaches the maximum depth D or the number of observations in this node is smaller than S. In this constructed tree, let O be the number of terminal nodes, R_o the subspace corresponding to the o-th terminal node, and $\hat{C}_o$ the predicted flow response assigned to R_o. The decisions of the splitting features $f_1', f_2', \ldots$ and thresholds $s_1', s_2', \ldots$ are chosen to minimize the sum of the squares of the errors in each terminal node using the training dataset:

$$\sum_{o=1}^{O} \sum_{(\mathcal{Y}^n, \boldsymbol{\alpha}^n) \in R_o} \left(C_2^n(\mathcal{Y}^n, \boldsymbol{\alpha}^n) - \hat{C}_o \right)^2.$$

Constructing the optimal decision tree is NP-hard. Consequently, a heuristic decision tree is often constructed using a greedy algorithm.

The interpretation of the final result from this tree structure is simple if the tree is short. However, this method is not robust: a slight change in the training dataset can result in a different tree structure. Moreover, it fails to deal with linear relationships as it creates a step function, which lacks smoothness.

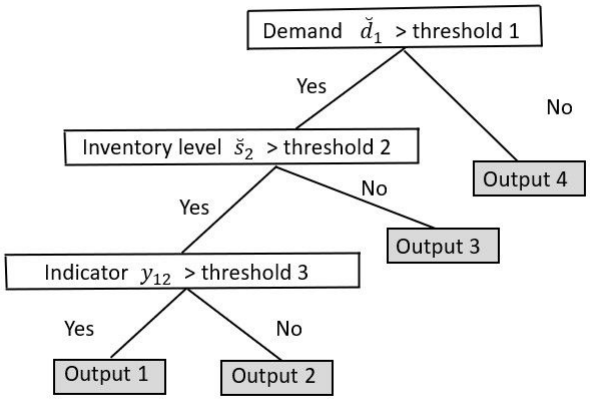


Figure B.1: Illustration of Decision Tree Regression.

- Random forest regression. To overcome the shortcomings of decision tree regression, this method uses the mean prediction of a collection of decision trees. The common hyper-parameters include the number of trees in the forest N_t, the number of samples to draw from dataset to train each decision tree n_s, the max number of features considered for splitting a node m_f, and the maximum depth of the decision tree D, and minimum number of samples required to split an internal node S. To train an individual decision tree, this method draws a random sample of size n_s with replacement from the training dataset, and repeats the same splitting process as in the decision tree regression using this sample. Different from decision tree regression, the splitting feature is chosen from a random subset of features with size m_f instead of all possible features. Using this method, the variance of the generalization error decreases as the number of trees increases, but at the expense of increased computational time and resources and less interpretability due to the infeasibility of examining each individual tree.

- SVR with RBF kernel. It has the form of $\hat{C}_2(\mathcal{Y}, \boldsymbol{\alpha}) = \sum_{n=1}^{N}(\beta_n^* - \beta_n)K\big((\mathcal{Y}, \boldsymbol{\alpha}),$

$(\mathcal{Y}^n, \boldsymbol{\alpha}^n)\big) + b$ where $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_N)$, $\boldsymbol{\beta^*} = (\beta_1^*, \ldots, \beta_N^*)$ and b are the model parameters, and $K(\cdot, \cdot)$ is radial basis function (RBF) kernel:

$$
\begin{aligned}
&K\big((\mathcal{Y}, \boldsymbol{\alpha}), (\mathcal{Y}', \boldsymbol{\alpha}')\big) \\
&= \exp\left\{-\frac{1}{IJ}\left(\sum_{i,j}(y_{ij} - y'_{ij})^2 + \sum_i (s_i - s'_i)^2 + \sum_j (d_j - d'_j)^2\right)\right\}.
\end{aligned}
$$

Given the hyper-parameters $\varepsilon, \bar{\beta}$, the model parameters are derived by maximizing

$$
\begin{aligned}
&-\frac{1}{2}\sum_{n=1}^{N}\sum_{n'=1}^{N}(\beta_n - \beta_n^*)(\beta_{n'} - \beta_{n'}^*)K\big((\mathcal{Y}^n, \boldsymbol{\alpha}^n), (\mathcal{Y}^{n'}, \boldsymbol{\alpha}^{n'})\big) - \varepsilon\sum_{n=1}^{N}(\beta_n + \beta_n^*) \\
&+\sum_{n=1}^{N} C_2^n(\mathcal{Y}^n, \boldsymbol{\alpha}^n)(\beta_n - \beta_n^*)
\end{aligned}
$$

subject to $\sum_{n=1}^{N}(\beta_n - \beta_n^*) = 0$ for $0 \leq \beta_n, \beta_n^* \leq \bar{\beta}$. Here, the computation of b is done by exploiting the Karush-KuhnTucker (KKT) conditions (Smola and Schölkopf 2004), and $\bar{\beta}$ is the regularization parameter.

