Large-Scale Optimization in Online-Retail Inventory Management

by

Annie I-An Chen

S.M., Massachusetts Institute of Technology (2012)
B.S., National Taiwan University (2010)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2017

© Massachusetts Institute of Technology 2017. All rights reserved.

Author .................................................................
Department of Electrical Engineering and Computer Science
May 17, 2017

Certified by ..............................................................
Stephen C. Graves
Abraham J. Siegel Professor of Management Science
Thesis Supervisor

Accepted by .............................................................
Leslie A. Kolodziejski
Professor
Chair of the Committee on Graduate Students
Large-Scale Optimization in
Online-Retail Inventory Management

by

Annie I-An Chen

Submitted to the Department of Electrical Engineering and Computer Science on May 17, 2017, in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

This thesis studies the related decisions of inventory placement and inventory replenishment in online retail. The placement decision focuses on the selection of fulfillment centers in which to place items, while the replenishment decision concerns the amount of inventory to order for each item. In contrast with traditional retail, a distinctive feature of online retail is the flexibility to ship items to customers from different fulfillment centers. This creates interdependence between fulfillment centers and poses new challenges in inventory management.

The placement decision can be formulated as a mixed-integer program. The objective is to minimize the sum of outbound shipping and fixed costs for all the items, while satisfying demand and capacity constraints. The large scale of the problem is due to the size of the fulfillment center network and the number of items. We propose a large-scale solution scheme that aggregates the items, solves a column-based reformulation of the aggregated problem using column generation, and then disaggregates the solution into placement plans for the individual items.

The replenishment decision for a single item can be formulated as a dynamic program, in which the objective is to minimize the long-term average of outbound shipping, stockout and holding costs. Its scale and complexity arises from demand stochasticity, inventory dynamics, non-identical fulfillment center lead times, and the integrality of inventory units. We propose approximation schemes that employ simulation to evaluate and optimize parameterized policies. In particular, we design the simulation to estimate gradient-like information, and use the information to enhance the efficiency a random search method.

For both problems, realistic numerical examples demonstrate that the large-scale optimization methods produce low-cost solutions in a relatively short amount of time, compared to simple heuristics that do not involve much optimization. We also study the properties of solutions in order to obtain managerial insights on the impact of key factors, including cost parameters, capacity constraints, and lead times.
Thesis Supervisor: Stephen C. Graves
Title: Abraham J. Siegel Professor of Management Science
Praise be to God, who shows me His love through the Lord Jesus Christ, and who has given me this wonderful opportunity to work on this thesis.

I would like to thank my advisor, Professor Stephen Graves, whose knowledge, experience, vision, intuition, attention to detail, and patient guidance have all been instrumental to the completion of this thesis. Working with him has been inspirational both professionally and personally, and there is so much more I could learn from him.

I am grateful to my committee members, Professor Patrick Jaillet and Professor John Tsitsiklis, who have generously gifted me with their time and insights. Their feedback has been helpful in shaping this work.

This thesis also owes its existence to many fellow researchers in online retail, with whom I have enjoyed wonderful conversations that sparked important ideas. In particular, I would like to acknowledge the input of my “academic sibling” Jason Acimovic; research scientists from Amazon.com, Salal Humair, Deepak Bhatia, Onur Ozkok, Ali Sadighian, Alvaro Maggiar, Jingchen Wu; and many other researchers from the online retail industry whose names are kept confidential.

I am deeply appreciative of the support and companionship of my husband, Peng Shi, who has contributed greatly to the sharpening of arguments for this thesis, the refinement of my thinking as a researcher, and much more.

I am much obliged to Wesley Harrell and his team at Sloan Technology Services for their excellent maintenance of the research computing clusters, without which the extensive empirical analysis in this thesis would not have been possible. I am also thankful for the administrative assistance of Janet Fischer, William Tilden, and David Merrill throughout my inter-departmental research appointment.

The generous financial support from the MIT Skoltech Initiative, the MIT SUTD Collaboration, and Samsung Electronics Corporation have been indispensable to the completion of this work.

This thesis is dedicated to my husband, my parents, and the brothers and sisters of my Bible study small groups in the MIT Graduate Christian Fellowship.
Contents

1 Introduction ....................................................... 19
  1.1 Motivation .................................................... 19
  1.2 Thesis scope and related problems ......................... 20
    1.2.1 Types of decisions in online-retail inventory management .... 20
    1.2.2 Assumptions about other decisions ...................... 23
    1.2.3 Relationship between placement and replenishment decisions .... 24
  1.3 Thesis outline and summary of contributions ............... 26

2 Online-retail inventory placement ................................ 29
  2.1 Introduction .................................................. 29
    2.1.1 Related work ............................................. 30
    2.1.2 Setup of numerical experiments ......................... 43
  2.2 Mixed integer program formulations ........................ 50
    2.2.1 Problem statement ....................................... 50
    2.2.2 Alternative formulations ................................ 54
    2.2.3 Empirical analysis of MIP formulations .................. 57
    2.2.4 Conclusions and future directions ....................... 68
  2.3 A large-scale solution scheme ............................... 70
    2.3.1 Aggregation of items .................................... 72
    2.3.2 Bundle characterization ................................ 73
    2.3.3 Column-based reformulation ............................. 75
    2.3.4 Disaggregation into items ............................... 85
    2.3.5 Enhancement of item placement plans .................... 88
2.4 Empirical analysis

2.4.1 Overview
2.4.2 Effects of problem size and aggregation method
2.4.3 Solution quality and item characteristics
2.4.4 A sequential placement heuristic
2.4.5 Effects of capacity constraint tightness
2.4.6 Effects of the fixed cost parameter
2.4.7 Scaling up

2.5 Conclusions

3 Online-retail inventory replenishment

3.1 Introduction
3.1.1 Related work
3.1.2 Chapter overview
3.2 Dynamic programming model
3.2.1 Modeling assumptions
3.2.2 Dynamic program formulation
3.2.3 Bounds
3.2.4 Exact solution methods
3.2.5 Empirical analysis
3.3 Approximation schemes for large-scale problems
3.3.1 Approximation in policy space: Policy parameterization
3.3.2 Approximation in value space: Simulation
3.3.3 Simulation optimization of parameterized policies
3.4 Empirical analysis of approximation schemes
3.4.1 A realistic numerical example
3.4.2 Simulation configuration
3.4.3 Performance of simulation optimization methods
3.4.4 Performance of parameterized policies ................. 186
3.4.5 Lead time staggering effects ............................. 189
3.4.6 Correlations between fulfillment centers ................. 193
3.4.7 Sensitivity analysis ...................................... 197
3.5 Conclusions .................................................. 199
3.5.1 Summary of contributions ................................. 199
3.5.2 Future directions .......................................... 202

A Supplementary materials for inventory placement empirical analysis
A.1 Detailed results for MIP experiments ....................... 205
A.2 Detailed results for large-scale solution scheme experiments .... 215

B Supplementary materials for inventory replenishment empirical analysis
B.1 R-BEESE with and without gradient-based sampling ......... 221
B.2 Detailed empirical results for the target local service level heuristic . 223

Bibliography ...................................................... 225
List of Figures

2.1 Online-retail inventory placement is related to facility location and fixed-charge network flow problems. ......................... 33
2.2 For empirical analysis, we construct a realistic numerical example with 88 fulfillment centers (based on Amazon’s network) and 98 regions (based on US census data). ......................... 46
2.3 The shipping cost is a piecewise-linear function of distance, fitted from UPS data. ......................... 48
2.4 When the placement problem is converted to the conventional multi-commodity capacitated fixed-charge network flow problem, the network size scales with the number of items. ......................... 54
2.5 When solving the exact MIP formulation, the default cut generation setting in Gurobi performs best. ......................... 60
2.6 For small problems with 10 items, the top-performing MIP formulations are lin_cap/flow, lin_cap/frac, lin_strong/flow, lin_strong/frac, and sos_sum/flow. ......................... 63
2.7 For medium problems with 100 items, the lin_strong formulations produce near-optimal solutions in a couple of hours. ......................... 66
2.8 For large problems with 1000 items, none of the MIP formulations perform well. ......................... 66
2.9 Greater capacity slackness results in slightly improved MIP solution efficiency. ......................... 68
2.10 As capacity slackness increases, there is diminishing returns in cost reduction. ......................... 69
2.11 Our large-scale solution scheme consists of fives stages, labeled A-E: Aggregation, Bundle characterization, Column-based reformulation, Disaggregation, and Enhancement.

2.12 In Stage C, most of the progress in \((RMP)\) is made in the initial iterations.

2.13 Relative improvement is correlated with convergence gap and can be used as a termination criteria for \((RMP)\).

2.14 The column generation subproblems \((CG_j)\) take longer to compute as the iterations go on.

2.15 When the number of bundles is large (e.g., \(|J|=100\) or 1000), most bundles have no more than a few columns in the solution of Stage C. In those cases, disaggregation in Stage D is easy, and for a single column, disaggregation is trivial.

2.16 The large-scale solution scheme tends to concentrate the suboptimality of placement on low-demand items.

2.17 The sequential placement heuristic places items of medium-to-low demand suboptimally.

2.18 As capacity constraints become tighter, the cost increases at an increasing rate, and the optimality gap of our method outperforms the sequential placement heuristic more significantly.

2.19 When the unit fixed cost is larger, fewer fulfillment centers are used, and the shipping cost increases.

2.20 For 1,000,000 items aggregated into 1,000 bundles, most bundles only use one column.

2.21 The sequential placement heuristic places the first 185 items (with the highest demand) optimally, but the cost for the remainder of the items can be as high as more than 6 times the item’s optimal placement.
3.1 Lead time staggering can have a system-wide impact of reducing inventory variation while achieving a higher service level through safety stock. .................................................. 123

3.2 Online-retail inventory replenishment can be modeled as a dynamic program (DP), in which the stage transition probability distribution is obtained by segmenting the review period using lead times as breakpoints. The derivation is similar for the expected stage cost. ........ 133

3.3 Among the three exact DP solution methods described, the hybrid method captures the best of both worlds by combining relative value iteration (RVI) and policy iteration (PI). ......................... 151

3.4 The exact DP methods are computationally prohibitive when the state space is too large. However, the state space may need to be large in order to contain the optimal solution. ....................... 152

3.5 The optimal policy has a structure that resembles the hybrid of base-stock and constant-order policies. ............................ 154

3.6 For base-stock policies of a small numerical example, simulation accurately estimates of the cost function, and the gradient-like information (red arrows) correctly indicates a descent direction of the cost function. 165

3.7 The set of \(\alpha\) policies considered by the target local service level heuristic may not contain the optimal solution. ....................... 174

3.8 For empirical analysis, we construct a realistic numerical example that consists of 15 fulfillment centers and 98 customer regions. The default lead time scenario \(L=\text{west}\) corresponds to having a supplier on the west coast. ................................................................. 179

3.9 The shipping cost function is a piecewise-linear function of distance, fitted according to UPS data for 1-pound packages. .......... 181

3.10 A simple heuristic for comparison, the Newsvendor base-stock policy, is computed by assuming that each region is served by its closest fulfillment center, and that spillovers are not allowed. ............. 181
3.11 Exploratory results suggest that in our case, a reasonable configuration for steady-state simulation is a length of 10,000 periods (after discarding the initial 1,000 periods of transient effects), and a batch size of 100 for the batch means variance estimator. .................................. 184

3.12 The confidence interval for different batch sizes is comparable if the simulation length is much longer than the batch size (in this case, say, over 10,000 periods). .................................................. 185

3.13 Among the three simulation-based optimization methods proposed, random search is the most competitive, producing the best solutions in the shortest time. .................................................. 187

3.14 Compared to base-stock policies, constant-basestock hybrid policies and projected base-stock policies are better able to mitigate fluctuations in order quantities, measured by the coefficient of variation and the relative difference. .................................................. 190

3.15 We compare the scenario of identical lead times of 2 days (labeled \( L=same2 \)) to staggered lead time scenarios (labeled \( L=stag3 \) to \( L=stag6 \)) in which some fulfillment center lead times are longer than 2 days. ........... 191

3.16 Staggered lead time scenarios achieve a better trade-off frontier between stockout and holding costs. .................................................. 192

3.17 Correlations between fulfillment centers can be identified by hierarchical clustering, in which the distance measures are computed from the set of near-optimal replenishment policies. .................................................. 198

3.18 Demand variability, overage, and underage costs have a nonlinear effect on the overall cost. .................................................. 200

A.1 Objective and bound over time for different \( Cuts \) settings (\( |\mathcal{I}| = 10 \), for every combination of problem instances (labeled 0-9) and formulations) 206

A.2 MIP gap over time for different \( Cuts \) settings (\( |\mathcal{I}| = 10 \), for every combination of problem instances (labeled 0-9) and formulations . . . 207
A.3 Objective and bound over time for different formulations ($|\mathcal{I}| = 10$, for 10 instances labeled 0-9) ................................. 208
A.4 MIP gap over time for different formulations ($|\mathcal{I}| = 10$, for 10 instances labeled 0-9) .................................................. 209
A.5 MIP performance on medium-sized problem instances ..................... 210
A.6 MIP performance on large problem instances ............................... 212
A.7 MIP gap at various time intervals for different levels of capacity constraint tightness ($|\mathcal{I}| = 10$, for 10 instances labeled 0-9) .............. 214

B.1 Gradient-based sampling improves the efficiency of R-BEESE. .......... 222
B.2 Detailed empirical results for the target local service level heuristic . . 224
## List of Tables

2.1 Related problems and their features ............................................. 33
2.2 Shorthand labels for MIP formulations ........................................ 56
2.3 Performance of MIP formulations (Average of 10 instances, $|\mathcal{I}| = 10$) . 62
2.4 MIP gap over time for medium- and large-scale MIP formulations (Average of 10 instances) .......................................................... 65
2.5 Initial solution performance for different levels of capacity constraint tightness ($|\mathcal{I}| = 10$, average of 10 instances) ........................................ 69
2.6 Performance of large-scale solution scheme ($|\mathcal{I}| = 1000$, average of 10 instances) .......................................................... 96
2.7 Unit computation time of each stage ($|\mathcal{I}| = 1000$, average of 10 instances) 96
2.8 Number of (RMP) iterations required to reach a certain convergence gap .......................................................... 100
2.9 Initial and final solutions of Stage D ($|\mathcal{I}| = 1000$, average of 10 instances) 103
2.10 Performance of large-scale solution scheme for various levels of capacity constraint tightness ($|\mathcal{I}| = 1000$, average of 10 instances) ............ 110
2.11 Solutions of various fixed cost values ($|\mathcal{I}| = 1000$, average of 10 instances) 110
2.12 Performance of large-scale solution scheme and sequential placement heuristic for an instance of 1 million items ........................................... 114
3.1 Comparison of the optimal policy and Newsvendor base-stock policy . 148
3.2 Optimal policy of the small numerical example ................................. 154
3.3 Staggered lead times results in lower costs ..................................... 154
3.4 Average policy evaluation time ................................................. 188
3.5  Average cost of policies found by random search  . . . . . . . . . . . . . . . 189
3.6  Cost of various lead time scenarios  . . . . . . . . . . . . . . . . . . . . . . . . . . 192

A.1  Optimality gap after each stage, $|\mathcal{I}| = 1000$  . . . . . . . . . . . . . . . . . . . . . 216
A.2  Total computation time for each stage, $|\mathcal{I}| = 1000$  . . . . . . . . . . . . . . . 217
A.3  Number of iterations in Stage C, $|\mathcal{I}| = 1000$  . . . . . . . . . . . . . . . . . . . . 219
A.4  Total computation time breakdown for Stage C, $|\mathcal{I}| = 1000$  . . . . . . . . . 219
Chapter 1

Introduction

1.1 Motivation

Online retail is a business model of rapid growth. In the United States, consumers reportedly make about half of their purchases online,\(^1\) and online retail accounted for 8.1% of total retail sales in 2016, which increased from 7.3% in 2015.\(^2\) Similar trends are observed and forecasted around the world. According to Forrester Research,\(^3\) the online retail market in 2016 grew by 20% in Russia and 18% in Poland; over the next five years, it is expected to grow at an annual rate of 16.6% in Latin America, and 12.3% in Western Europe; but the largest region for online retail sales is Asia Pacific, which contains both the largest market (China) and the fastest-growing market (India).

Effective inventory management is crucial to the success of online retail businesses. To stay competitive, online retailers need to offer customers a wide variety

---


\(^2\)United States Census Bureau, *4th Quarter 2016 Retail E-Commerce Sales Report*. February 17, 2017. https://www.census.gov/retail/index.html. This figure is smaller than the UPS survey because it includes categories that are not generally operated through online retail, such as motor vehicles and gasoline. For a complete list of retail categories, see the Annual or Monthly Retail Trade Report published on the same web page.

\(^3\)Forrester Research, Inc, *Forrester Data Report: Online Retail Forecast, 2016 To 2021*, for the regions of Poland And Russia (published on March 29, 2017), Latin America (April 7, 2017), Western Europe (January 11, 2017), and Asia Pacific (February 8, 2017). Brief summaries publicly available at https://www.forrester.com/.
of items that is kept available at all times, and to ship them to customers within a desired delivery time, while keeping prices low. The quest for variety, availability, and cost-effective shipping, along with many other goals, are achieved through inventory decisions, such as where to put the items, how much inventory to hold, and with which units of inventory to fulfill customer demand.

Online retail inventory is carried in an interdependent network of facilities called fulfillment centers, which are managed jointly in a centralized fashion. When a customer order is received, the online retailer has the flexibility to select the fulfillment center(s) from which to ship the order. To provide a high level of service, customer orders are typically fulfilled as long as it is feasible to ship the items to the customers within the specified delivery time, even if the shipping cost is high. This flexibility to serve customers from different fulfillment centers is distinct from the traditional retail setting where brick-and-mortar stores independently serve customers in their geographical vicinity.

Since the fulfillment centers are interdependent, online-retail inventory management decisions are larger in scale and more complex than problems in traditional retail. This motivates the need for developing new large-scale optimization models and methods that are applicable to this context.

1.2 Thesis scope and related problems

1.2.1 Types of decisions in online-retail inventory management

This thesis addresses two decisions in online retail: inventory placement and inventory replenishment. We now describe the scope of our studies with respect to other online retail inventory management problems, and provide pointers to existing literature. Specific literature review for each problem is given in the later chapters.

We focus on the setting in which the retailer either operates the online channel only, or manages the online channel inventory independently from that of the the
traditional retail channel. The integrated omni-channel setting is beyond the scope of our studies, but it is nonetheless a promising area for extending some of the large-scale solution schemes we develop. For issues in omni-channel retail, see Agatz et al. (2008), Hübner et al. (2016), and Govindarajan et al. (2017).

We categorize inventory management decisions in online retail as strategic, tactical, or operational, based on the planning horizon of the problem. While such categorization can help divide-and-conquer, it is important to recognize that all of these decisions are interconnected and, in practice, often integrated. Below, we briefly review some related work in each category.