The idea of SVR is to find a prediction function such that the observed $C_2^n(\mathcal{Y}^n, \boldsymbol{\alpha}^n)$ deviates from the predicted result by a value no greater than ε for each $(\mathcal{Y}^n, \boldsymbol{\alpha}^n)$. The use of Kernel function transforms the dataset into a higher dimensional space so that the non-linear problem with the original input features becomes a linear or approximately linear problem in the higher dimension. The Kernel function defines the inner product in the transformed feature space, which reduces the complexity of finding the mapping function of features and computing the coordinates in the transformed space. The drawbacks of this method include long computational time and lack of interpretability.

B.3.2 Nested Cross-validation

Nested cross-valuation tunes the hyper-parameters and provides an unbiased generalization performance of different machine learning methods. We illustrate the steps in an M-fold (typically, $M = 5$) nested cross-validation, using Model 1 and LASSO as the example. In nested cross-validation, we have an inner M-fold cross-validation loop to select the best hyper-parameters, and an outer M-fold cross-validation loop to compare different machine learning methods with the best hyper-parameters.

We divide the entire dataset into M outer folds where one outer fold is chosen as the test set, and each of the remaining $M-1$ outer folds is further split into M inner folds. In the inner loop, $M-1$ of the inner folds are selected as the training set, and the last fold is the validation set. For LASSO, we can choose a λ from $\{\lambda_1, \ldots, \lambda_L\}$, say λ_1. We first solve the model parameters $\boldsymbol{\beta}$ with λ_1 on the training dataset. Then we evaluate the performance of the LASSO prediction with the derived $\boldsymbol{\beta}$ and λ_1 on the validation set and record its MAPE. We repeat this process for all potential λs. After we complete this process with the current selection of the validation set, we rotate the validation set among the M inner folds. Now for LASSO and each hyper-parameter, we have recorded M MAPEs over the M validation sets. We choose the hyper-parameter λ^* with the lowest average MAPE as the best hyper-parameter. In the outer loop, the best hyper-parameter λ^* is used to train the model on the entire dataset, excluding the test set. Then we evaluate its MAPE on the test set. We also rotate the selection of the test set among the M outer folds and report MAPE on each outer fold. Eventually, we record the average MAPE over the M test sets for LASSO. This value reflects how good LASSO is after tuning its hyper-parameters.

For each predictive model, we repeat the same procedure and record this value for every machine learning method, and then select the predictive model and machine

learning method with the lowest average MAPE.

We also introduce cross-validation used in step (iii): it is similar to the nested cross-validation above except that it does not has the inner loop: it splits the dataset into M folds, where one fold is the test set, and the remaining $M-1$ folds are the training set. Each group of the hyper-parameters is selected to train the model on the training set and is then tested on the test set. For example, suppose Model 1 and LASSO is selected in the previous step. Then we first optimize $\boldsymbol{\beta}$ on the training set using LASSO with each hyper-parameter in $\{\lambda_1, \ldots, \lambda_L\}$. Each optimized model is then evaluated on the test set. The selection of the test set is rotated M times, and we choose the hyper-parameters with the lowest average MAPE over the M test sets.

B.4 Mixed-Integer Linear Program (MILP) Formulation

The decision variables are $\boldsymbol{y} = \{y_{ij} \in \{0,1\} : i \in \mathcal{I}, j \in \mathcal{J}\}$. To linearize the predicted flow response $\hat{x}_{ij}(\sum_j y_{ij}, \sum_i y_{ij}, y_{ij})$ on edge (i,j), we introduce several auxiliary binary variables: v_{il}, $w_{jl'}$, $z_{ijll'}$ and $z'_{ijll'}$, where $v_{il} = 1$ if and only if l edges ($l \in \{0,1,\ldots,J\}$) are assigned to origin i; $w_{jl'} = 1$ if and only if l' edges ($l' \in \{0,1,\ldots,I\}$) are assigned to destination j; $z_{ijll'} = 1$ if and only if $v_{il} = 1$, $w_{jl'} = 1$ and $y_{ij} = 0$, i.e., $z_{ijll'} = v_{il}w_{jl'}(1-y_{ij})$; $z'_{ijll'} = 1$ if and only if $v_{il} = 1$, $w_{jl'} = 1$ and $y_{ij} = 1$, i.e., $z'_{ijll'} = v_{il}w_{jl'}y_{ij}$. Using these auxiliary variables, the predicted flow response on edge (i,j) can be reformulated as $\sum_{l=0}^{J}\sum_{l'=0}^{I}\left(\hat{x}_{ij}(l,l',0)z_{ijll'} + \hat{x}_{ij}(l,l',1)z'_{ijll'}\right)$. In the MILP formulation below, we also linearize the equations $z_{ijll'} = v_{il}w_{jl'}(1-y_{ij})$ and $z'_{ijll'} = v_{il}w_{jl'}y_{ij}$ as they are products of three binary variables. We use the bold symbols $\boldsymbol{v}, \boldsymbol{w}, \boldsymbol{z}$ and $\boldsymbol{z'}$ to represent the sets of the auxiliary binary variables.