**Strategic.** Strategic decisions have a long-term impact, and usually involve substantial capital investments or contractual agreements that cannot easily be changed. These decision include, for example, network design—where to build (or add) fulfillment centers, and with what storage and throughput capacities; and sourcing—what SKUs, or items, to carry, which vendors to buy them from, and the terms of procurement such as replenishment frequency and lead time, minimum order quantities, pallet sizes, etc.

In principle, most network design models developed for other types of supply chains are applicable to online retail, possibly after some adaptation. For a review on network design, see Klose and Drexl (2005a), as well as other references in Section 2.1.1. Agatz et al. (2008) mention drop shipping as a network design (and sourcing) model unique to online retail; this is the option of shipping the product directly from the supplier. In recent years, new types of facilities have also emerged that add complexity to online-retail network design, such as sortation centers where items from different fulfillment centers are consolidated to reduce the number of shipments, lockers where customers pick up their items (rather than have them shipped to their

---

4Stock-keeping units, or SKUs, is a type of item for sale that is distinct from other types of items by its attributes. This may correspond to different products, or different sizes, colors and packaging of the same product.

homes), or facilities dedicated to the processing of returns.

Sourcing problems in online retail are reviewed in Swaminathan and Tayur (2003), with a focus on auctions and supplier alliances that have progressed with online retail. New business models such as “Fulfilled By Amazon” have also blurred the lines between retailer, supplier and platform provider. These advances pose new challenges in the management of inventory, contracts, and many other aspects; see Zhu and Liu (2016) for a recent study.

Tactical. Tactical decisions are typically made more frequently than strategic decisions, say every few months or on a seasonal basis, and are periodically re-evaluated to adapt to the evolving retail landscape. These include placement— in which fulfillment centers to put each of the items; and replenishment— how much inventory to order for each item. These two decisions are the main focus of this thesis.

Regarding inventory placement, Xu (2005) studies the problem of inventory allocation among fulfillment centers for low-demand items, and provides data from a large online retailer to show that low-demand items constitute a large proportion of the catalog. The author characterizes the inventory allocation plan for 2 units of the same item in 2 or 3 fulfillment centers, so as to minimize the expected outbound shipping cost under stochastic demand. We take a different modeling approach by decomposing the problem into a deterministic single-period placement problem and a stochastic multi-period replenishment problem. This keeps the problems tractable, and enables us to account for larger instances, as well as more general conditions such as capacity constraints and fulfillment center fixed costs.

Regarding inventory replenishment, Acimovic (2012) and Acimovic and Graves (2016) study a setup that resembles ours, but are distinct in some important features. A detailed review and comparison is given in Section 3.1.1.

Operational. Operational decisions correspond to the day-to-day deployment of inventory. Such decisions often follow certain pre-determined rules that allow them to be made quickly in real-time. These include fulfillment— which inventory units
to use for fulfilling customer demand; and transshipment—when and how much to transfer inventory between fulfillment centers so as to rebalance the system.

The primary objective of the fulfillment decision is to minimize the outbound shipping cost from fulfillment centers to customer. Acimovic (2012) and Acimovic and Graves (2015) provide literature reviews, and show that the myopic fulfillment policy (i.e., always shipping from the cheapest fulfillment center, which is typically closest to the customer) is not optimal. They propose a heuristic that is tractable and performs well in practice. Xu et al. (2009) focus on minimizing the total number of shipments for multi-item orders, and propose a heuristic for adjusting myopic order assignments (i.e., assignments of customer orders to fulfillment centers) after they have been made in real time. Other notable recent works include Lei et al. (2016), who study the benefit of considering fulfillment and pricing problems jointly; and He and Ke (2017), who study the effect of shipment pricing schemes on inventory allocation.

The idea of lateral transshipment is not unique to online retail; see Paterson et al. (2011) for a review. In online retail, transshipment may be viewed as a strategy to improve fulfillment costs, since orders with multiple items are best shipped in consolidated packages from fewer fulfillment centers, rather than independently from multiple fulfillment centers. See Torabi et al. (2015) for a recent study that uses Benders decomposition to solve the joint fulfillment and transshipment problem.

1.2.2 Assumptions about other decisions

Having reviewed the three decision levels, the remainder of this thesis focuses on the tactical decisions of inventory placement and inventory replenishment. Before proceeding, we state the assumptions we make about the strategic and operational decisions.

For strategic decisions, we limit ourselves to the case in which fulfillment centers are the only type of facilities in the network, and assume that their locations and capacities are known. We also assume that the following information is given as inputs: the assortment of items, as well as each item’s review period, replenishment
lead times, and various cost components including shipping, stockout, holding and fixed costs. While our models and methods are generally applicable to any problem instance that follows these assumptions, we test our work under realistic conditions by using publicly available data to create numerical examples that mimic the operations of a large online retailer in the United States, although our examples are based on publicly available data. In our empirical analysis, we investigate the effects of adjusting fulfillment center capacities, lead times, and various cost components. The results of our work can in turn provide valuable insights that guide the network design and sourcing decisions.

For tactical decisions, we assume that customer orders are served according to the myopic fulfillment policy, and that there is no transshipment between fulfillment centers. Although myopic fulfillment is not optimal, it serves as a simple starting point. More sophisticated fulfillment rules, as well as the integration of transshipment, can both be interesting directions of future work. The simulation-based large-scale solution schemes we develop for the replenishment problem may be adequate in handling such additional complexity.

### 1.2.3 Relationship between placement and replenishment decisions

The placement and replenishment decisions are closely related, and can even be viewed as one integrated decision: Given the shipping, stockout, holding and fixed cost parameters, along with the capacities and lead times of fulfillment centers and the customer demand forecast, how much inventory of each item, if any, should be held and ordered at each fulfillment center? The complexity of this integrated decision prevents it from being solved efficiently. Instead, we decouple the placement and replenishment decision in the following manner:

- The placement decision focuses on the selection of fulfillment centers, and the decision is made concurrently for all items to account for shared capacity. To estimate the cost of selecting a set of fulfillment centers, the shipping cost is es-
timated with a deterministic network flow problem using the expected demand. In contrast, the replenishment decision considers a single item with stochastic demand, for which we assume that there are no capacity constraints. By dealing with capacity constraints in one problem and demand stochasticity in the other, we are able to maintain tractability for both problems.

• We assume that the placement decision is made first, and then the replenishment decision. For each item, the set of fulfillment centers selected by the placement problem is an input for the replenishment decision. The latter may turn out to use fewer fulfillment centers than given by the former.

• The placement problem is a single-period model that represents the length of a few months. The replenishment problem is a multi-period model, in which each period is a review period, typically one or two weeks. In reality, we expect both decisions to be re-evaluated every few months.

• The objective to be minimized in the placement problem is the sum of outbound shipping and fixed costs, while the objective of the replenishment problem is the sum of shipping, stockout and holding costs. Both have the common component of the outbound shipping cost, which is a distinctive feature of online retail. In the former, the shipping cost is estimated with a transportation problem, which helps us focus on the effect of fulfillment center selection. In the latter, the exact shipping cost is calculated according to the myopic fulfillment policy, so as to focus on the interdependence of fulfillment centers. The latter explicitly accounts for the worst case in which demand is shipped from a faraway fulfillment center, which makes it a more accurate approximation of reality. As a result, it will likely report in a higher shipping cost than the former.

• Though beyond the scope of our study, we remark that the output of the replenishment problem can be used as feedback to tune parameters of the placement problem, thereby closing the loop and providing a more accurate model. For example, if the collective replenishment decisions for the items turn out to violate
capacity constraints, we may need to adjust the fixed cost parameters in the placement model so as to provide a different set of fulfillment centers for some items. As another example, the shipping cost computed by the replenishment model could also be used to tune the shipping cost parameters in placement, in order to better reflect reality.

1.3 Thesis outline and summary of contributions

As the title indicates, there are two goals for this thesis: To help online-retailers make better inventory management decisions, and to develop optimization methods that can tackle large-scale problems within and beyond the context of online retail. The goals are achieved through three types of contributions, including models that captures distinctive features of online retail while maintaining tractability, methods for solving mixed-integer programs and dynamic programs at scale, and managerial insights obtained from studying the solutions.

The next two chapters of the thesis study the respective problems of inventory placement and replenishment. The chapters have parallel outlines: Each begins with an introduction and a review of related work. The exact mathematical formulations follow, and we show that they suffer from the curse of dimensionality. We then propose optimization methods that can solve the problems at scale. The effectiveness of these methods are demonstrated by conducting empirical analysis on realistic numerical examples. We also study the effect of some important problem parameters on solution properties. Both chapters conclude with a more detailed summary of contributions, along with directions for future work.

Chapter 2 considers the inventory placement problem, in which the goal is to determine a cost-minimizing inventory placement plan, while respecting demand satisfaction and fulfillment center capacity constraints. The cost is the sum of shipping costs (from fulfillment centers to customers) and fixed costs (for each item’s use of a fulfillment center). The problem can be formulated as a mixed integer program, but it is too large to be solved even with state-of-the-art commercial software. The large
scale is due to the size of the network and the number of items, the latter of which is often on the order of the thousands or even millions.

We propose a large-scale solution scheme that combines the techniques of item aggregation and column generation. The main idea is to consolidate the items into aggregate items, solve an approximate column-based reformulation in the space of aggregate items, and then disaggregate the solution into placement plans for the individual items. The large-scale solution scheme is tested empirically and found to be competitive, compared with a simple heuristic that places items one at a time. The large-scale solution scheme is able to produce near-optimal solutions in a few hours, and the computation can be further accelerated with the use of parallel processors. The solution scheme also provides a quick way of investigating the effects of changing parameters such as fixed costs or capacity constraint tightness.

Chapter 3 considers the inventory replenishment problem, in which the goal is to find a periodic review replenishment policy for a single item that minimizes the expected long-term shipping, stockout and holding costs. We provide an exact characterization of the problem as a dynamic program, under assumptions of a memoryless stochastic demand process, integral replenishment quantities, non-identical fulfillment center lead times, and lost sales. The dynamic program cannot be solved at scale, as the discrete state space is large, and the cost function and inventory dynamics are complex.

We propose two approximation schemes to overcome these issues: Approximation in policy space, which is to focus on parameterized policies, and approximation in value space, which uses simulation to evaluate policies. Combining these approximations, we propose three simulation optimization methods to find low-cost parameterized policies. The most effective of the three is a random search method, which we develop from a state-of-the-art random search by augmenting it with gradient-like information estimate with simulation. The simulation optimization methods and various classes of parameterized policies are tested empirically, and compared with a simple base-stock policy obtained by a Newsvendor model assuming independent fulfillment centers. We find that simulation optimization is able to produce low-cost
solutions within a few minutes, and that there is a large set of such low-cost solutions to choose from. We study these low-cost solutions under various settings, and analyze the effect of lead time staggering, fulfillment center correlations, as well as sensitivity to demand variability and cost factors.

For a quick read, the titles of figures given in the List of Figures are designed to form a succinct summary of the sections (excluding figure titles in appendices).
Chapter 2

Online-retail inventory placement

2.1 Introduction

The topic of this chapter is the inventory placement problem, i.e., the choice of fulfillment centers in which to place each SKU (stock-keeping unit) or item. The goal is to determine a cost-minimizing placement plan for a single planning period that satisfies the expected geographical demand and respects fulfillment center capacity constraints. The objective function includes two types of costs: outbound shipping costs, which are proportional to the amount of goods sent from fulfillment centers to customers, and fixed costs, which reflect the operational overhead for each item held at a fulfillment center.

The online-retail inventory placement problem can be modeled as a mixed integer program with a binary variable for each item-fulfillment center pair. It can be viewed as an extension of the facility location problem and a special case of the multicommodity capacitated fixed-charge network flow problem. However, most existing methods for these problems are not designed to work efficiently with a large number of items, which is typically on the order of thousands or millions in online retail.

We propose a large-scale solution scheme that not only overcomes the large number of items, but in fact leverages it, turning the notorious “curse of dimensionality” into a blessing. Our approach is the integration of two techniques: item aggregation and column generation. Item aggregation reduces the number of variables and also allows
for a column-based reformulation of the problem, while column generation solves reformulated problem efficiently.

In the remainder of this section, we review related work and describe a numerical example that will be used throughout the chapter. Section 2.2 formulates the online-retail inventory placement problem as a mixed-integer program, and provides a numerical comparison of various alternative formulations. Section 2.3 describes our large-scale solution scheme in five steps. Section 2.4 demonstrates the effectiveness of our solution scheme with large-scale numerical experiments. Section 2.5 concludes with future work, including extensions to a class of related problems in online retail inventory management that have sparsity constraints, instead of fixed costs, on fulfillment center usage.

2.1.1 Related work

In this section, we review three strands of relevant literature: related problems, aggregation, and column generation. Our goal is to highlight important features of our problem in comparison with related problems, as well as to show that the unique effect of combining item aggregation and column generation has not been fully explored in previous work.

Related problems

Model features. The online-retail inventory placement problem can be viewed as either a generalization of the facility location problem (also known as the plant location problem), and a special case of the fixed-charge network flow problem. Given a network of nodes and arcs, the facility location problem chooses nodes at which to open facilities, while the fixed-charge network flow problem chooses arcs on which flow is permitted. Both problems have the objective of minimizing the sum of fixed costs and variable costs, the former being determined by the usage of nodes or arcs, and the latter being proportional to the amount of flow on each arc. These problems belong to the class of network design problems, for which there is a rich collection of
literature; see Magnanti and Wong (1984), Minoux (1989), Balakrishnan et al. (1997), Crainic (2000) and Klose and Drexel (2005a) for reviews and pointers to other related problems.

Figure 2.1 illustrates the relationship between a subset of representative problems, in the spectrum from the simplest version of facility location to the most general version of fixed-charge network flow. Table 2.1 summarizes four key features that differentiate the problems shown in Figure 2.1, which include:

- Number of commodities: single-commodity or multicommodity. The former is a special case of the latter.

- Capacity: uncapacitated, node-capacitated, or arc-capacitated. The convention for facility location problems is to define capacity constraints on nodes, while for network flow problems, they are defined on arcs. The latter is more general, because node capacities can be converted to arc capacities. There are two types of node capacity. The first type is a limit on the amount of flow passing through the node, which may represent, for example, the floorspace of a warehouse or the throughput of a production process. This type of node capacity can be converted to arc capacity by duplicating the node into a pair of so-called split nodes and creating a capacitated arc in between. The second type of node capacity is on the amount of flow that can originate from it. For example, the node may model an external supplier that can only provide a limited amount of materials. This type of node capacity can be converted to arc capacity by creating a virtual “supernode” that represents the single source of all flows, and imposing the capacity constraint on the arc directed from the supernode to the node.

- Fixed charge: whether the fixed charge is defined for the usage of nodes, edges, or both. Again, facility location problems typically model fixed charge on nodes, while network flow problems typically model fixed charge on edges. The former is a special case of the latter, due to the same conversion technique of split nodes described above.
• Network structure: two-echelon, multi-echelon, or general networks. The basic version of facility location focuses on bipartite graphs with two echelon of nodes that represent facilities and customers. Multiple echelons of facilities have also been proposed to capture hierarchical structures, which are common in distribution and telecommunication networks. As the network complexity increases, special algorithms that exploit network structure become less applicable, and the line between multi-echelon and general networks is blurred.

**Summary of models.** Below, we briefly describe each problem shown in Figure 2.1 and provide a non-exhaustive list of some previous work, with the intention of highlighting relevant features that provide the context for understanding and comparing our work.

(A) The single-commodity uncapacitated facility location problem is also known as the *simple plant location problem* (SPLP). This problem dates back to the 1960s, and it can be shown to be NP-hard.\(^1\) For a survey of its origins, complexity and solution approaches, see Krarup and Pruzan (1983). For important special cases and extensions, see for example Cornuejols et al. (1977), Revelle and Laporte (1996) and the references therein. Since there are no capacity constraints, it is clear that every customer will be served by the closest facility that is open, which is called the *single-assignment property* in Krarup and Pruzan (1983).

(B) The *multicommodity location problem*, proposed by Warszawski and Peer (1973) and Warszawski (1973), extends the simple plant location problem to the setting of a multi-commodity production system, but imposes the restriction that each

---

\(^1\) The class $NP$ is the set of problems for which verification of a given solution certificate (i.e., a version of the problem with the answer “yes” or “no”) can be done in polynomial time. The class $NP$-**complete** is the set of problems that are in $NP$ and at least as hard as any other problem in $NP$. The class $NP$-**hard** is the set of problems that is at least as hard as any problem in $NP$, but itself may not necessarily be in $NP$. The simple plant location problem is NP-hard, and the decision version of the problem—given an instance of the problem and some threshold $k$, does there exist a solution with cost at most $k$?—can be shown to be NP-complete by a polynomial-time reduction from the set covering problem, which is NP-complete. This reduction can be found, for example, in Krarup and Pruzan (1983). For a formal introduction to NP-completeness, see Chapter 34 of Cormen et al. (2009).
Figure 2.1: Online-retail inventory placement is related to facility location and fixed-charge network flow problems.

(Arrows indicate the direction of generalization.)

Table 2.1: Related problems and their features

<table>
<thead>
<tr>
<th>Problem</th>
<th>Feature</th>
<th>Commodities</th>
<th>Capacity</th>
<th>Fixed charge</th>
<th>Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A) Simple plant location</td>
<td></td>
<td>Single</td>
<td>None</td>
<td>Node</td>
<td>2-echelon</td>
</tr>
<tr>
<td>(B) Multicommodity location</td>
<td></td>
<td>Multiple</td>
<td>None</td>
<td>Node</td>
<td>2-echelon</td>
</tr>
<tr>
<td>(C) Capacitated plant location</td>
<td></td>
<td>Single</td>
<td>Node</td>
<td>Node</td>
<td>2-echelon</td>
</tr>
<tr>
<td>(D) Warehouse location</td>
<td></td>
<td>Multiple</td>
<td>Node</td>
<td>Node</td>
<td>3-echelon</td>
</tr>
<tr>
<td>(E) Capacitated facility location / network design</td>
<td></td>
<td>Single</td>
<td>Node</td>
<td>Node and arc</td>
<td>General</td>
</tr>
<tr>
<td>(F) Multicommodity capacitated fixed-charge network flow</td>
<td></td>
<td>Multiple</td>
<td>Arc</td>
<td>Arc</td>
<td>General</td>
</tr>
</tbody>
</table>

(C) Even with a single commodity, the *capacitated plant location problem* is much more difficult than the uncapacitated version. Sridharan (1995) surveys various techniques for solving the problem, including greedy and interchange heuristics, relaxations, dual ascent method, Bernders decomposition, cross decomposition, and reduction tests. A particularly noteworthy work is Cornuéjols et al. (1991), who analyze several Lagrangian relaxation bounds in terms of both tightness and computational complexity. They provide numerical results to demonstrate the computational efficiency of a heuristic based on Lagrangian relaxation bounds, and study the impact of the tightness of the capacity constraints.