$$
\begin{aligned}
\boldsymbol{P}_{\text{MILP}}\colon \min_{\boldsymbol{y},\boldsymbol{v},\boldsymbol{w},\boldsymbol{z},\boldsymbol{z'}} \quad & \sum_{ij} y_{ij}\kappa_{ij} + \sum_{ij}\sum_{l=0}^{J}\sum_{l'=0}^{I}\left(\hat{x}_{ij}(l,l',0)z_{ijll} + \hat{x}_{ij}(l,l',1)z'_{ijll'}\right) \\
s.t. \quad & \sum_{l=0}^{J} v_{il} = 1,\ \sum_{l=0}^{J} l\cdot v_{il} = \sum_{j} y_{ij},\ \forall i, \\
& \sum_{l'=0}^{I} w_{jl'} = 1,\ \sum_{l'=0}^{I} l'\cdot w_{jl'} = \sum_{i} y_{ij},\ \forall j, \\
& z_{ijll'} \le v_{il},\ z_{ijll'} \le w_{jl'},\ z_{ijll'} \le 1 - y_{ij},\ \forall i,j,l,l', \\
& z_{ijll'} \ge 0,\ z_{ijll'} \ge v_{il} + w_{jl'} + (1-y_{ij}) - 2,\ \forall i,j,l,l', \quad \text{(B.62)} \\
& z'_{ijll'} \le v_{il},\ z'_{ijll'} \le w_{jl'},\ z'_{ijll'} \le y_{ij},\ \forall i,j,l,l', \\
& z'_{ijll'} \ge 0,\ z'_{ijll'} \ge v_{il} + w_{jl'} + y_{ij} - 2,\ \forall i,j,l,l', \\
& \sum_{i,j} y_{ij} \le K, \\
& y_{ij}, v_{il}, w_{jl'}, z_{ijll'}, z'_{ijll'} \in \{0,1\},\ \forall i,j,l,l'.
\end{aligned}
$$

Problem $\boldsymbol{P}_{\text{MILP}}$ has $O(G^2)$ binary variables and constraints.

B.5 Network-decomposition Prediction with Random Network Generation

We have reported the prediction accuracy of 3 different predictive models using the data generated via the incumbent network configuration policy in Table 3.2. Here, we study a different setting where the networks are randomly generated with each edge has a 50% chance of being activated. We consider two different sizes of the networks: $I = 3, J = 20$ and $I = 10, J = 20$. We only show the result when the fulfillment policy is the myopic policy. The rest of the setting is the same as in Section 3.4.2.1.

In Table B.1, we first find that Model 2 and 3 are more accurate than Model 1

Table B.1: Average and Standard Error of MAPE on Flow Response in Nested Cross-validation.

$I = 3, J = 20$						
Machine Learning Method	Average of MAPE			Standard Error of MAPE		
	Model 1	Model 2	Model 3	Model 1	Model 2	Model 3
Lasso	13.47	8.07	9.09	3.25	1.09	1.32
Ridge Regression	9.03	7.93	9.12	1.08	1.05	1.34
Decision Tree Regression	16.96	5.52	7.29	2.20	0.51	0.84
Random Forest Regression	10.51	4.63	**4.69**	1.71	1.21	0.28
SVR with RBF kernel	9.06	**3.24**	6.66	1.16	0.42	0.73
$I = 10, J = 20$						
Machine Learning Method	Average of MAPE			Standard Error of MAPE		
	Model 1	Model 2	Model 3	Model 1	Model 2	Model 3
Lasso	12.25	10.17	12.25	2.91	1.19	2.91
Ridge Regression	16.50	9.82	12.26	2.10	1.01	2.92
Decision Tree Regression	12.69	10.66	9.70	2.27	1.29	0.94
Random Forest Regression	9.41	**8.44**	**6.99**	0.59	1.57	1.14
SVR with RBF kernel	11.72	8.97	11.81	0.78	3.09	2.86

when $I = 3, J = 20$. It shows that our network-decomposition prediction approach can improve the prediction accuracy if the networks are randomly generated.

We also note that Model 2 is more accurate than Model 3. We speculate the reason is that the sizes of the input features are very close between Model 2 and 3 when I is small. For the other network with $I = 10, J = 20$, we find Model 3 becomes the most accurate when I is large.

Appendix C

Appendix for Chapter 4

C.1 Line Graph

We use $(i,j),(i,j) \in E$ to denote nodes in the new graph. We consider edges are undirected here.