(D) To bridge the gap between the problems of facility location (which models fixed costs on nodes) and network design (which models fixed costs on arcs), Melkote and Daskin (2001a) proposed the *capacitated facility location/network design problem* for a single commodity, which models fixed costs on the use of both nodes and arcs. They propose several valid inequalities for the problem, utilize them in the solution process of the optimization software CPLEX, and study the effect of the tightness of the capacity constraints.

We remark that the problem can be transformed to a capacitated fixed-charge network flow problem, by adding a supernode that is connected to all the nodes and reformulating the fixed costs of nodes as fixed costs of the new arcs. A similar transformation technique is mentioned in Melkote and Daskin (2001b) for the uncapacitated version of this problem.

(E) In the presence of multiple commodities and capacity constraints, previous work focuses on the *warehouse location problem*, which models a three-echelon network structure consisting of plants (or factories), warehouses (or distribution centers) and customers. The facilities to be located are the warehouses. This problem
is motivated by the quest of improving cost or service in a distribution system by adding warehouses between plants and customers. Kuehn and Hamburger (1963) provide a formulation of the warehouse location problem where warehouse capacities are shared by all commodities and production capacities are given for each commodity at each plant. However, their heuristic procedure and test problem (which has since become a widely-used benchmark) are for a single commodity and do not explicitly account for capacity constraints. Geoffrion and Graves (1974) formulate a similar problem that has plant capacity and warehouse throughput constraints, but it requires each customer (who may demand multiple commodities) to be served by a single warehouse, which makes the problem well-suited for Benders decomposition. Pirkul and Jayaraman (1998) propose a more general formulation, which they call PLANWAR (PLANt and WAREhouse), that includes fixed and variable costs, as well as capacity constraints, for all plants and warehouses. They suggest a heuristic solution approach that uses the subgradient method to solve the Lagrangian relaxation of the flow conservation constraints, and generates a feasible solution based on the dual solution at each iteration of the subgradient method.

We note that the online-retail inventory placement problem can be formulated as a special case of PLANWAR, by modeling each fulfillment center as a warehouse, and the supply for each item at each fulfillment center as a plant (i.e., we need as many plants as the number of item-fulfillment center pairs). Based on their numerical results, the proposed heuristic appears unlikely to be scalable to problems of our size.

(F) All the problems above can be modeled as special cases of the multicommodity capacitated fixed-charge network flow (or network design) problem, where capacities, fixed costs and variable costs are all defined on the arcs rather than the nodes. There are two formulations for this problem, arc-based and path-based, each suitable for its own set of practical applications and solution approaches, as described in detail by Gendron et al. (1999). Considerable effort has been
devoted to exact solution approaches for the arc-based formulation, including cutting plane methods, Benders decomposition, and Lagrangian relaxation, as surveyed by Gendron (2011); see also Gendron and Larose (2014) for a review of recent advances. Hewitt et al. (2010) combine exact and heuristic approaches that utilize both arc-based and path-based formulations.

In Section 2.2.1, we show the conversion of our online-retail inventory placement problem to a multicommodity capacitated fixed-charge network flow problem. The special structure of our problem leads to the arc-based and path-based formulations being essentially the same, but the number of nodes and arcs under the fixed-charge network flow formulation has the same order of magnitude as the number of commodities, which makes our problem larger than all of the numerical examples in the work above.

In addition, there are many extensions to the models above that capture other practical considerations, including multi-period dynamics (e.g., Canel et al. (2001), Melo et al. (2006) and Owen and Daskin (1998)), stochasticity of demand or costs (e.g., Owen and Daskin (1998) and Snyder (2006)), nonlinear cost structures (such as concave or piecewise linear, e.g., Kuehn and Hamburger (1963), Balakrishnan and Graves (1989)), etc. Melo et al. (2009) provide a more comprehensive review of facility location problem features that are important for supply chain management, and Geoffrion and Graves (1974) describe eight types of analyses that guide the implementation of a network design solution in practice. While beyond the scope of this text, it is helpful to keep these aspects in mind as we move towards applying our work in real life.

**Summary of solution methods.** We now provide a synthesis of solution methods that have been proposed for the aforementioned problems, along with some representative work. These solution methods can be roughly categorized as (1) variants of branch-and-bound, (2) Benders decomposition, and (3) heuristics. The first two methods are “exact” in the sense that their goal is to solve the mixed-integer program to optimality. In contrast, heuristic methods seek to obtain “good” solutions
efficiently, with or without guarantees on solution quality.

The class of exact methods based on the branch-and-bound procedure has been widely studied and relies on two techniques. The first technique is relaxation, which includes linear programming (LP) relaxation, i.e., relaxing integer variables into continuous variables); Lagrangian relaxation, i.e., dualizing the constraints and adding them to the objective; and less commonly, relaxation simply by dropping constraints. These relaxations all provide lower bounds on the objective cost. The second technique is to make use of valid inequalities, which are redundant constraints that give an equivalent definition of the problem, but whose relaxation may lead to an improved lower bound. For a sample of valid inequalities and relaxation methods, see Cornuéjols et al. (1991) and Klose and Drexl (2005b) for the capacitated facility location problem, and Gendron et al. (1999) and Gendron (2011) for the multicommodity capacitated fixed-charge network flow problem.

Different combinations and applications of the two techniques lead to different variants of branch-and-bound methods. For example,

- The basic version of branch-and-bound begins with all the integer variables being relaxed, and searches different branches of the solution space by fixing an integer variable at different values. A relaxation is solved at each node to obtain a lower bound of the branch, and if the lower bound is greater than the value of the best feasible solution found so far, the branch does not contain the optimal solution and will thus be pruned from the search tree. Therefore, relaxations that give tighter lower bounds lead to a more efficient branch-and-bound process, although the bounds may also be more computationally intensive to compute.

- If only the LP relaxation is utilized, then the lower bound can be obtained by solving the dual of the LP relaxation, which may be much easier than solving the LP relaxation itself. This is the idea behind the dual ascent method, e.g., Guignard and Spielberg (1979) and Gendron (2002), which in general does not appear to be as competitive as other relaxation methods.
• If valid inequalities are identified only when they are violated, and added as cutting planes, we have the branch-and-cut method; e.g., Chouman et al. (2016)). If there is a large number of valid inequalities and flow variables, and both are identified and added dynamically, the method is called branch-and-price-and-cut; e.g., Gendron and Larose (2014).

• If Lagrangian relaxation is used as a lower bound in the branch-and-bound process, there are various ways of optimizing the dual variables, such as the subgradient method, e.g., Pirkul and Jayaraman (1998) and Holmberg and Yuan (2000), or the bundle method, e.g., Crainic et al. (2001). If, moreover, new valid inequalities are added in the process of solving the Lagrangian dual, the method is referred to as relax-and-cut; see Kliwer and Timajev (2005).

Another class of exact methods is Benders decomposition, which is based on the observation that for given values for the binary (usage) variables, the cost corresponding to continuous (flow) variables is relatively easy to optimize, especially in the case where it is possible to further decouple the continuous variables into smaller problems. The dual of the continuous variable cost optimization problem provides a way to express that cost in terms of the binary variables. Therefore, we can formulate a so-called master problem that contains only the binary variables, and iteratively update the dual expression by solving the cost optimization problem for continuous variables.

Geoffrion and Graves (1974) reports the successful application of Benders decomposition to the multicommodity warehouse location problem, which only requires a few iterations due to the structure of the problem. Van Roy (1986) uses Lagrangian relaxation to speed up Benders decomposition, a technique called cross decomposition, to solve the single-commodity capacitated facility location problem. Benders decomposition has also been applied to the fixed-charge network flow problem, for example by Costa et al. (2012) (see also Costa (2005) for a survey). However, Gendron (2011) points out that Benders decomposition appears to be computationally inferior than branch-and-cut methods (since capacity constraints prevent further de-
composition of the continuous variables), and that, nevertheless, the formulation of Benders decomposition may be useful for cut-generation.

Heuristics come in a variety of flavors. For example, for the single-commodity capacitated facility location problem, some early approaches include greedy heuristics, i.e., iteratively adding or dropping the set of nodes or arcs in use, as well as interchange heuristics, which attempts to make improvements at the end of greedy procedures; see Jacobsen (1983) and Sridharan (1995) for more details. Another type of heuristics, as suggested by Cornuéjols et al. (1991), is based on directly solving the aforementioned Lagrangian relaxation with an iterative method, such as the subgradient method, and generating a feasible solution in each iteration.

For the multicommodity fixed-charge network flow problem, many heuristics have also been developed recently, especially with the rise of large-scale applications. Examples include but are not limited to: tabu search methods of Crainic et al. (2000) (simplex-based), Crainic and Gendreau (2002) (parallelized), Ghamlouche et al. (2003) and Ghamlouche et al. (2004) (cycle-based); the capacity scaling heuristic of Katayama et al. (2009); the local branching heuristic of Rodríguez-Martín and Salazar-González (2010); and the metaheuristic approach of Hewitt et al. (2010), which employs a local search heuristic on the arc-based formulation to construct primal solutions, and applies cuts found in the search process to strengthen relaxations for the path-based formulation and generate tight bounds, which in turn informs the search of primal solutions.

In summary, many solution approaches have been developed for the large collection of related problems. However, to the best of our knowledge, there are no existing methods specially designed to exploit the structure of the online-retail inventory placement problem, or to address its large-scale nature especially in the number of commodities, both of which are our contributions.

Aggregation

In our proposed large-scale solution scheme, item aggregation contributes to the efficiency in two ways. First, it reduces the dimensionality of the placement problem
by making the bulk of the decisions through an optimization problem for aggregate items. Secondly, it enables the optimization problem for aggregate items to be relaxed and solved as a linear program, by providing an interpretation for fractional solutions. Existing literature focuses primarily on the first benefit of item aggregation.

**Production planning.** In the context of production planning, Hax and Meal (1975) introduce the framework of *hierarchical production planning* that relies on the idea of item aggregation. In their framework, there are three levels of aggregation: *items, families,* and *types.* Items with a common production setup are aggregated as a *family,* and are scheduled consecutively to save on setup costs. Families with similar seasonal demand patterns and production rates (i.e., inventory investment produced per unit time) are aggregated as a *type,* for which multi-period capacity planning decisions are made first. For every period (usually only the immediate period is of interest), the production capacity is disaggregated into production schedules for its families, which is further disaggregated into detailed schedules for each item of a family.

This hierarchical decision-making process is easier to compute and manage, but its sequential nature may also result in the final solution being infeasible or suboptimal. Several remedies have been proposed, including the “look ahead feasibility rule” of Bitran et al. (1981) to prevent infeasibility, and the Lagrangian approach of Graves (1982), which can be viewed as an iterative feedback mechanism between the two hierarchical decisions of capacity planning and production scheduling.

For a detailed description of the hierarchical production planning framework, see Bitran and Tirupati (1993), which includes discussions on the benefits, various methods for aggregation and disaggregation, as well as extensions to multi-stage and stochastic settings.

In comparison with hierarchical production planning, our work also has the goal of allocating capacity across aggregate items. However, we are concerned with a single-period problem rather than a multi-period scheduling problem, and our cost and network structure is very different from the setting in production planning.
Mathematical programming. In the context of mathematical programming, aggregation and disaggregation have been widely recognized as a technique for reducing computational complexity, and are sometimes called *surrogate programming* because aggregation produces surrogate models. This line of work originates from large-scale transportation problems and dates back to Balas (1965) and Lee (1975). A detailed summary and comparison of important early work is given by Francis (1985). For a comprehensive survey, see Rogers et al. (1991), and also the recent textbook Litvinchev and Tsurbkov (2013).

Much of the research work focuses on bounding the error or loss as a result of aggregation. Geoffrion (1976) provides bounds for customer aggregation in a single-commodity, capacitated distribution network model that includes the classical transportation problem (i.e., bipartite min-cost network flow) and facility location problem as special cases. For general linear programming problems, Zipkin (1977) derives bounds on the objective cost between the original and aggregate problems; see also Zipkin (1980b) for models obtained by aggregating columns (variables), and Zipkin (1980a) for row-aggregation, both of which use the *fixed-weight aggregation method*, i.e., replace a collection of columns (rows) with their convex combination.

An algorithmic framework that is often used with aggregation and disaggregation is the idea of *iterative aggregation and disaggregation*. In this framework, the model is not aggregated or disaggregated in a single step, but iteratively, until a solution of desired quality is produced. Dudkin et al. (1987) provide a detailed review. See also Shetty and Taylor (1987), who derive error bounds that can be used as the termination criteria. More recently, Bärmann et al. (2015) (see also Bärmann 2016) use iterative aggregation to solve a railway expansion problem in network design, and relates their formulation to Benders decomposition.

Most of the application to network flow problems performs aggregation on nodes in the network, with the exception of Geoffrion (1977), who studies the aggregation of commodities (or items) in a three-echelon network where items flow from procurement zones (or suppliers) through facilities to customers. Under the assumption that procurement zones have capacity constraints that are given for aggregate items,
which is argued to be a more manageable policy in practice, the author proposes a
method for aggregating item flow variables and derives a bound between the cost of
the original problem and the problem with aggregated flow variables. While we also
focus on item aggregation, the intent and setup of this model is distinct from our
work.

Aggregation in integer programming problems has been relatively under-explored,
and our work can be thought of as contributing to this area. On the aggregation
of constraints, Glover (1968) studies the strength of surrogate constraints that are
implied by a set of constraints; Bradley (1971) shows that any bounded integer pro-
gramming problem with equality constraints can be transformed to an equivalent
problem with a single aggregate constraint and the same number of integer variables;
Rosenberg (1974) provides further results on the existence of such aggregate equality
constraints; and Chvátal and Hammer (1977) propose an algorithm that determines
the existence of an aggregate inequality constraints. On the aggregation of integer
variables (i.e., columns), Hallefjord and Storøy (1990) derive bounds on the original
integer program by solving a relaxed problem in which the binary variables are ag-
gregated. Their work bears a similar spirit as ours in their statement, “If sufficiently
many columns are aggregated, relaxation of the integer requirements is justified.”
Their problem consists of only binary variables, and aggregation is performed by re-
placing subsets of binary variables with their sum. In contrast, our problem setting
and solution framework are far more complex: we apply aggregation on our original
problem, which consists of both continuous and binary variables, and then reformulate
the problem into column form, on which relaxation is applied.

Column generation

Column generation has been applied extensively to the path-based formulation of
multicommodity network flow, where columns correspond to the large number of al-
ternative paths. Examples include Barnhart et al. (1994) for the case without fixed
charge; Barnhart et al. (2000) and Holmberg and Yuan (2003) who study multicom-
modity network flow in the presence of additional path-based constraints; and more
recent heuristics for the fixed-charge case such as Crainic et al. (2000) and Katayama et al. (2009).

For the arc-based formulation, Gendron and Larose (2014) present a large-scale branch-and-price-and-cut method that uses column generation to progressively expand the set of active flow variables, which is combined with row generation for cutting planes.

Our definition of columns is closest to that of Klose and Drexl (2005b), in which columns correspond to integer solutions of the capacitated facility location problem. They use column generation to find integer solutions of various relaxations, thereby producing lower bounds. They also compute upper bounds by constructing feasible solutions by applying a rounding heuristic to the solution of the master problem, which is a linear combination of columns and therefore contains continuous values for integer variables. In contrast, a linear combination of columns is readily interpretable in our solution scheme since it represents the solution of aggregate items, and we construct feasible solutions from these columns through optimized disaggregation of items.

2.1.2 Setup of numerical experiments

To test the practicality of our work, we conduct experiments on a series of numerical examples modeled after realistic online-retail inventory systems. In our empirical analysis of numerical experiment results, we measure the performance of a solution method by the trade-off it achieves between computational resources and solution quality. Computational resources include time, memory, and the number of processors (CPUs). Since memory is relatively cheap, we focus on computation time assuming a single processor, and highlight potential improvements from parallel computing on multiple processors whenever applicable. Solution quality is quantified by both the objective cost of the best feasible solution found, as well as the best lower bound, since we have a cost-minimization problem.

Unless otherwise stated, all numerical experiments in this chapter are conducted on machines with a single processor of 2.0 GHz CPU each, 32 GB RAM, and running
the CentOS version 6 operating system. Our test programs are written and executed in Python, version 2.7.5, and its interface to Gurobi Optimizer, version 7.0.1.

The remainder of this section provides details on the setup of numerical example instances that are used in this chapter. We begin by describing the network of fulfillment centers and customer regions, which we created based on publicly available datasets. Then, we present the process for generating items independently at random, which allows for controlled studies that yield both quantitative and qualitative insights. Throughout the chapter, we use subscripts for indices of fulfillment centers and regions, and superscripts for indices of items.

**Online retail network**

**Fulfillment centers.** We use Amazon’s fulfillment center network in the United States, taken from an unofficial public dataset.\(^2\) As of December 2015, there are \(N = 88\) fulfillment centers, indexed by \(n = 1, \ldots, N\), for which we extract the following attributes:

- **Name:** We use Amazon’s names for the fulfillment centers, which are given by the three-letter code of the nearest airport, followed by a single-digit number reflecting the chronological order of its opening.

- **Location:** We convert the physical address into longitude and latitude coordinates by looking up the ZIP code in a table provided as part of 2010 US Census data.\(^3\)

- **Capacity \(b_n\):** The throughput capacity that fulfillment center \(n\) is able to serve in the planning period, e.g., units of items per week.\(^4\) To compute \(b_n\), we first calculate the total capacity by adding a 10% slack to the total demand across

---


\(^3\)United States Census Bureau, 2010 Census Gazetteer Files: ZIP Code Tabulation Areas. \url{http://www.census.gov/geo/maps-data/data/gazetteer2010.html}. Accessed in December 2015. For most areas, the ZIP Code Tabulation Area (ZCTA) is the same as the ZIP code. (For more details on how ZCTAs are created, see \url{https://www.census.gov/geo/reference/zctas.html}.)

\(^4\)Other types of capacities, such as storage capacity, can be modeled with some modification, as described in Section 2.2.1.
all the items (i.e., we assume that $\sum_{n=1}^{N} b_n$ is 110% of the total demand), and then distribute the total capacity to each fulfillment center in proportion to its square footage, which is given by the public dataset. In some experiments, we vary the amount of slack to study the effect of capacity constraint tightness.

Although the dataset also provides partial information on the types of items that some fulfillment centers are dedicated to (e.g., sortable, non-sortable, large, small, apparel, etc), we make no such distinctions in our example, and simply allow every fulfillment center to carry all kinds of items.

**Customer regions.** The customer regions are created from 2010 US Census data. The raw data contains 3221 counties, which we group into $R = 98$ regions, indexed by $r = 1, \ldots, R$. We compute the following attributes for each region $r$:

- **Population $p_r$:** Directly taken from census data.
- **Location:** The weighted average of longitude and latitude coordinates of the counties in the region, weighted by population.
- **Relative longitude $\gamma_r$:** A scalar in the [0,1]-interval given by

$$\gamma_r = \frac{m_r - m}{m - \bar{m}}, \quad (2.1)$$

where $m_r$ is the longitude of region $r$, and the minimum and maximum longitudes of the regions are given by $\bar{m} = -122.4682$ and $\bar{m} = -69.7692$, respectively. In other words, the value of $\gamma_r$ is closer to 0 for regions farther west.

---


6We group counties into regions by dropping the last two digits of the counties’ GEOID, or Geographic Identifiers, which are numerical codes that uniquely identify geographic areas for census data purposes. (For more details on how GEOIDs are created, see [https://www.census.gov/geo/reference/geoidentifiers.html](https://www.census.gov/geo/reference/geoidentifiers.html).) While GEOIDs are generally assigned by geographical location, we remark that dropping the last two digits of the GEOID does not necessarily produce groups of counties that are contiguous with each other, but merely serves as a proxy. This grouping produces 102 regions (i.e., groups of counties), from which we remove the six that are not in continental US (one from Hawaii, two from Puerto Rico, and three from Alaska) and divide the greater Los Angeles counties (which has more than twice the population of the second largest region) into three regions (Los Angeles county, counties east of LA, and counties west of LA), resulting in 98 final regions.
Figure 2.2: For empirical analysis, we construct a realistic numerical example with 88 fulfillment centers (based on Amazon’s network) and 98 regions (based on US census data).

The red triangles indicate the locations of fulfillment centers, each with a size proportional to its square footage. The gray dots indicate the location of customer regions, each with a size proportional to the population.

closer to 1 for regions farther east. It is defined for the purpose of generating item demand distributions, as described in the next subsection.

Shipping costs. The shipping cost structure is modeled after the two-day shipping services of UPS. We assume that for a given UPS shipment mode (e.g., ground or two-day air), the shipping cost $UPS(w, l)$ is a function of two parameters: the weight $w$ of the package, and the distance $l$ on the sphere of the Earth, which can be calculated from longitude and latitude values using the haversine formula.

To capture the nonlinearity of shipping costs as a function of distance, we assume that ground shipment is used for distances no greater than 1,000 miles; otherwise, second-day air shipment is used. The shipment mode threshold of 1,000 miles is determined by observing the two-day radius on a limited number of UPS ground

\footnote{United Parcel Service of America, Inc. \url{https://www.ups.com/}.}
We then fit the shipping cost function of each shipment mode according to the 2016 UPS Daily Rates data,\(^9\) which provides a table of shipping rates for each pair of package weight and shipment zones.\(^{10}\) In the table, each row corresponds to a package weight, and each column corresponds to a zone. We take the first 35 rows for packages weighing 1 to 35 pounds, and the 7 zones that represent shipments within continental US. We align the smallest and largest zone labels with a distance of 0 and 3,000 miles, respectively; i.e., we assume that the first zone corresponds to a distance of 0, the second zone a distance of 500 miles, and so on. The shipping cost table is then fitted with the function

\[
UPS(w, l) = a_3wl + a_2l + a_1w + a_0,
\]

where \(w\) is the package weight, ranging from 1 to 35 pounds, \(l\) is the distance, given in thousands of miles, and \(a_3, a_2, a_1\) and \(a_0\) are scalars. The fitted shipping cost functions are given by

\[
UPS_{\text{Ground}}(w, l) = 0.247847wl - 0.227878l + 0.132063w + 7.553833,
\]
\[
UPS_{\text{2DayAir}}(w, l) = 1.37674wl + 4.02487l + 0.75604w + 11.10890,
\]

with \(R^2 = 0.9735\) for \(UPS_{\text{Ground}}(w, l)\) and \(R^2 = 0.9682\) for \(UPS_{\text{2DayAir}}(w, l)\).

In summary, for an item \(i\) of weight \(w_i\), a fulfillment center \(n\) and a customer

---


\(^{10}\)The zone of a shipment can be calculated with an online tool on the UPS website, and is correlated to the physical distance between the origin and the destination. Shipments within the 48 contiguous states of continental US is divided into 7 zones, labeled 2 to 8 for ground shipping, and 202-208 for second-day air shipping. The shipping cost roughly has an affine relationship with the zone labels.
Figure 2.3: The shipping cost is a piecewise-linear function of distance, fitted from UPS data.

region \( r \) whose distance is given by \( l_{nr} \), the shipping cost is

\[
C^i_{nr} = \begin{cases} 
UPS_{Ground}(w^i, l_{nr}), & \text{if } l_{nr} \leq 1000, \\
UPS_{2DayAir}(w^i, l_{nr}), & \text{otherwise.}
\end{cases}
\] (2.2)

This shipping cost function is illustrated in Figure 2.3 for a sample of \( w^i \).

Random item generation

Let \( I \) denote the set of items in an instance of our numerical example. The online-retail inventory placement problem requires the following item characteristics as inputs for every \( i \in I, \ n = 1, \ldots, N, \ r = 1, \ldots, R \), all of which are non-negative scalars by definition:

- \( d^i_r \), the expected regional demand of item \( i \) by customers in region \( r \) in the planning period of, say, one month.

- \( c^i_{nr} \), the variable cost of fulfilling a unit of item \( i \) with fulfillment center \( n \) for customer region \( r \). This could include the outbound shipping cost, as well as the handling cost or the inbound replenishment cost of the item at the fulfillment center. In our experiments, we focus on the shipping cost, and assume that the other costs are constant or insignificant.
• $f^i_n$, the fixed cost of carrying item $i$ in fulfillment center $n$.

For simplicity, we assume that the fixed cost is identical for all items and all fulfillment centers, and is given by $f^i_n = 1$ unless otherwise stated. In some experiments, we also explore the effect of varying the value of $f^i_n$.

The characteristics $d^i_r$ and $c^i_m$ are generated independently at random for each item. This is done by sampling the weight $w^i$, total demand $d^i$, and geographical distribution parameter $\alpha^i$, for each item $i \in I$, in the following manner:

• The weight $w^i$ is drawn uniformly at random from the interval $[1, 35]$. We then calculate the shipping cost according to (2.2).

• The total demand $d^i$ is drawn uniformly at random from the interval $[1, 100]$. It represents the expected units of customer demand in the planning period totaled across all regions. (Recall that $\sum_{i \in I} d^i$ is also used for computing fulfillment center capacities $b_n$.)

• The geographical distribution parameter $\alpha^i$ is drawn uniformly at random from the interval $[0,1]$. We assume that the regional demand $d^i_r$ relates to $\alpha^i$ by

$$d^i_r := d^i \cdot \frac{p_r a_r}{\sum_{r=1}^{R} p_r a_r}$$

for every $r = 1, \ldots, R$, where $a_r$ is given by

$$a_r := \alpha^i \gamma_r + (1 - \alpha^i)(1 - \gamma_r),$$

and where $p_r$ is the population of region $r$, and $\gamma_r$ is its relative longitude, as defined in (2.1).

The scalar $a_r \in [0, 1]$ is close to $\gamma_r$ when $\alpha^i$ is large, and close to $1 - \gamma_r$ when $\alpha^i$ is small. Recall that the relative longitude $\gamma_r$ ranges from 0, for the westernmost region, to 1, for the easternmost region. Therefore, $\alpha^i = 0$ corresponds to a regional demand pattern where demand is greater in regions with higher populations and are closer to the west coast. Similarly, $\alpha^i = 1$ corresponds to a regional demand pattern that
is positively biased by population and proximity to the east coast. When $\alpha^i = 0.5$, we have $a_r = 0.5$ for all $r$, and the regional demand is entirely proportional to the population. Thus, by sampling $\alpha^i$ at random, we capture demand patterns that are biased, at a range of levels, toward either coast. Of course, in reality, items have a more complex variety of regional demand patterns; nevertheless, the parameter $\alpha^i$ allows us to demonstrate, to some extent, that our solution method is capable of handling a large set of items with different regional demand patterns.

In summary, for a numerical experiment with a specified number of items $|\mathcal{I}|$, we generate the items by independently sampling the three parameters $(w^i, d^i, \alpha^i)$. We then convert these parameters into characteristics of each item, namely, the regional demand $d^i_r$ and shipping cost $c^i_{nr}$. These characteristics serve as inputs to our model, along with a given identical fixed cost $f^i_n$.

## 2.2 Mixed integer program formulations

In this section, we describe the online-retail inventory placement problem in the form of a mixed-integer program (MIP). We compare several MIP formulations which are equivalent in the sense that they have the same optimal solution. We show empirically that state-of-the-art commercial solvers may handle some formulations more efficiently than others, but in spite of that, solving the MIP at scale is computationally prohibitive in terms of the required time and memory.

### 2.2.1 Problem statement

Consider an online-retail inventory system with $N$ fulfillment centers and $R$ customer regions, represented by nodes in a graph and indexed respectively by $n = 1, \ldots, N$ and $r = 1, \ldots, R$. Let $\mathcal{I}$ be the set of items sold by the online retailer, and let the
items be indexed by $i \in \mathcal{I}$. The \textit{online-retail inventory placement problem} is given by

\[
\begin{align*}
\min_{x,y} & \quad \sum_{i=1}^{N} \sum_{n=1}^{R} \sum_{r=1}^{R} c_{nr}^i d_r^i x_{nr}^i + \sum_{i=1}^{N} \sum_{n=1}^{R} f_n^i y_n^i \\
\text{subject to} & \quad \sum_{n=1}^{N} x_{nr}^i = 1 & \forall r = 1, \ldots, R, \forall i \in \mathcal{I}, \\
& \sum_{i=1}^{R} \sum_{r=1}^{R} d_r^i x_{nr}^i \leq b_n & \forall n = 1, \ldots, N, \\
& \sum_{r=1}^{R} d_r^i x_{nr}^i \leq d_r^i y_n^i & \forall n = 1, \ldots, N, \forall i \in \mathcal{I}, \\
& x_{nr}^i \geq 0 & \forall n = 1, \ldots, N, \forall i \in \mathcal{I}, \\
& y_n^i \in \{0, 1\} & \forall n = 1, \ldots, N, \forall i \in \mathcal{I},
\end{align*}
\]  

(MIP)

where, for every $n, r$ and $i$, we have the following variables and parameters:

- $x_{nr}^i$ is a decision variable that indicates the fraction of item $i$ demand in region $r$ that is served by fulfillment center $n$.

- $y_n^i$ is a binary decision variable that indicates whether ($y_n^i = 1$) or not ($y_n^i = 0$) fulfillment center $n$ carries item $i$.

- $b_n$ is the throughout capacity of fulfillment center $n$ for the entire planning period.

- $c_{nr}^i$, the cost of fulfilling one unit of item $i$ demand in region $r$ by shipping it from fulfillment center $n$. We focus on the outbound shipping cost in our case, but remark that this parameter could also include the labor cost of handling the item at the fulfillment center, as well as variable costs incurred by inbound replenishment.

- $f_n^i$ is the fixed cost of carrying item $i$ in fulfillment center $n$ for the planning period. This could represent actual costs incurred by the item’s use of the fulfillment center, such as a flat-rate inbound replenishment cost charged by the vendor. It could also be a virtual device that is used to prevent the item
from being placed in too many fulfillment centers, i.e., to induce sparsity in the placement plan.

- $d_i^r$ is the demand of item $i$ in region $r$ during the planning period.
- $d_i = \sum_{r=1}^{R} d_i^r$ is the total demand across all regions.

The objective function is the sum of shipping and fixed costs over the entire planning period, for example, over one month. (2.3) is the set of demand satisfaction constraints; (2.4) is the set of capacity constraints; and the inequalities (2.5), which we call usage constraints, ensures that for every given $i$ and $n$, we have $y_{in}^i = 1$ if $x_{in}^i > 0$ for any $r$.

While (2.4) is most easily understood as throughput capacity constraints, it can also represent other types of constraints, as long as the capacity $b_n$, demand $d_i^r$ and shipping cost $c_{nr}^i$ are all given in consistent units. For example, if we wish to express storage capacity constraints with (2.4), we can let $b_n$ be the available of storage space at a fulfillment center, $d_i^r$ be the storage space required for the amount of items demanded by region $r$, and $c_{nr}^i$ be the shipping cost in dollars per unit storage space of the item.

In addition, we remark that our model and solution framework can be generalized to incorporate multiple types of linear constraints of $x_{nr}^i$. For example, if we wish to capture storage capacity constraints while keeping (2.4) as throughput capacity constraints, we can add the following expressions,

$$\sum_{i \in I} \sum_{r=1}^{R} v_i^r d_i^r x_{nr}^i \leq v_n \quad \forall n = 1, \ldots, N,$$

where the input parameter $v_i^r$ is the storage space required per unit of item $i$ demand, and $v_n$ is the available storage space at fulfillment center $n$. This formulation is consistent with the common practice of expressing the inventory requirement as a multiple of the demand rate, through the definition of inventory turnover. For example, if the planning period is one month and we require an inventory turnover of 6 times per year, i.e., we need to hold two months of inventory, then we can set $v_i^r$ to be 2 times
the unit storage volume of the inventory.

**Formulation as multi-commodity capacitated fixed-charge network flow**

While (MIP) can be thought of as a special case of the multicommodity fixed-charge network flow problem, it is worth pointing out two features of this equivalence that clarifies the scale of our problem. First, every item has a unique shipping cost between a given fulfillment center and customer region pair \((n, r)\), which is typically determined by its weight. The shipping cost is not proportional to the capacity it takes up in fulfillment centers; the latter may be a different function of other item characteristics. Therefore, the shipment of different items from \(n\) to \(r\) cannot be combined into one flow with a unified shipping cost, but must remain distinct for each item. Secondly, the fixed cost of a fulfillment center is charged for the inclusion of every item, not for their combined usage. It reflects the operational overhead of processing multiple items at a fulfillment center, which is slightly different than the more common notion of fixed cost as the cost of building a facility to handle all commodities in network design problems.

Figure 2.4 depicts our problem as a multicommodity fixed-charge network flow problem. By convention, such problems have a single pair of origin-destination nodes per commodity, no parallel arcs, and each arc is associated with a fixed cost, a variable cost and a capacity. The resulting graph has \(|\mathcal{I}|\) commodities, \(N|\mathcal{I}|\) uncapacitated arcs with only fixed costs and no variable costs (representing the flow of each item to each fulfillment center), \(NR|I|\) uncapacitated arcs with only variable costs and no fixed costs (representing shipping from fulfillment centers to regions for each item), \(N\) cost-free capacitated arcs representing fulfillment centers, and \(R|\mathcal{I}|\) cost-free capacitated arcs representing the region-item pairs.

In this formulation, the nodes and arcs are both on the order of \(|\mathcal{I}|\). This is in contrast with the traditional multi-commodity network flow setting where the number of nodes and arcs is independent from the number of commodities. This puts our problem at a much larger scale than most numerical examples given in previous work.

53
2.2.2 Alternative formulations

There are many ways of modeling the statement of “\( y_{nr}^i = 1 \) if \( x_{nr}^i > 0 \) for some \( r \),” for every fulfillment center and item pair \((n,r)\). The set of constraints (2.5) in the formulation above describes the relationship with expressions of capacity usage \( \sum_{r=1}^{R} d_r^i x_{nr}^i \) and maximum capacity usage \( d_i^f \) of the item at fulfillment center.

An alternative formulation is to replace (2.5) with the set of constraints

\[
 x_{nr}^i \leq y_{nr}^i \quad \forall n = 1, \ldots, N, \forall r = 1, \ldots, R, \quad \forall i \in I. \tag{2.6}
\]

These are often referred to as strong inequalities in network flow literature due to the stronger linear programming (LP) relaxation bounds they yield (i.e., (2.6) implies (2.5), and therefore the polyhedral set of feasible solutions obtained by relaxing the formulation with (2.6) is a subset of that with (2.5).) However, there is a large number of strong inequalities, which may be computationally cumbersome, and the resulting formulation is often highly degenerate (i.e., most of the strong inequalities are not tight at the optimal solution).

Yet another formulation is the well-known big-\( M \) technique of modeling “or” state-
ments,

$$\sum_{r=1}^{R} x_{nr}^i \leq R y_n^i \quad \forall n = 1, \ldots, N, \forall i \in I,$$  

(2.7)

which relies on the fact that each $x_{nr}^i$ is in the [0,1]-interval. As is the case with (2.5), this set is weaker but has fewer constraints than (2.6).

We can also express our conditions using another class of constraints called special ordered sets (SOS),

$$SOS1(x_{nr}^i, 1 - y_n^i), \quad \forall n = 1, \ldots, N, \forall r = 1, \ldots, R, \forall i \in I,$$  

(2.8)

or alternatively,

$$SOS1 \left( \sum_{r=1}^{R} x_{nr}^i, 1 - y_n^i \right), \quad \forall n = 1, \ldots, N, \forall i \in I,$$  

(2.9)

where $SOS1(\cdots)$ is the constraint that at most one of the arguments can take on a positive value. Many modern commercial solvers have the built-in capability of handling SOS constraints effectively (see Bertsimas et al. (2016) for a recent example.)

In addition to variations in usage constraint formulations, we can also change the meaning of each decision variable $x_{nr}^i$. Instead of representing the fraction of demand $d_r^i$ served by fulfillment center $n$, the variable $x_{nr}^i$ can also directly represent the flow
of demand. This leads to a slightly different formulation:

\[
\begin{align*}
\min_{x,y} & \quad \sum_{i=1}^{N} \sum_{r=1}^{R} c_{i}^{r} x_{i}^{r} + \sum_{i=1}^{N} f_{i}^{r} y_{i}^{r} \\
\text{subject to} & \quad \sum_{n=1}^{N} x_{i}^{r} = d_{i}^{r} \quad \forall r = 1, \ldots, R, \forall i \in I, \\
& \quad \sum_{i \in I} x_{i}^{r} \leq b_{n} \quad \forall n = 1, \ldots, N, \\
& \quad \sum_{r=1}^{R} x_{i}^{r} \leq d_{i}^{r} y_{i}^{r} \quad \forall n = 1, \ldots, N, \forall i \in I, \\
& \quad x_{i}^{r} \geq 0 \quad \forall n = 1, \ldots, N, \forall r = 1, \ldots, R, \forall i \in I, \\
& \quad y_{i}^{r} \in \{0, 1\} \quad \forall n = 1, \ldots, N, \forall i \in I.
\end{align*}
\] (MIP2)

Note that in this case, (2.10) represents both the capacity usage formulation and the big-M formulation. As before, we also have the alternatives of strong inequalities

\[
x_{i}^{r} \leq d_{i}^{r} y_{i}^{r} \quad \forall n = 1, \ldots, N, \forall r = 1, \ldots, R, \forall i \in I,
\] (2.11)

as well as the SOS constraints (2.8) and (2.9).