New nodes $(i,j),(i',j') \in E$ are connected if either $S_{(i,j)} = S_{(i',j')}$ or $D_{(i,j)} = D_{(i',j')}$.

For the node features on node (i,j), we use $x_{(i,j)}$.

Algorithm 8: Transformation into Line Graph

Inputs: Graph $G(V,E)$
Output: Graph $G(V',E')$
$V' = E, E' = \emptyset$
for $(i,j) \in E$ **do**
  for $(i',j') \in E$ **do**
    if $(i,j) \neq (i',j')$ & $(S_{(i,j)} = S_{(i',j')} \| D_{(i,j)} = D_{(i',j')})$ **then**
      Add $((i,j),(i',j'))$ into E'
    end
  end
end

The idea of line graph is illustrated using a 2x3 network in Figure C.1.

C.2 Explanation of the Benchmark Learning Algorithms

We explain the idea of the bag of nodes and bag of edges, convolutional neural network and random forest.

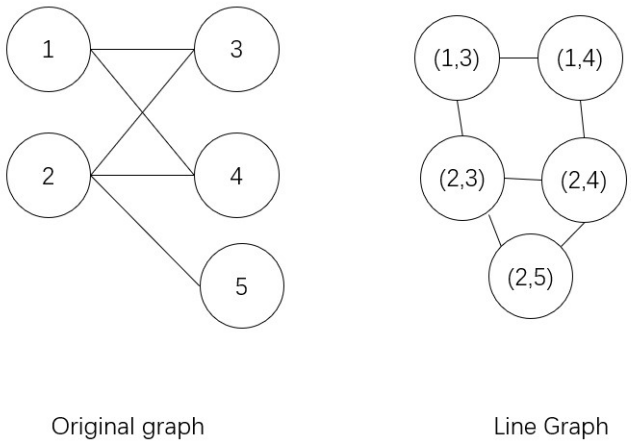


Figure C.1: Illustration of Line Graph.

Bag of nodes (edges): First, we define graph-level features by aggregating node-level (edge-level) statistics. The node features can be the summary statistics, such as degrees, centralities, and clustering coefficients of the nodes in the graph, and any other attributes to describe the node. This aggregated information is then used as the graph-level features.

Convolutional neural network: For prediction problems that have image- or grid-like inputs, convolutional neural networks have been shown to be particularly effective. We consider a specific neural network architecture. The input layer of the neural network is an mxn grid, and the output layer is a single unit that outputs a continuous value corresponding to the graph-level operational performance. Between the input and output layers are two hidden layers with 1024 and 128 hidden units, respectively.

Random forest: For each graph, we have $m + n$ $(m \cdot n)$ observations of nodes (edges). We apply random forest in node-level (edge-level) predictions, and then aggregate the predictions over all nodes (edges) to get the graph-level prediction.

C.3 Evaluation Framework

The following algorithms describe the evaluation framework for our numerical test, which combines model assessment (Algorithm 10) and an internal model selection procedure (Algorithm 9).

Algorithm 9: Select (Model Selection)

Inputs: train_i,Θ
Split train_i into *train* and *valid*
$p_\theta = \emptyset$
for $\theta \in \Theta$ **do**
  model $\leftarrow$ Train(train_i, θ)
  $p_\theta \leftarrow p_\theta \cup \text{Eval}(\text{model}, \mathit{valid})$
end
$\theta^{\text{best}} \leftarrow \arg\max_\theta p_\theta$
return θ^{best}

In Algorithm 9: the split of train_i is based on a 75%/25% split.

Eval is the function which evaluates the input model on the input dataset.

Train is the function which trains the model with input hyperparameter set on the input dataset. In model selection, we have Train(train_i, θ) = Train(*train*, *valid*, θ), which means we use *train* to train the model with hyperparameter set θ, and *valid* for early stopping (note that the criterion for early stopping is part of θ). The reason why we can directly use *valid* for early stopping is because validation performance itself is a biased estimate of the true generalization capabilities as it is used for tuning the hyperparameter. The real generalization performance will be evaluated via model assessment instead.

In Algorithm 10: Select is the function which does the model selection as in Algorithm 9: it selects the best hyperparameter set based on the given dataset and full set of hyperparameters.

For Train(train_i, θ_i^{best}), we will split train_i into two datasets $\mathit{subtrain}_i$ and $\mathit{subvalid}_i$

Algorithm 10: Model Assessment

Inputs: Dataset $\mathcal{D}$, set of configurations Θ
Split $\mathcal{D}$ into k folds $F_1, \ldots, F_k$
for $i = 1$ *to* k **do**
  $\text{train}_i, \text{test}_i \leftarrow \cup_{j \neq i} F_j, F_i$
  $\theta_i^{\text{best}} \leftarrow \text{Select}(\text{train}_i, \Theta)$
  for $r = 1$ *to* R **do**
    $\text{model}_r \leftarrow \text{Train}(\text{train}_i, \theta_i^{\text{best}})$
    $p_r \leftarrow \text{Eval}(\text{model}_r, \text{test}_i)$
  end
  $\text{perf}_i \leftarrow \sum_{r=1}^{R} p_r / R$
end
return $\sum_{i=1}^{k} \text{perf}_i / k$

based on a 75%/25% split, and $\text{Train}(\text{train}_i, \theta_i^{\text{best}}) = \text{Train}(subtrain_i, subvalid_i, \theta_i^{\text{best}})$, which means we use $subtrain_i$ to train the model with hyperparameter set θ_i^{best}, and $subvalid_i$ for early stopping.