Table 2.2 summarizes the nine formulations described thus far, with shorthand labels assigned to each for ease of future referencing.

Table 2.2: Shorthand labels for MIP formulations

<table>
<thead>
<tr>
<th>Flow formulation label (i.e., the meaning of $$x_{i}^{r}$$)</th>
<th>Expression</th>
<th>Usage constraint formulation label (i.e., the relationship of $$x_{i}^{r}$$ and $$y_{i}^{r}$$)</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>frac (MIP)</td>
<td></td>
<td>lin_cap</td>
<td>(2.5)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>lin_strong</td>
<td>(2.6)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>lin_bigM</td>
<td>(2.7)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sos_each</td>
<td>(2.8)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sos_sum</td>
<td>(2.9)</td>
</tr>
<tr>
<td>flow (MIP2)</td>
<td></td>
<td>lin_cap (equivalent to lin_bigM)</td>
<td>(2.10)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>lin_strong</td>
<td>(2.11)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sos_each</td>
<td>(2.8)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sos_sum</td>
<td>(2.9)</td>
</tr>
</tbody>
</table>
2.2.3 Empirical analysis of MIP formulations

Overview

In this section, we analyze empirical results for solving the numerical example of Section 2.1.2 with the nine aforementioned MIP formulations listed in Table 2.2. The goal is to identify one or more MIP formulations that achieve the best performance from an off-the-shelf optimization software, which would serve as a basis for comparison with our large-scale solution scheme.

For the optimization software, we choose Gurobi version 7.0.1, a state-of-the-art commercial solver that provides free academic licenses and a user-friendly interface with Python. Performance is measured by the trade-off between computation time and solution quality. Solution quality includes both the objective value of the best feasible solution found, and the best lower bound. It is often presented in terms of the MIP gap,

\[
\text{MIP gap} = \left( \frac{\text{best objective} - \text{best bound}}{\text{best bound}} \right) \times 100\%.
\]

Four sets of experiments are conducted, each replicated on 10 instances of randomly generated items. We briefly summarize the setup and findings here; details are given in the subsequent sections.

1. **Gurobi cut generation setting**: For small problems with \(|\mathcal{I}| = 10\) items, we test various levels of cut generation aggressiveness in Gurobi. The results confirm that the default setting achieves the best balance between solution search and cut generation, for all the MIP formulations.

2. **Performance of MIP formulations for small problems**: We solve numerical examples with \(|\mathcal{I}| = 10\) items using all nine MIP formulations.\(^{11}\) The top five performers are `lin_cap/frac`, `lin_cap/flow`, `lin_strong/frac`, `lin_strong/flow`, and `sos_sum/flow`, all of which achieve comparable solution quality given the

\(^{11}\)In the results below, many figures contain 10 formulations, with `lin_bigM/flow` being a duplicate trial that is equivalent to `lin_cap/flow`. 
computation time, although they exhibit different behavior.

3. Performance of MIP formulations for medium and large problems:
The best five formulations identified by the previous experiment are tested at scale, with numerical examples that have $|I| = 100$ and $|I| = 1000$ items. The formulations of lin_cap/frac and lin_cap/flow are found to be the most scalable.

4. Effects of capacity constraint tightness: The tightness of capacity constraints (2.4) are known to affect the difficulty of the problem. For the small numerical example of $|I| = 10$ items, we compare the performance of all nine formulations under various capacity constraint tightness settings.

Gurobi cut generation setting

Branch-and-cut methods form an important class of algorithms for facility location and multi-commodity capacitated fixed-charge network flow problems, as reviewed in Section 2.1.1. This inspires us to investigate whether or not cut generation also improves the efficiency of our problem on a commercial solver like Gurobi.

The Cuts parameter in Gurobi 7.0.1 controls the aggressiveness of cut generation. It can take on the value of $-1$ (default automatic choice), 0 (shut off cuts), 1 (moderate cut generation), 2 (aggressive cut generation), or 3 (very aggressive cut generation). The representative values of $-1$, 0 and 3 are tested. For each Cuts value, we solve the 9 formulation with Gurobi and allow it to run for up to one-hour. The experiment is replicated for 10 instances of small numerical examples with $|I| = 10$. This results in a total of $3 \times 9 \times 10 = 270$ trials.

Figure 2.5 show the results for one problem instance.\textsuperscript{12} In the figure, the solution quality ($y$-axis) is plotted over time ($x$-axis, log scale) for every formulation. Solution quality is given in terms of objective and bound values on the top subfigure, and in terms of MIP gap in the bottom subfigure. Colors of the lines correspond to different Cuts parameter values.

\textsuperscript{12}Results for other instances are similar and are given in Appendix A.1, Figures A.1 and A.2.
The setting \( Cuts = 0 \) represents the control case of searching the branch-and-bound tree with no additional cut-generation methods besides LP relaxation, which explains why the lower bound does not improve.

It may appear somewhat surprising that the most aggressive cut generation setting, \( Cuts = 3 \), is not the quickest to improve bounds. While this setting does not capture all the intricacies of previous work on branch-and-cut methods, the results still reveal the importance of striking a good balance between feasible solution search and cut generation. Finding feasible solutions may help generate better cuts, which would in turn prune more of the branch-and-bound tree and make the search for feasible solutions more efficient.

The figure shows that the default value of \( Cuts = -1 \) achieves the best solution quality over time, providing the fastest improvements in both objective and bound.

**Performance of MIP formulations for small problems**

We compare the performance of the nine MIP formulations under Gurobi’s default \( Cuts \) setting. The experiment is replicated for 10 instances of small numerical examples with \(|I| = 10\). Each formulation and instance is solved on Gurobi for up to one hour.

None of the instances were solved to optimality within one hour. However, the final MIP gaps are very small for all the formulations. To identify the most competitive formulations, we perform a detailed comparison between the formulations of their progress during the one-hour running time. This helps us understand which formulations to use when we are given less computation time or larger problems.

The progress of solution quality over time is presented in Table 2.3 and Figure 2.6. Table 2.3 includes two types of performance results for each formulation: (1) the MIP gap over time, recorded at selected computation time intervals, and (2) the computational time and MIP gap of initial feasible solutions and bounds.\(^{13}\) Both

\(^{13}\)It is worth keeping in mind that Gurobi is not optimized for generating initial feasible solutions quickly, so the initial feasible solution time and quality is merely a proxy for the difficulty of the problem. Also, there is no simple way of probing Gurobi for the initial feasible solution and bound, so results presented throughout this chapter can correspond to the first or the second initial feasible solution found by Gurobi, along with the initial bound.
Figure 2.5: When solving the exact MIP formulation, the default cut generation setting in Gurobi performs best.

(a) Objective and bound over time\(^a\)

(b) MIP gap over time\(^a\)

---

\(^a\)Solid lines indicate the objective value of the incumbent, and dashed lines indicate the best lower bound. The y-axis ranges from 0 to \(3600\) seconds (1 hour).

\(^a\)MIP gap = (objective - bound)/bound \(\times 100\%\).
results are averaged over the 10 instances. Figure 2.6 illustrates the solution quality over time for one problem instance.\footnote{Results for other instances are similar and are given in Appendix A.1, Figures A.3 and A.4.}

From these results, we make following observations:

- The two \texttt{lin\_strong} formulations (with both \texttt{frac} and \texttt{flow}) take a long time to produce initial feasible solutions and lower bounds. This is in line with the facts that \texttt{lin\_strong} is the strongest usage formulation theoretically, and that it also has the largest number of constraints. Nevertheless, the quality of the initial solutions and bounds makes up for the extra time needed.

- For similar reasons, the two \texttt{sos\_each} formulations require the longest to produce initial feasible solutions and lower bounds, but also have the smallest initial MIP gap. However, the gap does not improve as quickly as the \texttt{lin\_strong} formulations.

- The two \texttt{lin\_cap} formulations are the fastest in obtaining a pair of reasonably good initial solution and bound. The solution quality also improves efficiently over time.

- \texttt{sos\_sum/flow} has a similar performance as the two \texttt{lin\_cap} formulations, in terms of both initial solution quality and improvement over time.

- The worst formulations are \texttt{sos\_sum/frac} and \texttt{lin\_bigM/frac}. They produce the worst initial feasible solutions and lower bounds, and are not able to catch up over time.

Based on these results, we select the following formulations for further testing with larger problem instances: \texttt{lin\_cap/flow}, \texttt{lin\_cap/frac}, \texttt{lin\_strong/flow}, \texttt{lin\_strong/frac}, and \texttt{sos\_sum/flow}.

In closing, we remark that the Gurobi solver appears to convert \texttt{sos\_sum} formulations into the big-M formulation for our problem,\footnote{According to the online user’s manual for Gurobi 7.0, “It is often more efficient to capture SOS structure using linear constraints rather than SOS constraints. The optimizer will often perform this} which explains why the initial
Table 2.3: Performance of MIP formulations (Average of 10 instances, \(|I| = 10\))

<table>
<thead>
<tr>
<th>Formulation</th>
<th>Time (sec)</th>
<th>Initial solution</th>
<th>MIP gap (%) over time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time (sec)</td>
<td>Gap (%)</td>
<td>10</td>
</tr>
<tr>
<td>lin_bigM/frac</td>
<td>5.96</td>
<td>3.14</td>
<td>2.32</td>
</tr>
<tr>
<td>lin_cap/flow</td>
<td>4.04</td>
<td>1.14</td>
<td>0.77</td>
</tr>
<tr>
<td>lin_cap/frac</td>
<td>4.04</td>
<td>1.05</td>
<td>0.76</td>
</tr>
<tr>
<td>lin_strong/flow</td>
<td>NA</td>
<td>0.76</td>
<td>0.62</td>
</tr>
<tr>
<td>lin_strong/frac</td>
<td>NA</td>
<td>0.74</td>
<td>0.6</td>
</tr>
<tr>
<td>sos_each/flow</td>
<td>NA</td>
<td>1.07</td>
<td>0.67</td>
</tr>
<tr>
<td>sos_each/frac</td>
<td>NA</td>
<td>1.01</td>
<td>0.76</td>
</tr>
<tr>
<td>sos_sum/flow</td>
<td>4.24</td>
<td>1.52</td>
<td>1.06</td>
</tr>
<tr>
<td>sos_sum/frac</td>
<td>6.84</td>
<td>3.89</td>
<td>2.64</td>
</tr>
</tbody>
</table>

The best value (smallest gap or shortest time) in each column is marked in bold. “NA” indicates that either the feasible solution or the lower bound, or both, have not been found.

The solution and bound of sos_sum/flow and lin_cap/flow are the same, as is the case with sos_sum/frac and lin_bigM/frac. However, the behavior of SOS formulations still differ with that of the big-M formulations over the course of time, so each may still have its own competitive edge. Moreover, this conversion does not always take place, as the sos_each formulations do not have the same initial solutions as the lin_strong formulations. Further investigation is needed to understand what types of problems or solution methods benefit from the use of SOS constraints.

---

conversion automatically. [...] The conversion is done by adding constraints of the form \(x \leq Mk\)”

Figure 2.6: For small problems with 10 items, the top-performing MIP formulations are \texttt{lin\_cap/flow}, \texttt{lin\_cap/frac}, \texttt{lin\_strong/flow}, \texttt{lin\_strong/frac}, and \texttt{sos\_sum/flow}.

(a) Objective and bound over time$^a$

$^a$Solid lines indicate the objective value, and dashed lines indicate the bound. Dots indicate initial values.

(b) MIP gap over time$^a$

$^a$MIP gap = (objective - bound)/bound $\times$ 100\%.
Performance of MIP formulations for medium and large problems

The five most competitive formulations identified in the previous experiments are tested at scale. We use these formulations on Gurobi to solve numerical examples with $|I| = 100$ and $|I| = 1000$ items, each with 10 randomly generated instances. The solution process is allowed to run for 10 hours for each formulation and for each instance. None of the instances were solved to optimality.

Table 2.4 shows the MIP gap over time, averaged over the 10 instances. Figures 2.7 and 2.8 show the detailed performance for one instance of each problem size.\textsuperscript{16}

For the medium-sized problem with $|I| = 100$, the two lin\_strong formulations perform best if given sufficient time.

However, for $|I| = 1000$, none of the formulations exhibit satisfactory performance. The MIP gap is large even after 10 hours. For sos\_sum/flow, solutions are found for only 6 of the 10 instances. For the two lin\_strong formulations not listed in the table, lin\_strong/frac does find a solution for 6 out of the 10 instances, but they all have a MIP gap of over 280%. lin\_strong/flow was unable to find any solution for any instance within 10 hours.

These results motivate the need for a large-scale solution scheme, as we propose in the next section.

\textsuperscript{16}Results for other instances are similar and can be found in Appendix A.1, Figures A.5 and A.6.
Table 2.4: MIP gap over time for medium- and large-scale MIP formulations
(Average of 10 instances)

(a) $|I| = 100$

<table>
<thead>
<tr>
<th>Formulation</th>
<th>Time</th>
<th>100 sec</th>
<th>1000 sec</th>
<th>1 hr</th>
<th>2 hr</th>
<th>3 hr</th>
<th>5 hr</th>
<th>10 hr</th>
</tr>
</thead>
<tbody>
<tr>
<td>lin_cap/flow</td>
<td>6.66</td>
<td>5.78</td>
<td>4.96</td>
<td>3.64</td>
<td>2.73</td>
<td>2.42</td>
<td>2.24</td>
<td></td>
</tr>
<tr>
<td>lin_cap/frac</td>
<td>6.9</td>
<td>5.72</td>
<td>4.31</td>
<td>3.5</td>
<td>3</td>
<td>2.56</td>
<td>2.19</td>
<td></td>
</tr>
<tr>
<td>lin_strong/flow</td>
<td>NA</td>
<td>NA</td>
<td>0.15</td>
<td>0.11</td>
<td>0.1</td>
<td>0.05</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>lin_strong/frac</td>
<td>NA</td>
<td>NA</td>
<td>0.09</td>
<td>0.05</td>
<td>0.05</td>
<td>0.02</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>sos_sum/flow</td>
<td>NA</td>
<td>6.78</td>
<td>6.19</td>
<td>5.89</td>
<td>5.88</td>
<td>5.56</td>
<td>5.21</td>
<td></td>
</tr>
</tbody>
</table>

(b) $|I| = 1000$

<table>
<thead>
<tr>
<th>Formulation</th>
<th>Time</th>
<th>2hr</th>
<th>3 hr</th>
<th>5 hr</th>
<th>10 hr</th>
<th># solved</th>
</tr>
</thead>
<tbody>
<tr>
<td>lin_cap/flow</td>
<td>7.07</td>
<td>7.14</td>
<td>7.01</td>
<td>6.83</td>
<td>10/10</td>
<td></td>
</tr>
<tr>
<td>lin_cap/frac</td>
<td>NA</td>
<td>NA</td>
<td>6.59</td>
<td>6.19</td>
<td>10/10</td>
<td></td>
</tr>
<tr>
<td>sos_sum/flow</td>
<td>7.11</td>
<td>7.09</td>
<td>7.09</td>
<td>7</td>
<td>6/10</td>
<td></td>
</tr>
</tbody>
</table>
Figure 2.7: For medium problems with 100 items, the lin_strong formulations produce near-optimal solutions in a couple of hours.

Figure 2.8: For large problems with 1000 items, none of the MIP formulations perform well.
Effects of capacity constraint tightness

Capacity constraint tightness affects the problem difficulty and optimal cost of the MIP. In general, tighter capacity constraints make the MIP more difficult to solve, and also result in an optimal placement plan of higher cost. To quantify these effects on the nine MIP formulations, we test them under various levels of capacity constraint tightness.

Recall from Section 2.1.2 that all the aforementioned experiments are performed with a default slack of 10% in capacity constraints, i.e., the total capacity $\sum_{n=1}^{N} b_n$ is 110% of the total demand $\sum_{i \in Z} d_i$. In this experiment, we vary the tightness of capacity constraints by setting the amount of slack to 5%, 20%, and 50%. We perform one-hour runs of all nine formulations, for 10 instances with $|Z| = 10$, the same set of problems as in Sections (2.2.3) and (2.2.3).

Problem difficulty. As a proxy for MIP difficulty, we measure how quickly the MIP gap improves over time, as well as the computation time and MIP gap of the initial feasible solution and bound.

Figure 2.9 depicts the average MIP gap for the 10 problem instances, measured at various computation time intervals. The detailed progression of MIP gap over time for each instance is given in Appendix A.1, Figure A.7. According to these results, greater slackness in capacity constraints results in an improved efficiency in closing the MIP gap, but the effect does not appear to be significant.

Table 2.5 reports the average time and MIP gap for the initial solution (and bound) of each formulation. From the table, we see that for the four strong formulations with lin_strong and sos_each, an increase in capacity constraint slackness leads to a reduction in initial solution time, as well as an improved MIP gap. This indicates that problems with greater excess capacity are easier to solve for these formulations—a better MIP gap can be found in less time. On the other hand, the effect is not very pronounced for the other formulations, but we see a general trend of a slight decrease in initial solution time and a slight increase in the initial MIP gap as slackness increases. In other words, it is easier to find an initial feasible solution, but the solution
Figure 2.9: Greater capacity slackness results in slightly improved MIP solution efficiency.

The MIP gap of each formulation and each instance is recorded after computation times of 1, 5, 10, 50, 100, 500, 1000, and 3600 seconds, and averaged over 10 instances.

quality may not be great.

**Objective cost.** Figure 2.10 shows the best objective cost for each instance at each level of capacity constraint tightness.\(^\text{17}\) The figure validates the fact that tighter capacity constraints result in a higher cost. Moreover, it indicates a general trend of diminishing returns— the cost savings decrease at a decreasing rate as excess capacity increases.