We have $R = 3$ separate runs to smooth the effect of unfavorable random weight initialization on test performance.

C.4 Data Generation

We explain the details of the data generation setup for the two supply chain systems here.

Assemble-to-order system. In each dataset, a set of system parameters are randomly generated. The holding costs are uniformly selected from $\{1, \ldots, 10\}$. The leadtimes are uniformly selected from $\{1, \ldots, 10\}$. The demand rates are uniformly selected from $\{1, 2, 3\}$ and the backorder cost rates are $b_j = \sum_{i \in N(i)} h_i \cdot (1 - 0.7)/0.7$. The inventory levels s are set to achieve 70% service level. The inventory cost are the average cost over the time until any product see the 10,000 orders. The inventory follows the constant base-stock policy and the component allocation follows a first-

come-first-served rule. 100 bill of materials are randomly generated. In each bill of materials, the first 30% of components are required by 80% of products while the rest 20% of components are required by 20% of products.

Process flexibility network. Rand10x10, Rand6x43, Rand17x17, Rand12x21 are based on four fixed charge transportation problem (FCTP) instances from Mittelmann and Spellucci (2005). In an FCTP, products are shipped from origins with fixed capacities to destinations with fixed demands, and the goal is to minimize the total transportation cost plus the fixed cost of transporting on an origin-destination transportation network. An FCTP can be similar to a process flexibility design problem with deterministic demand, with the origins viewed as supply nodes and destinations viewed as demand nodes. To create our process flexibility network datasets, we set the unit profit from resource i to demand j, p_{ij}, to be

$$p_{ij} = t_{\max} - t_{ij},$$

where t_{ij} is the unit transportation cost from the FCTP instance from origin i to destination j, and $t_{\max}$ is the largest unit transportation cost among all arcs in the given transportation network. We set the capacity vector as the capacity of the origins and the mean for demand nodes, $\boldsymbol{\mu}$, as the demand of the destinations from the FCTP problem. We further set that the distributions of demands as independent and normal with mean $\boldsymbol{\mu}$ and standard deviation $\boldsymbol{\sigma} = 0.8\boldsymbol{\mu}$, and we then generate scenarios of the demands based on these distributions. JG is based on the test scenario "Automotive Manufacturing", and Obermeyer is based on the test scenario "Fashion Manufacturing" from Wei et al. (2021). 100 networks are randomly generated and an edge is connected with probability $(0.5 - 1.5 \cdot \max\{m, n\}/(m \cdot n)) \cdot 100\%$ in each graph.

Bibliography

Acimovic J, Graves SC (2014) Making better fulfillment decisions on the fly in an online retail environment. *Manufacturing & Service Operations Management* 17(1):34–51.

Acimovic J, Graves SC (2017) Mitigating spillover in online retailing via replenishment. *Manufacturing & Service Operations Management* 19(3):419–436.

Albrecht M (2014) Determining near optimal base-stock levels in two-stage general inventory systems. *European Journal of Operational Research* 232(2):342–349.

Amazon.com, Inc (2018) Form 10-k for 2018. https://www.sec.gov/Archives/edgar/data/1018724/000101872419000004/amzn-20181231x10k.htm.

Anderson C (2006) *The long tail: why the future of business is selling less of more* (Hachette Books).

Ang E, Kwasnick S, Bayati M, Plambeck EL, Aratow M (2016) Accurate emergency department wait time prediction. *Manufacturing & Service Operations Management* 18(1):141–156.

Ata B, Kumar S (2005) Heavy traffic analysis of open processing networks with complete resource pooling: Asymptotic optimality of discrete review policies. *Annals of Applied Probability* 15(2):331–391–516.

Atan Z, Ahmadi T, Stegehuis C, de Kok T, Adan T (2017) Assemble-to-order systems: a review. *European Journal of Operational Research* 261(3):866–879.

Baardman L, Boroujeni SB, Cohen-Hillel T, Panchamgam K, Perakis G (2020) Detecting customer trends for optimal promotion targeting. *Manufacturing & Service Operations Management* .

Baardman L, Cohen MC, Panchamgam K, Perakis G, Segev D (2018) Scheduling promotion vehicles to boost profits. *Management Science* .