### 2.2.4 Conclusions and future directions

The main purpose of this section is to illustrate that even the best MIP formulation and the most up-to-date commercial solver cannot effectively handle the online-retail inventory placement problem at a practical scale. This motivates the need for a

\(^{17}\)While none of the instances were solved to optimality within one hour by any formulation, we collected the best objective cost and best lower bound for each instance, produced across all formulations, and computed the resulting MIP gap. The MIP gap for all instances are no greater than 0.27%. This indicates that the best objective costs are a good approximation of the true optimal cost for each instance.
Table 2.5: Initial solution performance for different levels of capacity constraint tightness
\((|I| = 10, \text{ average of } 10 \text{ instances})\)

<table>
<thead>
<tr>
<th>Formulation</th>
<th>Slack</th>
<th>Initial solution time (sec)</th>
<th>Initial MIP gap (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5%</td>
<td>10%</td>
<td>20%</td>
</tr>
<tr>
<td>lin_bigM/frac</td>
<td></td>
<td>2.32</td>
<td>2.23</td>
</tr>
<tr>
<td>lin_cap/flow</td>
<td></td>
<td>2.04</td>
<td>2.05</td>
</tr>
<tr>
<td>lin_cap/frac</td>
<td></td>
<td>2.00</td>
<td>1.82</td>
</tr>
<tr>
<td>lin_strong/flow</td>
<td></td>
<td>31.22</td>
<td>26.38</td>
</tr>
<tr>
<td>lin_strong/frac</td>
<td></td>
<td>31.44</td>
<td>24.00</td>
</tr>
<tr>
<td>sos_each/flow</td>
<td></td>
<td>75.81</td>
<td>63.15</td>
</tr>
<tr>
<td>sos_each/frac</td>
<td></td>
<td>78.31</td>
<td>56.73</td>
</tr>
<tr>
<td>sos_sum/flow</td>
<td></td>
<td>3.14</td>
<td>2.85</td>
</tr>
<tr>
<td>sos_sum/frac</td>
<td></td>
<td>3.39</td>
<td>2.93</td>
</tr>
</tbody>
</table>

Figure 2.10: As capacity slackness increases, there is diminishing returns in cost reduction.
large-scale solution scheme that is specifically designed to accommodate, and even take advantage of, the multi-item aspect of the problem. We present such a solution scheme in the next section.

We acknowledge, however, that there are other factors which may affect the efficiency of MIP computations that we have not fully explored. For example, the difficulty of the problem may depend on the magnitude of the fixed cost relative to the variable cost. Also, the level of heterogeneity in item characteristics may also impact the difficulty of the problem—identical items lead to multiple near-optimal solutions, while extremely heterogeneous items may lead to greater complexity, both of which can increase computational difficulty. While these factors do not fundamentally change the curse of dimensionality for the MIP, they may affect the ability of commercial solvers to handle particular problem instances in practice, and further investigation may be an interesting topic of future work.

Another direction of future work is to compare the MIP with the wide variety of exact and heuristic methods reviewed in Section 2.1.1 and beyond. There is no evidence that they would work well, as they are not designed to exploit the structure of our problem, and no results have been reported for numerical examples at our scale. Nevertheless, some of these methods may outperform commercial solvers, especially in their ability to provide lower bounds on the objective cost.

2.3 A large-scale solution scheme

We propose a large-scale solution scheme for the online-retail inventory placement problem that is designed to handle a large number of items, which is typical in practice. This solution scheme is built upon two techniques: item aggregation and column generation. It consists of five stages, as illustrated in Figure 2.11:

(A) Aggregation of items: The large set of items is first partitioned into subsets, each of which is represented as an aggregate item called a bundle.
Figure 2.11: Our large-scale solution scheme consists of five stages, labeled A-E: Aggregation, Bundle characterization, Column-based reformulation, Disaggregation, and Enhancement.

(B) **Bundle characterization:** The cost and demand characteristics are calculated for each bundle so as to be representative of its items. This allows us to formulate the placement problem in terms of bundles.

(C) **Column-based reformulation:** We then employ column generation to solve an approximate column-based formulation of the bundle placement problem. In this formulation, each column represents a placement plan for a bundle. The resulting column solution is a convex combination of selected columns for each bundle.

(D) **Disaggregation into items:** The column solution of each bundle is disaggregated into placement plans for individual items in the bundle.

(E) **Enhancement of item placement plans:** Finally, we enhance the placement plan for each item by searching for an improved placement plan that achieves a lower cost while using the same amount of capacity in each fulfillment center.

Details for each stage are explained in the subsequent sections. Before proceeding, we highlight the way in which item aggregation and column generation go hand-in-
hand in this solution scheme. Item aggregation reduces the number of variables in
the placement problem. It also allows us to relax the column-based formulation,
as it provides a meaningful interpretation for the convex combination of columns.
Column generation then solves the relaxed column-based formulation efficiently. The
column generation subproblems are also decomposed by bundles, which allows for
parallelized computation. Neither technique works without the other, and together,
they overcome and even take advantage of the curse of dimensionality.

2.3.1 Aggregation of items

Item aggregation is the act of partitioning the set of items by \( I = \bigcup_{j \in J} I_j \), where
each \( j \in J \) is called a bundle. After aggregation, the placement decision for every
individual item \( i \in I_j \) is virtually placed by a single decision for the bundle \( j \).

The overall goal of aggregation is to group together the items for which similar
placement plans are desirable. Thus, the appropriate method of aggregation may
depend on several considerations, such as:

- **The business context**: In many businesses, items are often already organized in
  a hierarchical or categorical manner that lends itself naturally to some partition.

- **Additional data about the items**: If there is additional data on which items are
  likely to be purchased together by customers (e.g., from the history of orders),
  then it may be a good idea to aggregate those items in the same bundle, so as
to make it easier for those items to be shipped from the same fulfillment center.

- **Computing resources**: As we shall see shortly, the size and characteristics of
  bundles affect the complexity of the large-scale solution scheme, and should
  be chosen to achieve the desired trade-off between computational effort and
  solution quality.

Given these considerations, the aggregation step may involve an optimization
problem to obtain the best partitioning of items according to some measure. For
instance, we may wish to aggregate “similar” items together by minimizing some
notion of “distance” between items in the same bundle, where distance can be defined in terms of item characteristics and category, likelihood of joint purchase, spatial or seasonal demand patterns, etc. For certain types of distance measures, clustering algorithms such as K-means can be efficient heuristics that produce good partitions without solving a full-fledged optimization problem.

For the purpose of this section, we assume that the partition \( \{I_j\}_{j \in J} \) is determined by an exogenous aggregation method and is given as input. In Section 2.4, we empirically evaluate two general-purpose aggregation methods, as well as various number of bundles, to study their effects on the performance of the large-scale solution scheme.

### 2.3.2 Bundle characterization

To rewrite the inventory placement problem in terms of bundles, we now derive bundle characteristics that accurately represent those of its items. For our problem formulation, the relevant characteristics include the regional demand, shipping costs, and fixed costs.

Mathematically, given a partition \( \{I_j\}_{j \in J} \), item aggregation corresponds to replacing \( x_{n r}^i \) by \( x_{n r}^j \) in (MIP), and likewise replacing \( y_n^i \) with \( y_n^j \), for all \( i \in I_j \). This results in the following formulation:

\[
\begin{align*}
\min_{x,y} & \quad \sum_{j \in J} \sum_{i \in I_j} \sum_{n=1}^{N} \sum_{r=1}^{R} c_{r n}^j d_{r n} x_{n r}^j + \sum_{j \in J} \sum_{i \in I_j} \sum_{n=1}^{N} f_{n}^i y_{n}^j \\
\text{subject to} & \quad \sum_{n=1}^{N} x_{n r}^j = 1 \quad \forall r = 1, \ldots, R, \forall j \in J, \\
& \quad \sum_{i \in I_j} \sum_{n=1}^{N} \sum_{r=1}^{R} d_{r n} x_{n r}^j \leq b_{n} \quad \forall n = 1, \ldots, N, \\
& \quad \sum_{r=1}^{R} d_{r n} x_{n r}^j \leq d_{n}^j y_{n}^j \quad \forall n = 1, \ldots, N, \forall i \in I \text{ and } \forall j \in J \text{ such that } i \in I_j, \\
& \quad x_{n r}^j \geq 0 \quad \forall n = 1, \ldots, N, \forall r = 1, \ldots, R, \forall j \in J, \\
& \quad y_{n}^j \in \{0, 1\} \quad \forall n = 1, \ldots, N, \forall j \in J.
\end{align*}
\] (2.12)
By defining bundle characteristics

\[ d^i_r := \sum_{i \in I_j} d^i_r, \quad d^j := \sum_{i \in I_k} d^i, \quad c^i_{nr} := \frac{\sum_{i \in I_j} c^i_{nr} d^i_r}{\sum_{i \in I_j} d^i_r}, \quad f^j_n := \sum_{i \in I_j} f^i_n \quad \forall j \in J, \]

we can write the formulation above in the exact same form as (MIP), but with \( j \in J \) rather than \( i \in I \):

\[
\begin{align*}
\min_{x,y} & \quad \sum_{j \in J} \sum_{n=1}^{N} \sum_{r=1}^{R} c^j_{nr} d^j_{nr} x^j_{nr} + \sum_{j \in J} \sum_{n=1}^{N} f^j_n y^j_n \quad \text{(MIP-B)} \\
\text{subject to} & \quad \sum_{n=1}^{N} x^j_{nr} = 1 \quad \forall r = 1, \ldots, R, \forall j \in J, \quad (2.13) \\
& \quad \sum_{n=1}^{N} d^j_{nr} x^j_{nr} \leq b_n \quad \forall n = 1, \ldots, N, \quad (2.14) \\
& \quad \sum_{r=1}^{R} d^j_{nr} x^j_{nr} \leq d^j y^j_n \quad \forall n = 1, \ldots, N, \forall j \in J, \quad (2.15) \\
& \quad x^j_{nr} \geq 0 \quad \forall n = 1, \ldots, N, \forall r = 1, \ldots, R, \forall j \in J, \quad (2.16) \\
& \quad y^j_n \in \{0, 1\} \quad \forall n = 1, \ldots, N, \forall j \in J. \quad (2.17)
\end{align*}
\]

For the objective, it is straightforward to verify that \( c^j_{nr} d^j_{nr} = \sum_{i \in I_j} c^i_{nr} d^i \). Note that although (2.12) is not equivalent to (2.15) (in fact, the former implies the latter, but not the other way round), both are valid expressions of the relationship that “\( y^j_n = 1 \) if \( x^j_{nr} > 0 \) for some \( r \).”

The optimal solution of (MIP-B) is an upper bound for the optimal solution of (MIP), because (MIP-B) can also be obtained by adding to (MIP) the variables \( \{x^j_{nr}\}_{j \in J} \) and constraints

\[ x^i_{nr} = x^j_{nr}, \quad y^i_n = y^j_n, \quad \forall i \in I \text{ and } \forall j \in J \text{ such that } i \in I_j. \]

Substituting \( x^i_{nr} \) with \( x^j_{nr} \) results in (MIP-B) having fewer decision variables than (MIP), which makes it easier to solve, although it may still be computationally
prohibitive as a mixed integer program. Fortunately, our large-scale solution scheme does not require actually solving (MIP-B). Instead, we work with an LP relaxation of an equivalent formulation, presented in the following section.

2.3.3 Column-based reformulation

This section describes a three-step approach for solving an approximation of (MIP-B) with the technique of column generation. First, we derive a column-based formulation that is equivalent to (MIP-B), in which a column corresponds to the placement plan of a bundle, and there are infinitely many columns. Then, we show that for the relaxation of the column-based formulation, it suffices to optimize over a finite, though large, subset of columns. Finally, we present a column generation procedure that efficiently solves the relaxed column-based formulation without having to consider all the columns. The outcome of this stage is a column solution that is a convex combination of columns.

A column-based formulation

Let \( x^j = (x^j_{nr})_{n=1,...,N,r=1,...,R} \) and \( y^j = (y^j_n)_{n=1,...,N} \) be shorthand notations for decision variables corresponding to bundle \( j \). To differentiate between the solution of (MIP-B) and the solution of the column-based formulation below, we shall refer to \((x^j, y^j)\) as the placement plan whenever appropriate.

Let \( X_j \) be the set of placement plans that satisfy constraints (2.13), (2.15), (2.16) and (2.17) for some bundle \( j \in J \), i.e.,

\[
X_j := \left\{ (x^j, y^j) \left| \sum_{n=1}^{N} x^j_{nr} = 1, \sum_{r=1}^{R} d^j_r x^j_{nr} \leq d^j y^j_n, \ x^j_{nr} \geq 0, \ y^j_n \in \{0, 1\}, \ \forall n, \forall r \right. \right\}.
\] (2.18)
Then (MIP-B) can be simplified as

$$\min_{x,y} \sum_{j \in \mathcal{J}} \sum_{n=1}^N \sum_{r=1}^R c_{nr}^j d_{nr}^j x_{nr}^j + \sum_{j \in \mathcal{J}} \sum_{n=1}^N f_{n}^j y_{n}^j$$

(subject to \(\sum_{j \in \mathcal{J}} \sum_{r=1}^R d_{nr}^j x_{nr}^j \leq b_n\) \quad \forall n = 1, \ldots, N,

\((x^j, y^j) \in X_j\) \quad \forall j \in \mathcal{J}.

To derive a column-based formulation for (MIP-B), let \(\mathcal{K}_j\) be a (possibly uncountable) set with a one-to-one correspondence to the placement plans in \(X_j\), where \(k \in \mathcal{K}_j\) represents a placement plan \((x^j(k), y^j(k)) \in X_j\). For every \(k \in \mathcal{K}_j\), denote its objective cost as

\[c_k^j := \sum_{n=1}^N \sum_{r=1}^R c_{nr}^j d_{nr}^j x_{nr}^j(k) + \sum_{n=1}^N f_{n}^j y_{n}^j(k),\]

and its amount of capacity usage at fulfillment center \(n\) as

\[u_{kn}^j := \sum_{r=1}^R d_{nr}^j x_{nr}^j(k), \quad n = 1, \ldots, N.\]

With these definitions, it is easy to see that \((P_1)\) is equivalent to the following column-based formulation, in the sense that there is a one-to-one mapping of feasible solutions (and their objective costs) between the two problems:

$$\min_{\lambda} \sum_{j \in \mathcal{J}} \sum_{k \in \mathcal{K}_j} c_k^j \lambda_k^j$$

(subject to \(\sum_{j \in \mathcal{J}} \sum_{k \in \mathcal{K}_j} u_{kn}^j \lambda_k^j \leq b_n\) \quad \forall n = 1, \ldots, N,

\(\sum_{k \in \mathcal{K}_j} \lambda_k^j = 1\) \quad \forall j \in \mathcal{J},

\(\lambda_k^j \in \{0, 1\}\) \quad \forall k \in \mathcal{K}_j, \forall j \in \mathcal{J},

where each decision variable \(\lambda_k^j\) indicates whether or not bundle \(j\) takes on the placement plan of \((x^j(k), y^j(k)) \in X_j\). Equation (2.19) is the set of capacity constraints,
and equations (2.20) and (2.21) together ensure that exactly one placement plan is selected. Since each decision variable corresponds to a column in the constraint matrix of (P_2), we shall refer to a placement plan in X_j, or interchangeably, its index in K_j, as a column for bundle j \in J. A column k is said to be active if \lambda^j_k > 0.

This column-based formulation (P_2) does not currently have any advantages over the original expression of (MIP-B). If anything, we seem to have created an impractical formulation by suggesting that we can somehow enumerate the uncountably infinite number of placement plans as columns! Fortunately, as we show next, the relaxation of (P_2) has desirable properties that is useful for our large-scale solution scheme.

**Relaxation of the column-based formulation**

Let (P_3) denote the relaxation of (P_2),

\[
\min_{\lambda} \sum_{j \in J} \sum_{k \in K_j} c^j_k \lambda^j_k \\
\text{subject to } \sum_{j \in J} \sum_{k \in K_j} u^j_{kn} \lambda^j_k \leq b_n \quad \forall n = 1, \ldots, N, \\
\sum_{k \in K_j} \lambda^j_k = 1 \quad \forall j \in J, \\
0 \leq \lambda^j_k \leq 1 \quad \forall k \in K_j, \forall j \in J.
\]

As a relaxation, the optimal objective cost of (P_3) is a lower bound of that of (P_2). (Unfortunately, this does not tell us much about the cost of the original problem (MIP), as the cost of (P_2) is equal to that of (MIP-B) and is an upper bound of (MIP).)

An important effect of this relaxation is that the optimal solution of (P_3) can be fully captured by the union of vertices of each X_j, i.e., the finite subset of integer placement plans in X_j for which every x^j_{nr} is 0 or 1. As a result, while (P_2) is not tractable, (P_3) is a linear program with a finite set of variables. This is stated formally as follows:
Theorem 1. For every bundle $j \in J$, let $\bar{X}_j$ be the vertices of $X_j$, and let $\bar{K}_j \subset K_j$ be the index set corresponding to $\bar{X}_j$. Then there exists an optimal solution for $(P_3)$ such that $\lambda^j_k = 0$ for every $j \in J$ and every $k \in K_j \setminus \bar{K}_j$.

Proof. Let $\lambda = (\lambda_k^j)_{j \in J, k \in K_j}$ be an optimal solution of $(P_3)$ where there exists some bundle $j \in J$ with an active column $k^* \in K_j \setminus \bar{K}_j$, i.e., $\lambda^j_{k^*} > 0$ and $x^j(k^*)$ is fractional. We show that there exists an optimal solution in which column $k^*$ is replaced by other columns with fewer fractional entries. Then, by induction, there exists an optimal solution in which all active columns correspond to integer placement plans.

Let $r'$ be a region such that $0 < x_{n_1 r'}^j(k^*) < 1$ and $0 < x_{n_2 r'}^j(k^*) < 1$ for some fulfillment centers $n_1 \neq n_2$. (Note that the demand satisfaction constraint (2.13) implies that if $x_{n r'}^j(k^*)$ is fractional for some $n$, then $r'$ must be served by at least two fulfillment centers.) For simplicity in notation, let $v_1 = x_{n_1 r'}^j(k^*)$ and $v_2 = x_{n_2 r'}^j(k^*)$ be the fractional values.

Consider the columns $k_1, k_2 \in K_j$ corresponding to the placement plans

$$
\begin{align*}
    x_{n r}^j(k_1) &= \begin{cases} 
        v_1 + v_2, & r = r', n = n_1, \\
        0, & r = r', n = n_2, \\
        x_{n r}^j(k^*), & \text{otherwise},
    \end{cases}
    \quad \text{and} \quad
    x_{n r}^j(k_2) &= \begin{cases} 
        0, & r = r', n = n_1, \\
        v_1 + v_2, & r = r', n = n_2, \\
        x_{n r}^j(k^*), & \text{otherwise}.
    \end{cases}
\end{align*}
$$

In other words, the columns $k_1$ and $k_2$ are obtained by consolidating the fractional demand allocation to either of the two fulfillment centers. It is clear that $x^j(k_1)$ and $x^j(k_2)$ both have fewer fractional entries than $k^*$. Moreover, differences in the cost of columns are given by

$$
\begin{align*}
    c^j_{k_1} - c^j_{k^*} &= (c^j_{n_1 r'} - c^j_{n_2 r'}) d^j_{r'} v_2 + f^j_{n_2} (y^j_{n_2}(k_1) - y^j_{n_2}(k^*)) , \\
    c^j_{k_2} - c^j_{k^*} &= (c^j_{n_2 r'} - c^j_{n_1 r'}) d^j_{r'} v_1 + f^j_{n_1} (y^j_{n_1}(k_2) - y^j_{n_1}(k^*)) ,
\end{align*}
$$

(2.22) (2.23)
and differences in their capacity usage are given by

\[
\begin{align*}
    w^j_{kn} - w^*_{kn} &= \begin{cases} 
        v_2d_{jr}, & n = n_1, \\
        -v_2d_{jr}, & n = n_2, \\
        0, & \text{otherwise,}
    \end{cases} \\
    \text{and } u^j_{kn} - u^*_{kn} &= \begin{cases} 
        -v_1d_{jr}, & n = n_1, \\
        v_1d_{jr}, & n = n_2, \\
        0, & \text{otherwise.}
    \end{cases}
\end{align*}
\]

(2.24)

We now show that another optimal solution of (P_3) is given by \( \tilde{\lambda} = (\tilde{\lambda}^j_{k})_{j \in J, k \in K_j} \), which has the same value as \( \lambda \), except for the following entries for columns \( k^*, k_1, k_2 \in K_j \):

\[
\begin{align*}
    \tilde{\lambda}^j_{k_1} &= \lambda^j_{k_1} + \frac{v_1}{v_1 + v_2} \lambda^j_{k^*}, \\
    \tilde{\lambda}^j_{k_2} &= \lambda^j_{k_2} + \frac{v_2}{v_1 + v_2} \lambda^j_{k^*}, \\
    \tilde{\lambda}^j_{k^*} &= 0. 
\end{align*}
\]

(2.25)

In other words, \( \tilde{\lambda} \) is modified from \( \lambda \) by deselecting column \( k^* \) for bundle \( j \) and distributing its selection variable value, \( \lambda^j_{k^*} > 0 \), between columns \( k_1 \) and \( k_2 \).