Ban GY, Rudin C (2018) The big data newsvendor: practical insights from machine learning. *Operations Research* 67(1):90–108.

Bayati M, Shah D, Sharma M (2008) Max-product for maximum weight matching: Convergence, correctness, and lp duality. *IEEE Transactions on information theory* 54(3):1241–1251.

Bengio Y, Courville A, Vincent P (2013) Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence* 35(8):1798–1828.

Benjaafar S, ElHafsi M (2006) Production and inventory control of a single product assemble-to-order system with multiple customer classes. *Management Science*

52(12):1896–1912.

Bertsimas D, Kallus N (2020) From predictive to prescriptive analytics. *Management Science* 66(3):1025–1044.

Bravo F, Shaposhnik Y (2020) Mining optimal policies: A pattern recognition approach to model analysis. *INFORMS Journal on Optimization* 2(3):145–166.

Buchbinder N, Feldman M, Naor JS, Schwartz R (2014) Submodular maximization with cardinality constraints. *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, 1433–1452 (Society for Industrial and Applied Mathematics).

Buchbinder N, Feldman M, Seffi J, Schwartz R (2015) A tight linear time (1/2)-approximation for unconstrained submodular maximization. *SIAM Journal on Computing* 44(5):1384–1402.

CBRE (2019) How high will e-commerce sales go? URL http://www.cbre.us/real-estate-services/real-estate-industries/omnichannel/the-definitive-guide-to-omnichannel-real-estate/by-the-numbers/how-high-will-e-commerce-sales-go/.

Chan T, Letourneau D, Potter B (2019) Sparse flexible design: a machine learning approach. *Available at SSRN 3390000* .

Chan TC, Letourneau D, Potter BG (2022) Sparse flexible design: a machine learning approach. *Flexible Services and Manufacturing Journal* 1–51.

Chen F, Zheng Y (1994) Lower bounds for multi-echelon stochastic inventory systems. *Management Science* 40(11):1426–1443.

Chen S, Song JSJ, Wei Y (2020) Data-Driven Scalable E-commerce Transportation Network Design with Unknown Flow Response. *Available at SSRN 3590865* .

Chen X, Owen Z, Pixton C, Simchi-Levi D (2015) A statistical learning approach to personalization in revenue management. *Available at SSRN 2579462* .

Cheng F, Ettl M, Lin G, Yao DD (2002) Inventory-service optimization in configure-to-order systems. *Manufacturing & Service Operations Management* 4(2):114–132.

Chicago Consulting (2019) Ten best warehouse networks, optimization, consultant in USA. URL http://www.chicago-consulting.com/ten-best-warehouse-networks-consultant-usa/.

Cohen M, Kalas J, Perakis G (2017) Optimizing promotions for multiple items in supermarkets. *Available at SSRN 3061451* .

Cohen MC, Leung NHZ, Panchamgam K, Perakis G, Smith A (2017a) The impact of linear optimization on promotion planning. *Operations Research* 65(2):446–468.

Corbett E (2018) Amazon prime members ordered more than 2 nillion products for 1-day or sooner delivery in 2018. URL https://fortune.com/2018/12/03/amazon-prime-members-2-billion-products-2018//.

Cui R, Gallino S, Moreno A, Zhang DJ (2018) The operational value of social media information. *Production and Operations Management* 27(10):1749–1769.

DeValve L, Pekeč S, Wei Y (2020) A primal-dual approach to analyzing ATO systems. *Management Science* 66:5389–5407.

DeValve L, Wei Y, Wu D, Yuan R (2021) Understanding the value of fulfillment flexibility in an online retailing environment. *Manufacturing & Service Operations Management* .

Doğru M, Reiman M, Wang Q (2010) A stochastic programming based inventory policy for assemble-to-order systems with application to the W model. *Operations research* 58(4):849–864.

Doğru M, Reiman M, Wang Q (2017) Assemble-to-order inventory management via stochastic programming: chained BOMs and the M-system. *Production and Operations Management* 26(3):446–468.

ElHafsi M, Fang J, Hamouda E (2020) A novel decomposition-based method for solving general-product structure assemble-to-order systems. *European Journal of Operational Research* .

Elmachtoub AN, Grigas P (2017) Smart "predict, then optimize". *arXiv preprint arXiv:1710.08005* .

Elmachtoub AN, Liang JCN, McNellis R (2020) Decision Trees for Decision-Making under the Predict-then-Optimize Framework. *arXiv preprint arXiv:2003.00360* .

Errica F, Podda M, Bacciu D, Micheli A (2019) A Fair Comparison of Graph Neural Networks for Graph Classification. *International Conference on Learning Representations*.

Ferreira KJ, Lee BHA, Simchi-Levi D (2015) Analytics for an online retailer: demand forecasting and price optimization. *Manufacturing & Service Operations Management* 18(1):69–88.