The optimality of \( \tilde{\lambda} \) can be established by showing the following two properties:

- **The solution \( \tilde{\lambda} \) is feasible.** It is straightforward to verify that \( \tilde{\lambda} \) satisfies the selection constraints (2.20). Moreover, the left-hand side of capacity constraints (2.19) have the same value for both solutions \( \tilde{\lambda} \) and \( \lambda \). Indeed, for these two solutions, the difference in capacity usage for fulfillment center \( n = n_1 \) is

\[
\begin{align*}
    \sum_{k \in K_j} u^j_{kn_1} (\tilde{\lambda}^j_k - \lambda^j_k) &= u^j_{kn_1} (\tilde{\lambda}^j_{k_1} - \lambda^j_{k_1}) + u^j_{kn_1} (\tilde{\lambda}^j_{k_2} - \lambda^j_{k_2}) - u^j_{kn_1} \lambda^j_{k^*} \\
    &= (u^j_{kn_1} + v_2d_{jr}) \frac{v_1}{v_1 + v_2} \lambda^j_{k^*} + (u^j_{kn_1} - v_1d_{jr}) \frac{v_2}{v_1 + v_2} \lambda^j_{k^*} - u^j_{kn_1} \lambda^j_{k^*} \\
    &= 0,
\end{align*}
\]

where the second line is obtained using (2.24) and (2.25), and the last line by rearranging the terms. Similarly, we can show that the difference in capacity usage for fulfillment center \( n_2 \). Capacity usage for rest of the fulfillment center are unaffected.

79
• The objective cost of solution \( \tilde{\lambda} \) is no greater than that of \( \lambda \). The difference in the objective cost of \( \tilde{\lambda}^j_k \) and \( \lambda^j_k \) is

\[
\sum_{k \in \mathcal{K}_j} c_j^k (\tilde{\lambda}^j_k - \lambda^j_k) = c_{j_{k_1}}^j (\tilde{\lambda}^j_{k_1} - \lambda^j_{k_1}) + c_{j_{k_2}}^j (\tilde{\lambda}^j_{k_2} - \lambda^j_{k_2}) - c_{k^*}^j \lambda^j_{k^*}
\]

\[
= c_{j_{k_1}}^j \frac{v_1}{v_1 + v_2} \lambda^j_{k^*} + c_{j_{k_2}}^j \frac{v_2}{v_1 + v_2} \lambda^j_{k^*} - c_{k^*}^j \lambda^j_{k^*}
\]

\[
= \frac{v_1}{v_1 + v_2} \lambda^j_{k^*} \left(c_{j_{k_1}}^j - c_{k^*}^j\right) + \frac{v_2}{v_1 + v_2} \lambda^j_{k^*} \left(c_{j_{k_2}}^j - c_{k^*}^j\right)
\]

\[
= \frac{v_1}{v_1 + v_2} \lambda^j_{k^*} \left[(c_{j_{n_2r'}}^j - c_{j_{n_2r}}^j) d_{r'} v_2 + f_{n_2}^j (y_{n_2}^j(k_1) - y_{n_2}^j(k^*))\right]
\]

\[
+ \frac{v_2}{v_1 + v_2} \lambda^j_{k^*} \left[(c_{j_{n_2r}}^j - c_{j_{n_1r}}^j) d_{r'} v_1 + f_{n_1}^j (y_{n_1}^j(k_2) - y_{n_1}^j(k^*))\right]
\]

\[
= \frac{v_1}{v_1 + v_2} \lambda^j_{k^*} f_{n_2}^j (y_{n_2}^j(k_1) - y_{n_2}^j(k^*))
\]

\[
+ \frac{v_2}{v_1 + v_2} \lambda^j_{k^*} f_{n_1}^j (y_{n_1}^j(k_2) - y_{n_1}^j(k^*))
\]

\[
\leq 0,
\]

where the second equality is obtained by (2.25); the third equality by rearranging the terms; the fourth equality by the expressions for cost differences (2.22) and (2.23); the fifth equality by canceling terms. The final inequality is due to the fact that \( y_{n_2}^j(k_1) \leq y_{n_2}^j(k^*) \), since \( x_{n_2r'}^j(k_1) = 0 < x_{n_2r'}^j(k^*) \), i.e., fulfillment center \( n_2 \) serves one fewer region (the region \( r' \)) in column \( k_1 \) than in column \( k^* \); and similarly, \( y_{n_1}^j(k_2) \leq y_{n_1}^j(k^*) \).

Thus, \( \tilde{\lambda} \) is a feasible solution whose cost is at least as good as \( \lambda \), making it an optimal solution as well. Moreover, \( \tilde{\lambda} \) has fewer active columns in \( \bar{\mathcal{K}} \) than \( \lambda \). By repeating this argument until all active columns in \( \bar{\mathcal{K}} \) are removed, we have shown that there exists an optimal solution in which all active columns are in \( \mathcal{K} \setminus \bar{\mathcal{K}} \).

\[\square\]

As a result of Theorem 1, (P3) can be simplified into the optimization over the set of columns \( \bar{\mathcal{K}}_j \) that corresponds to the set of integer placement plans \( X_j \), for every
bundle $j \in J$; i.e.,

$$
\min_{\lambda} \sum_{j \in J} \sum_{k \in K_j} c^j_k \lambda^j_k \quad (P_4)
$$

subject to

$$
\sum_{j \in J} \sum_{k \in K_j} u^j_{kn} \lambda^j_k \leq b_n \quad \forall n = 1, \ldots, N,
$$

$$
\sum_{k \in K_j} \lambda^j_k = 1 \quad \forall j \in J,
$$

$$
\lambda^j_k \geq 0 \quad \forall k \in K_j, \forall j \in J,
$$

Column generation

Even though the number of columns in $(P_4)$ is finite, it is still exponential with respect to $|J|, N$ and $R$, which is computationally prohibitive for realistic problem sizes.

Column generation is a technique for solving $(P_4)$ efficiently. Instead of considering all the columns in $\bar{K}_j$, we restrict our attention to a subset of columns $\bar{K}'_j \subset \bar{K}_j$, and solve the so-called restricted master problem

$$
\min_{\lambda} \sum_{j \in J} \sum_{k \in \bar{K}'_j} c^j_k \lambda^j_k \quad (RMP)
$$

subject to

$$
\sum_{j \in J} \sum_{k \in \bar{K}'_j} u^j_{kn} \lambda^j_k \leq b_n \quad \forall n = 1, \ldots, N, \quad (2.26)
$$

$$
\sum_{k \in \bar{K}'_j} \lambda^j_k = 1 \quad \forall j \in J, \quad (2.27)
$$

$$
\lambda^j_k \geq 0 \quad \forall k \in \bar{K}'_j, \forall j \in J.
$$

The column generation procedure is as follows. First, the sets $\{\bar{K}'_j\}_{j \in J}$ are initialized in a way that ensures the feasibility of $(RMP)$. Then, $(RMP)$ is solved as a linear program, and its dual variables are given as inputs to the column generation subproblems, which produce new columns that have a potential of reducing the objective value. If any such columns are identified, they are added to the subsets $\bar{K}'_j$, and $(RMP)$ is solved again. The process continues iteratively until no more cost-reducing columns can be found, upon which it terminates with the optimal solution of $(P_4)$. 81
A formal description of the column generation procedure is given in Algorithm 2.3.1. Below, we provide further details on important components of the procedure, namely, the column generation subproblems (Algorithm 2.3.1, Line 8), initialization of the sets \( \bar{K}_j \) (Line 1), and the possibility of early termination (Line 2).

**Algorithm 2.3.1** Stage C: Solving the column-based reformulation with column generation

1: \( \bar{K}_j' \leftarrow \bar{K}_j^0 \quad \forall j \in J \) \( \Rightarrow \bar{K}_j^0 \) is given by (2.29)
2: while termination criteria not met do
3: Solve the restricted master problem (RMP)
4: \( \{p_n\}_{n=1}^N \leftarrow \) dual variables of (2.26)
5: \( \{q_j\}_{j \in J} \leftarrow \) dual variables of (2.27)
6: added \( \leftarrow \) False
7: for all \( j \in J \) do
8: Solve the column generation subproblem (CGj)
9: \( q \leftarrow \) objective value of (CGj)
10: \( k \leftarrow \) optimal solution of (CGj) as column index
11: if \( q < q_j \) then \( \Rightarrow \) Found column with negative reduced cost
12: \( \bar{K}_j' \leftarrow \bar{K}_j' \cup \{k\} \)
13: \( \text{added} \leftarrow \text{True} \)
14: end if
15: end for
16: if \( \text{added} = \text{False} \) then
17: Go to Line 20
18: end if
19: end while
20: return Optimal column solution of (P4)

**Column generation subproblems.** Let \( \{p_n\}_{n=1}^N \) and \( \{q_j\}_{j \in J} \) be the dual variables of (2.26) and (2.27), respectively, of the current solution of (RMP). For every bundle \( j \in J \), the reduced cost of a column \( k \in \bar{K}_j \setminus \bar{K}_j' \) is given by

\[
c_k^j = \sum_{n=1}^N p_n w_{k,n}^j - q_j, \tag{2.28}
\]
and reflects the first derivative of the objective cost of \( (P_4) \) with respect to the column variable \( \lambda^j_k \), given the current solution of \( (RMP) \) (which can be trivially converted into a solution of \( (P_4) \) by setting the selection variables of columns in \( \bar{K}_j \backslash \bar{K}'_j \) to 0 for all \( j \in J \)). The optimal solution of \( (RMP) \) is equal to the optimal solution of the unrestricted problem \( (P_3) \) (as well as \( (P_4) \)) if and only if all columns in \( \bar{K}_j \backslash \bar{K}'_j \) have nonnegative reduced costs. If we can identify a column with a negative reduced cost, then this column should be added to \( \bar{K}'_j \), and \( (RMP) \) should then be re-solved. Note also that by linear programming duality, the reduced cost is nonnegative for every column in \( \bar{K}'_j \), i.e., all columns that are already considered by \( (RMP) \).

To generate cost-reducing columns for each \( j \in J \), or otherwise prove that there are none, we find the column in the set \( \bar{K}_j \) with the smallest reduced cost (2.28) and check if the reduced cost is negative. This amounts to solving the following column generation subproblem for bundle \( j \):

\[
\begin{align*}
\min_{x^j, y^j} & \quad \sum_{n=1}^{N} \sum_{r=1}^{R} (c^j_{nr} - p_n) d_r^j x^j_{nr} + \sum_{n=1}^{N} f^j_n y^j_n \\
\text{subject to} & \quad (x^j, y^j) \in X_j,
\end{align*}
\]

where the objective function consists of the first two terms in the reduced cost (2.28). If the optimal objective value of the problem above is strictly smaller than \( q_j \), then we add the optimal placement plan as the column to \( \bar{K}'_j \). (As noted above, such a column cannot already be contained in the set \( \bar{K}'_j \), as all columns in \( \bar{K}'_j \) have a nonnegative reduced cost by duality.)

The collection of column generation subproblems \( (P_5) \) are decomposed by bundles, which means that \( (P_5) \) can be solved independently for each \( j \in J \). Each \( (P_5) \) only has \( NR + N \) variables, making it relatively easy to solve, despite being an integer program.

In fact, \( (P_5) \) can be made even easier to solve. Observe that there are no capacity constraints on the flow variables \( x^j_{nr} \), which implies that if we relax their integrality, the vertices of the feasible set will be the integer points. Therefore, we can obtain the same optimal integer placement plan for \( (P_5) \) by optimizing over the set of continuous
placement plans $X_j$:

$$\begin{align*}
\min_{x^j, y^j} & \sum_{n=1}^{N} \sum_{r=1}^{R} (c_{nr}^j - p_n) d_r^j x_{nr}^j + \sum_{n=1}^{N} f_n^j y_n^j \\
\text{subject to} & \quad \sum_{n=1}^{N} x_{nr}^j = 1 \quad \forall r = 1, \ldots, R, \\
& \quad \sum_{r=1}^{R} d_r^j x_{nr}^j \leq d_r^j y_n^j \quad \forall n = 1, \ldots, N, \\
& \quad x_{nr}^j \geq 0 \quad \forall n = 1, \ldots, N, \forall r = 1, \ldots, R, \\
& \quad y_n^j \in \{0, 1\} \quad \forall n = 1, \ldots, N,
\end{align*} \quad (CG_j)$$

where the constraints are the expansion of the statement that $(x^j, y^j) \in X_j$, by the definition of placement plans $X_j$ given in (2.18).

In our implementation of the large-scale solution scheme, we use $(CG_j)$ rather than $(P_5)$, as we found that the former requires a shorter solution time in Gurobi because it has fewer binary variables. If fractional solutions are somehow produced (either because there are multiple optimal solutions, some of which are fractional, or because of numerical limitations of computer software, so that very small differences in objective values cannot be detected), we round the solution to a vertex in $\bar{X}_j$ and verify its optimality before using it (by comparison with the objective of the fractional solution).

**Initial feasible solutions.** As initial feasible solutions of $(P_4)$, we take the set of $n$ columns

$$\mathcal{K}_j^0 = \{1, \ldots, N\} \quad (2.29)$$
for every $j \in J$, where $k = 1, \ldots, N$ corresponds to the solution of using only fulfillment center $k$, i.e.,

$$x_{nr}^j(k) = y_{kj}^j = \begin{cases} 1, & n = k, \\ 0, & \text{otherwise}, \end{cases} \quad \forall k = 1, \ldots, N.$$ 

The corresponding cost and usage of $k = 1, \ldots, N$ is

$$c^j_k = c^j_{nr} d^j + f^j_n,$$

$$u^j_{k,n} = \begin{cases} d^j, & n = k, \\ 0, & \text{otherwise}, \end{cases} \quad \forall n = 1, \ldots, N.$$

It is straightforward to verify that the solution

$$\lambda_k^j = \frac{b_k}{\sum_{n=1}^{N} b_n}, \quad \forall k = 1, \ldots, N, \forall j \in J,$$

is feasible for $(P_4)$.

**Early termination.** The solution process for $(P_4)$ produces a feasible solution at every iteration. Thus, it is possible to terminate the process and move on to the next stage of disaggregation when the desired computing budget has been reached, or when the gap is sufficiently low (note that the sum of negative reduced costs of the newly added columns corresponds to the optimality gap of $(P_4)$). This is a desirable feature especially because the marginal cost improvement typically decreases over time, as we show empirically in Section 2.4.

### 2.3.4 Disaggregation into items

Stage C terminates with a feasible solution of $(P_4)$, denoted $(\lambda_k^j)_{j \in J, k \in K_j}$. The solution is potentially fractional, and represents a convex combination of columns in $\tilde{K}_j$ for each bundle $j \in J$. Using this as a starting point, our next step is to construct a
placement plan for each individual item.

To illustrate this disaggregation process, consider the hypothetical example of a bundle $j$ that consists of infinitely many identical items of infinitesimal demand. Then a naïve disaggregation approach is to assign column $k$ as the placement solution for a fraction $\lambda_j^k$ of items in the bundle. In other words, $\lambda_j^k$ of the items would take on the placement plan of $(x^i, y^i) = (x^i(k), y^i(k))$. Note that the objective cost of bundle $j$ in $(P_4)$ is a lower bound on the actual cost of the items, because the bundle’s fixed cost is the sum of fractional fixed costs of the columns, while each of the items are charged the full fixed cost. For this reason, the naïve disaggregation approach may not be optimal.

In reality, each bundle contains finitely many non-identical items of positive demand, and disaggregation is no longer a straightforward task. A good disaggregation approach should account for the following considerations:

- It may not be possible to split up a bundle $j$ exactly into fractions of $\lambda_j^k$;

- As explained above, the actual fixed cost of the items may be higher than that of the bundle, and is best modeled explicitly;

- Due to the heterogeneity of items, there is some flexibility and opportunity to match items with columns in a cost-minimizing manner;

- The overall capacity constraint needs to be satisfied;

- Last but not least, the disaggregation process should be computationally tractable.

With these considerations in mind, we propose the following method of disaggregation. First, we calculate the amount of capacity allocated to each bundle according to the column solution. Let

$$\mathcal{K}_j^+ := \{ k \in \mathcal{K}_j' \mid \lambda_j^k > 0 \}$$
be the set of active columns for bundle \( j \in J \), and let

\[
\Delta b_n = b_n - \sum_{j \in J} \sum_{k \in K_j^+} u_{k,n}^j \lambda_k^j, \quad n = 1, \ldots, N
\]

be the amount of unused excess capacity, i.e., the slack in constraints (2.26). We distribute this excess capacity proportionally to the bundles according to their demand, which results in bundle \( j \in J \) having an allocated capacity of

\[
w_n^j := \sum_{k \in K_j^+} u_{k,n}^j \lambda_k^j + \frac{d_j}{\sum_{j \in J} d_j} \Delta b_n.
\]

We then use the values of \( x_{nr}^i(k) \) and \( u_n^j \) as inputs for solving the following bundle-to-item disaggregation problem for every bundle \( j \in J \):

\[
\begin{align*}
\min_{\lambda^i, x^i, y^i} & \quad \sum_{i \in I_j} \sum_{n=1}^{N} \sum_{r=1}^{R} c_{nr}^i d_r^i x_{nr}^i + \sum_{i \in I_j} \sum_{n=1}^{N} f_i^i y_i^i \\
\text{subject to} & \quad x_{nr}^i = \sum_{k \in K_j^+} x_{nr}^i(k) \lambda_k^i, \quad \forall n = 1, \ldots, N, \forall r = 1, \ldots, R, \forall i \in I_j, \\
& \quad \sum_{k \in K_j^+} \lambda_k^i = 1, \quad \forall i \in I_j, \quad (2.30) \\
& \quad \sum_{i \in I_j} \sum_{r=1}^{R} d_r^i x_{nr}^i \leq u_n^j \quad \forall n = 1, \ldots, N, \quad (2.32) \\
& \quad \sum_{r=1}^{R} d_r^i x_{nr}^i \leq d^i y_i^i \quad \forall n = 1, \ldots, N, \forall i \in I_j, \quad (2.33) \\
& \quad \lambda_k^i \geq 0 \quad \forall i \in I_j, \forall k \in K_j^+, \\
& \quad y_i^i \in \{0, 1\} \quad \forall n = 1, \ldots, N, \forall i \in I_j,
\end{align*}
\]

where the decision variables are \( \lambda_k^i \), the fraction of item \( i \in I_j \) that uses column \( k \in K_j^+ \), and \( y_i^i \), whether or not item \( i \) uses fulfillment center \( n \), through any of the columns. Constraint (2.31) ensures that the fractional use of columns sum up to 1. (2.32) and (2.33) are the capacity and usage constraints, respectively, which resemble
(2.4) and (2.5) in the original item-based inventory placement problem (MIP).