Gale D, Politof T (1981) Substitutes and complements in network flow problems. *Discrete Applied Mathematics* 3(3):175–186.

Gamarnik D, Shah D, Wei Y (2012) Belief propagation for min-cost network flow: Convergence and correctness. *Operations research* 60(2):410–428.

Gong L, Cheng Q (2019) Exploiting edge features for graph neural networks. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 9211–9219.

Gori M, Monfardini G, Scarselli F (2005) A new model for learning in graph domains.

Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005., volume 2, 729–734 (IEEE).

Guo X, Grushka-Cockayne Y, De Reyck B (2021) Forecasting airport transfer passenger flow using real-time data and machine learning. *Manufacturing & Service Operations Management* .

Hamilton WL (2020) Graph representation learning. *Synthesis Lectures on Artifical Intelligence and Machine Learning* 14(3):1–159.

Hamilton WL, Ying R, Leskovec J (2017) Inductive representation learning on large graphs. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 1025–1035.

Herrera S (2019) How Amazon's shipping empire is challenging UPS and FedEx. https://www.wsj.com/articles/how-amazons-shipping-empire-is-challenging-ups-and-fedex-11567071003.

Huang K, de Kok T (2015) Optimal FCFS allocation rules for periodic-review assemble-to-order systems. *Naval Research Logistics* 62(2):158–169.

Jain A, Liu I, Sarda A, Molino P (2019) Food discovery with uber eats: Using graph learning to power recommendations. *Accessed March* 1:2021.

Jasin S, Sinha A (2015) An LP-based correlated rounding scheme for multi-item ecommerce order fulfillment. *Operations Research* 63(6):1336–1351.

JD.com, Inc (2018) 2017 annual report. http://ir.jd.com/annual-reports.

Kapuscinski R, Zhang RQ, Carbonneau P, Moore R, Reeves B (2004) Inventory decisions in Dell's supply chain. *Interfaces* 34(3):191–205.

Kipf TN, Welling M (2016) Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* .

LeCun Y, Bengio Y, Hinton G (2015) Deep learning. *nature* 521(7553):436–444.

Lei Y, Jasin S, Sinha A (2018) Joint dynamic pricing and order fulfillment for e-commerce retailers. *Manufacturing & Service Operations Management* 20(2):269–284.

Levs J (2013) UPS backlog means missing Christmas gifts; Amazon responds. URL https://www.cnn.com/2013/12/25/us/ups-delays/index.html.

Liu S, He L, Max Shen ZJ (2021) On-time last-mile delivery: Order assignment with travel-time predictors. *Management Science* 67(7):4095–4119.

Lu L, Song J, Zhang H (2015) Optimal and asymptotically optimal policies for assemble-to-order N-and W-systems. *Naval Research Logistics* 62(8):617–645.

Lu Y, Song J (2005) Order-based cost optimization in assemble-to-order systems. *Operations Research* 53(1):151–169.

Lu Y, Song J, Yao D (2003) Order fill rate, leadtime variability, and advance demand information in an assemble-to-order system. *Operations Research* 51(2):292–308.

Lu Y, Song J, Zhao Y (2010) No-holdback allocation rules for continuous-time assemble-to-order systems. *Operations Research* 58(3):691–705.

Maimon O, Rokach L (2005) Decomposition methodology for knowledge discovery and data mining. *Data mining and knowledge discovery handbook*, 981–1003 (Springer).

Mangasarian OL, Shiau TH (1987) Lipschitz continuity of solutions of linear inequalities, programs and complementarity problems. *SIAM J. Control and Optimizations* 25:583–595.

Mirzasoleiman B, Badanidiyuru A, Karbasi A, Vondrák J, Krause A (2015) Lazier than lazy greedy. *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Mitra M (2019) Shoppers ring up record Thanksgiving sales online. Black Friday's tally to exceed $7 billion, Adobe says. URL https://www.cnbc.com/2019/11/29/black-friday-online-sales-up-19percent-by-9-am-thanksgiving-sales-hit-record-online.html/.

Mittelmann HD, Spellucci P (2005) Decision tree for optimization software.

Muller J (2010) BMW's push for made-to-order cars. Forbes (September 9).

Nadar E, Akan M, Scheller-Wolf A (2014) Optimal structural results for assemble-to-order generalized M-systems. *Operations Research* 62(3):571–579.

Nadar E, Akan M, Scheller-Wolf A (2016) Experimental results indicating lattice-dependent policies may be optimal for general assemble-to-order systems. *Production and Operations Management* 25(4):647–661.

Naughton K, Boyle M (2019) Walmart targets automated 'middle-mile' delivery to cut shipping costs. https://www.ttnews.com/articles/walmart-targets-automated-middle-mile-delivery-cut-shipping-costs.

Nemhauser GL, Wolsey LA, Fisher ML (1978) An analysis of approximations for maximizing submodular set functions—I. *Mathematical programming* 14(1):265–294.

Perozzi B, Al-Rfou R, Skiena S (2014) Deepwalk: Online learning of social representations. *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 701–710.

Plambeck E, Ward A (2006) Optimal control of a high-volume assemble-to-order system. *Mathematics of Operations Research* 31(3):453–477.

Reiman M, Wan H, Wang Q (2020) Asymptotically optimal inventory control for assemble-to-order systems. *arXiv preprint arXiv:1809.08271* .

Reiman M, Wang Q (2015) Asymptotically optimal inventory control for assemble-to-order systems with identical lead times. *Operations Research* 63(3):716–732.

Rosling K (1989) Optimal inventory policies for assembly systems under random demands. *Operations Research* 37(4):565–579.

Salari N, Liu S, Shen ZJM (2022) Real-time delivery time forecasting and promising in online retailing: when will your package arrive? *Manufacturing & Service Operations Management* .

Scarselli F, Gori M, Tsoi AC, Hagenbuchner M, Monfardini G (2008) The graph neural network model. *IEEE transactions on neural networks* 20(1):61–80.

Schoolov K (2019) Amazon is rapidly expanding its air fleet to handle more of its own shipping. URL https://www.cnbc.com/2019/02/15/amazon-will-compete-with-fedex-and-ups-to-become-logistics-company.html.

ScrapeHero (2018) How many products does amazon sell? URL https://www.scrapehero.com/many-products-amazon-sell-january-2018/.

Shang Y, Dunson D, Song JS (2017) Exploiting big data in logistics risk assessment via bayesian nonparametrics. *Operations Research* 65(6):1574–1588.

Shapiro A, Dentcheva D, Ruszczyński A (2014) *Lectures on stochastic programming: modeling and theory* (SIAM).

Shen X, Pan S, Liu W, Ong YS, Sun QS (2018) Discrete network embedding. *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, 3549–3555.

Smola AJ, Schölkopf B (2004) A tutorial on support vector regression. *Statistics and computing* 14(3):199–222.

Song J (1998) On the order fill rate in a multi-item, base-stock inventory system. *Operations Research* 46(6):831–845.

Song J (2002) Order-based backorders and their implications in multi-item inventory systems. *Management Science* 48(4):499–516.

Song J, Xu S, Liu B (1999) Order-fulfillment performance measures in an assemble-to-order system with stochastic leadtimes. *Operations Research* 47(1):131–149.

Song J, Zhao Y (2009) The value of component commonality in a dynamic inventory system with lead times. *Manufacturing & Service Operations Management* 11(3):493–508.

Song J, Zipkin P (2003) Supply chain operations: assemble-to-order systems. *Handbooks in Operations Research and Management Science* 11:561–596.

Statista (2019) E-commerce share of total global retail sales from 2015 to 2023. URL https://www./statistics/534123/e-commerce-share-of-retail-sales-worldwide/.

van Jaarsveld W, Scheller-Wolf A (2015) Optimization of industrial-scale assemble-to-order systems. *INFORMS Journal on Computing* 27(3):544–560.

Veličković P, Cucurull G, Casanova A, Romero A, Liò P, Bengio Y (2018) Graph Attention Networks. *International Conference on Learning Representations*.

Wei Y, Zhang L, Zhang R, Si S, Zhang H, Carin L (2021) Reinforcement Learning for Flexibility Design Problems. *arXiv preprint arXiv:2101.00355* .

Wu Z, Pan S, Chen F, Long G, Zhang C, Philip SY (2020) A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems* 32(1):4–24.

Xiao Y, Chen J, Lee CY (2010) Single-period two-product assemble-to-order systems with a common component and uncertain demand patterns. *Production and Operations Management* 19(2):216–232.

Xu PJ, Allgor R, Graves SC (2009) Benefits of reevaluating real-time order fulfillment decisions. *Manufacturing & Service Operations Management* 11(2):340–355.

Yang H, Pan S, Zhang P, Chen L, Lian D, Zhang C (2018) Binarized attributed network embedding. *2018 IEEE International Conference on Data Mining (ICDM)*, 1476–1481 (IEEE).

Yu S (2019) Double 11 record sales signal strength of Chinese consumption. URL https://www.chinadaily.com.cn/a/201911/12/WS5dca40a8a310cf3e35576d7f.html/.

Zhao Y, Simchi-Levi D (2006) Performance analysis and evaluation of assemble-to-order systems with stochastic sequential lead times. *Operations Research* 54(4):706–724.

Zhao Y, Wang X, Xin L (2020) Multi-Item online order fulfillment: a competitive analysis, working paper, University of Chicago Booth School of Business.

Zhou J, Cui G, Hu S, Zhang Z, Yang C, Liu Z, Wang L, Li C, Sun M (2020) Graph neural networks: A review of methods and applications. *AI Open* 1:57–81.