The quantity $x_{ir}$, as defined by (2.30), is the fraction of item $i$ demand in region $r$ served by fulfillment center $n$, which has the same meaning as in (MIP). The definition of $y_n^i$ is also the same of that in (MIP). Therefore, any collection of feasible solutions of (D$_j$) for all $j \in J$ corresponds to a feasible solution of (MIP). This shows that we do not have to solve the disaggregation problem (D$_j$) to optimality. In fact, doing so may be unnecessarily time-consuming, since it is a mixed-integer program with $|I_j||K_j^+|$ continuous variables and $|I_j|N$ binary variables. Instead, we can terminate the process after a predetermined computing budget as long as a feasible solution is found.

2.3.5 Enhancement of item placement plans

Let $\{(\lambda_k^*, x_{kn}^*, y_n^i) \mid k \in K_j, n = 1, \ldots, N\}$ denote the solution of item $i \in I_j$ upon the termination of (D$_j$). While this solution is already feasible, we can further improve upon it by performing a second stage of re-optimizing each item.

For every $j \in J$, let

$$\Delta u_n^j := u_n^j - \sum_{i \in I_j} \sum_{r=1}^R d_r^i x_{ir}, \quad n = 1, \ldots, N$$

be the slack of (2.32), i.e., the remaining capacity of what has been allocated to bundle $j$. Again, we can distribute this evenly among the items and obtain the allocated capacity of

$$u_n^i := \sum_{r=1}^R d_r^i x_{ir} + \frac{\Delta u_n^j}{|I_j|}, \quad n = 1, \ldots, N.$$
Then the re-optimization problem for item $i$ is given by

$$\begin{align*}
\min_{x^i, y^i} & \quad \sum_{n=1}^{N} \sum_{r=1}^{R} c^i_{nr} d^i_r x^i_{nr} + \sum_{n=1}^{N} f^i_n y^i_n \\
\text{subject to} & \quad \sum_{n=1}^{N} x^i_{nr} = 1 \quad \forall r = 1, \ldots, R, \\
& \quad \sum_{r=1}^{R} d^i_r x^i_{nr} \leq u^i_n y^i_n \quad \forall n = 1, \ldots, N, \\
& \quad x^i_{nr} \geq 0, \quad y^i_n \in \{0, 1\} \quad \forall n = 1, \ldots, N, \forall r = 1, \ldots, R,
\end{align*}$$

(E$_i$)

This formulation is basically the part of (MIP) that correspond to item $i$, except that the capacity and usage constraints (2.4), (2.5) are replaced by (2.34), which captures both the requirement of staying within the allocated capacity $u^i_n$ of every fulfillment center $n$, as well as the relationship between $x^i_{nr}$ and $y^i_n$.

While (E$_i$) is a mixed-integer program, it is computationally tractable in practice because it only has $N$ binary variables and can be solved independently for each item. Moreover, the solution obtained from (D$_j$) can be used as the initial solution, which is typically not far from optimal. However, if solving a mixed-integer program for every item is undesirable, we can omit this re-optimization stage altogether, terminate it after a given computing budget, or remove the binary variables $y^i_n$ for the following alternative that is less aggressive in re-optimization:

$$\begin{align*}
\min_{x^i} & \quad \sum_{n=1}^{N} \sum_{r=1}^{R} c^i_{nr} d^i_r x^i_{nr} \\
\text{subject to} & \quad \sum_{n=1}^{N} x^i_{nr} = 1 \quad \forall r = 1, \ldots, R, \\
& \quad \sum_{r=1}^{R} d^i_r x^i_{nr} \leq \bar{u}^i_n \quad \forall n = 1, \ldots, N, \\
& \quad x^i_{nr} \geq 0 \quad \forall n = 1, \ldots, N, \forall r = 1, \ldots, R,
\end{align*}$$

89
where the fulfillment center capacity for every \( n = 1, \ldots, N \) is given by

\[
\bar{u}_n^i = \begin{cases} 
\sum_{r=1}^{R} d_{r} i^* + \frac{\Delta u_i^j}{\sum_{n=1}^{N} y_n^j}, & y_n^j = 1, \\
0, & \text{otherwise},
\end{cases}
\]

i.e., the slack capacities are only distributed across fulfillment centers that admit positive flow after the solution of \((D_j)\).

### 2.4 Empirical analysis

#### 2.4.1 Overview

In this section, we establish the effectiveness of the large-scale solution scheme proposed in Section 2.3 by applying it to large numerical examples, generated according to Section 2.1.2. We organize our experiments in three parts:

1. **Effect of problem size and aggregation method**: We run the large-scale solution scheme on 10 instances of \(|I| = 1000\) randomly generated items, under various combinations of bundle counts and aggregation methods. All cases produce near-optimal solutions within a few hours on a single processor. We study the effect of bundle size and aggregation method on the solution quality and computation time, and identify the combination that achieves the best performance.

2. **Solution quality, item characteristics, and the Sequential Placement Heuristic** For the placement plan produced by the large-scale solution scheme, we analyze the correlation between the solution quality and characteristics of an item. We observe that the placement plan of most items are similar to that of the uncapacitated problem, and the items that are impacted the most are the low-demand items. This inspires us to develop the *Sequential Placement Heuristic*, which places items one at a time in the order of decreasing demand. We compare the performance of the heuristic with our large-scale solution scheme.
3. **Effect of capacity constraint tightness**: Similar to the experiments for MIP, we study the performance of our large-scale solution scheme under various capacity constraint tightness settings, and study the effect on solution quality and computation time.

Since Stages A and B of the large-scale solution scheme can be pre-computed offline, we focus our analysis on the computation time and solution quality of Stages C, D and E. In our experiments, Stage C is allowed to run for as many iterations as needed to solve for the relaxed column-based formulation to optimality. Stage D is terminated after an optimization time of 5 minutes, which is sufficient for all cases to find a bundle-to-item disaggregation. Stage E, per-item re-optimization, is performed for all items with no time limits.

A lower bound on the optimal cost can be found by removing the capacity constraint, which allows the problem to be decomposed into the following uncapacitated *Simple Plant Location Problem (SPLP)* for each item $i \in \mathcal{I}$,

$$
\min_{x^i, y^i} \sum_{n=1}^{N} \sum_{r=1}^{R} c^i_{nr} d^i_r x^i_{nr} + \sum_{n=1}^{N} f^i_n y^i_n \quad \text{(SPLP}_i)\text{)}
$$

subject to

- $\sum_{n=1}^{N} x^i_{nr} = 1 \quad \forall r = 1, \ldots, R,$
- $x^i_{nr} \leq y^i_n \quad \forall n = 1, \ldots, N, \forall r = 1, \ldots, R,$
- $x^i_{nr} \geq 0 \quad \forall n = 1, \ldots, N, \forall r = 1, \ldots, R,$
- $y^i_n \in \{0, 1\} \quad \forall n = r, \ldots, N,$

where all variables and parameters have the same definition as in (MIP). Note that we have used the strong formulation, which provides the tightest LP relaxation bound, and is tractable since the problem only concerns a single item. For the size of our inventory network, Gurobi solves (SPLP$_i$) to optimality in 0.186 seconds per item.
2.4.2 Effects of problem size and aggregation method

Experiment design

We vary the number of bundles between the values of \(|J| \in \{1, 10, 100, 1000\}\), which produces problems of various sizes for each step of the large-scale solution scheme. We also consider the following two aggregation methods:

- **K-means**: Items are aggregated by performing K-means clustering, using the Euclidean distance, on normalized item parameters \(\left\{ \left( \frac{w^i - 1}{35 - 1}, \frac{d^i - 1}{100 - 1}, \frac{\alpha^i - 1}{1} \right) \right\}_{i \in J}\) (recall that \(w^i, d^i\) and \(\alpha^i\) are drawn from the uniform distribution on intervals \([1, 35]\), \([1, 100]\) and \([0, 1]\), respectively.) This creates bundles with similar items.

- **Random**: Items are assigned to bundles independently at random. The resulting bundles have similar characteristics on average, but the items in each bundle may be very different.

The combinations of bundle counts and aggregation methods result in a total of six ways of partitioning the items into bundles (note that \(|J| = 1\) and \(|J| = 1000\) each correspond to a unique partition regardless of the aggregation method). We refer to these as the six configurations of our large-scale solutions scheme. Each configuration is executed on 10 problem instances.

Our empirical analysis below is divided into four parts. First, we analyze the effect of problem size and aggregation method on the overall performance of the large-scale solution scheme. Then, we discuss the impact of parallel computing on the computation time for each stage. Finally, we provide further details on the optimization process in each stage, and assess the possibility and effect of terminating the computation in a shorter amount of time.

**Overall performance**

Table 2.6 summarizes the overall performance of the large-scale solution scheme, averaged over the 10 problem instances. (Detailed results for each instance are given
in Appendix A.2, Tables A.1, A.2 and A.3.) In this table, each column correspond to a configuration, and the rows are divided into three sections:

(a) **Upper bound on optimality gap**: This is the average percentage that the objective value after each stage is above the SPLP lower bound, i.e.,

\[
\text{Upper bound on optimality gap} = \frac{v_S - \underline{v}}{\underline{v}} \times 100\%,
\]

where \(v_S\) is the cost after Stage \(S\), with \(S = C, D\) or \(E\), and \(\underline{v}\) is the optimal cost of \((\text{SPLP}_i)\). Although this produces an upper bound rather than the actual optimality gap, it provides a reasonable way of aggregating results from different problem instances, each of which has a different optimal cost and SPLP lower bound.

(b) **Total computation time**: This is the time required to complete the computation of each stage on a single processor.

(c) **Number of iterations**: This is the average number of iterations required for solving the relaxed column-based formulation to optimality.

Table 2.7 lists the unit computation time, i.e., the average time it takes for Gurobi to solve one instance of an optimization problem. This is calculated from the information given in Table 2.6, as well as the breakdown of computation time in Stage C spent on the master problem \((\text{RMP})\) and column generation subproblems \((\text{CG}_j)\), which is given in Appendix A.2, Table A.4. Specifically,

- \(t_{\text{RMP}}\) is the average time per iteration for solving \((\text{RMP})\), obtained by summing up the total time spent on solving \((\text{RMP})\) across all instances (Table A.4(a)), and dividing it with the total number of iterations across all instances (Table 2.6(c) multiplied by 10, the number of instances).

- Similarly, \(t_{\text{CG}}\) is the average time per bundle, per iteration, for generating a column with \((\text{CG}_j)\); and

- \(t_E\) is the average time per item for solving \((\text{E}_i)\).
• Note that $t_D$ is the average time per bundle spent in Stage D, which includes both the formulation and optimization of $(D_j)$. The formulation time can be significant if there are a lot of items in each bundle, for example in the case of $|J| = 1$. The maximum time for optimizing $(D_j)$ is set to 300 seconds, or until the initial solution and gap are both found. Therefore, values of $t_D$ above 300 seconds reflects the time it takes to identify the initial solution and gap. On the other hand, values below 300 seconds indicate that some instances of $(D_j)$ terminated at optimality before the time limit. In particular, if the relaxed column solution for the bundle consists of only one column, then it is the unique solution of $(D_j)$, and thereby optimal.

Effects of $|J|$ on solution quality. Comparing the columns of Table 2.6(a) reveals the effect of $|J|$ on solution quality. (For $|J| = 10$ and 100, compare columns with the same aggregation method.) We see that the optimality gap decreases with increasing $|J|$. This is when there are fewer bundles, the bundle characteristics are generally a coarser representation of the item characteristics. This results in two effects: First, the relaxed column-based formulation is a worse approximation of the original problem, and therefore, the column solution produced by Stage C is farther from optimal. Secondly, the disaggregated item placement plan obtained in Stage D is more distinct from the column solution in Stage C, which is why there is a greater increase in optimality gap from Stage C to Stage D for fewer bundles.

Effect of $|J|$ on computation time. For computation time, Table 2.6(c) shows that with more bundles, fewer iterations are required to reach optimality in Stage C, possibly because more columns are added per iteration. Moreover, from Table (2.7), we see that the unit time for solving an instance of (RMP) increases with $|J|$, which is as expected, since the number of variables in (RMP) is proportional to $|J|$. However, the unit computation time for (CG$_j$) decreases with increasing $|J|$, indicating that the column generation subproblems become easier to solve, on average, when there are many bundles. Further investigation of this observation is given in Section 2.4.2.
For Stages D and E, recall that when there are more bundles, each bundle contains fewer items. Therefore, each bundle-to-item disaggregation problem is easier to solve, and the resulting placement plan is also closer to optimal for the individual items. This is reflected by $t_D$ and $t_E$ both decreasing with increasing $|J|$ in Table (2.7).

**Effect of aggregation method on solution quality and computation time.**

For $|J| = 10$ and 100, the aggregation method also has a significant impact on the performance. *K-means* appears to be a better aggregation method than *Random*, providing better quality solutions in a shorter amount of time.

*Random* also requires more iterations to find the optimal column solution, possibly because there is greater homogeneity between the bundles. (This may be related to the idea that the simplex method for solving a linear program generally requires more iterations to converge in face of degeneracy, i.e., when at a basic feasible solution, we can exchange a basic variable with a non-basic variable and obtain another basic feasible solution with the same cost, so that moving from one basis to another does not improve the objective cost. Bundles with similar characteristics can create such distinct basic feasible solutions that have the same cost.)

From these observations, we conclude that $|J| = 100$ with *K-means* aggregation achieves a good trade-off between computation time and solution quality. This configuration shall serve as the base case for other experiments in Sections 2.4.3 and 2.4.5.

**Time reduction with parallel computing**

The unit computation time given in Table 2.7 provides a way to estimate the time savings when parallel computing resources are available. Recall that $(CG_j)$ and $(D_j)$ can be solved concurrently for all bundles, and that $(E_i)$ for all items. Therefore, given up to $|J|$ parallel processors, the time spent on $(CG_j)$, $(D_j)$ and $(E_i)$ can be reduced by a factor roughly equal to the number of processors.

As an example, if we execute the configuration of $|J| = 100$ with *K-means* on 10 parallel processors, then each iteration of Stage C takes $0.75 + 1.98 \times \frac{|J|}{10}$ seconds on
Table 2.6: Performance of large-scale solution scheme
(|\mathcal{I}| = 1000, average of 10 instances)

| Stage | |J| = 1 | |J| = 10 | |J| = 10 | |J| = 100 | |J| = 100 | |J| = 1000 |
|-------|-----------------|--------|--------|--------|--------|--------|--------|
|       | kmeans | random | kmeans | random | kmeans | random | kmeans | random | kmeans | random | kmeans | random | kmeans | random | kmeans | random | kmeans | random | kmeans | random | kmeans | random | kmeans | random |
|       |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| (a) Upper bound on optimality gap (%) |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| C     | 2.04   | 1.13   | 1.99   | 0.80   | 1.78   | 0.74   |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| D     | 11.12  | 1.15   | 2.00   | 0.86   | 1.85   | 0.79   |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| E     | 7.68   | 1.12   | 1.90   | 0.84   | 1.76   | 0.78   |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| (b) Total computation time (seconds) |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| C     | 1394.32 | 636.15 | 3892.47 | 1520.99 | 6659.45 | 13616.70 |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| D     | 380.08  | 2412.10 | 2966.58 | 3339.63 | 3099.63 | 18.80  |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| E     | 2838.97 | 320.90 | 419.81 | 205.63 | 390.72 | 181.38 |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| Total | 4613.37 | 3369.15 | 7278.86 | 5066.25 | 10149.80 | 13816.88 |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| (c) Number of iterations |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| C     | 597.7  | 159.3  | 259.0  | 76.4  | 127.0  | 67.8  |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |

Table 2.7: Unit computation time of each stage
(|\mathcal{I}| = 1000, average of 10 instances)

| Stage | |J| = 1 | |J| = 10 | |J| = 10 | |J| = 100 | |J| = 100 | |J| = 1000 |
|-------|-----------------|--------|--------|--------|--------|--------|--------|
|       | kmeans | random | kmeans | random | kmeans | random | kmeans | random | kmeans | random | kmeans | random | kmeans | random | kmeans | random | kmeans | random | kmeans | random | kmeans | random |
|       |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| t_{RMP} | 0.002  | 0.01   | 0.01   | 0.08   | 0.13   | 1.71   |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| t_{CG} | 2.33   | 0.40   | 1.50   | 0.20   | 0.52   | 0.20   |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| t_{D}  | 380.08 | 241.21 | 296.66 | 33.40  | 31.00  | 0.02   |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| t_{E}  | 2.84   | 0.32   | 0.42   | 0.21   | 0.39   | 0.18   |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
average, and the expected completion time of Stage C is approximately

\[
\left( 0.08 + 0.20 \times \frac{100}{10} \right) \times 76.4 = 158.91 \text{ seconds.}
\]

Similarly, Stage D takes about \(33.4 \times \frac{|\mathcal{J}|}{10} = 334\) seconds to complete, and Stage E takes about \(0.21 \times \frac{|\mathcal{J}|}{10} = 21\) seconds. The overall completion time is around 513.91 seconds, or less than 9 minutes, which is roughly a ten-fold speedup compared to the single-processor setting.

In conclusion, having more parallel processors reduces the computation time, and also enables working with a larger \(|\mathcal{J}|\), which generally improves the solution quality. However, depending on the implementation, parallel computing may come with a certain amount of overhead (for example, additional time and memory from having to send data back and forth between processors). In addition, a larger \(|\mathcal{J}|\) also results in a slight increase in \(t_{RMP}\), which is not parallelizable. Therefore, the appropriate number of parallel processors and number of bundles need to be chosen carefully with all the above considerations in mind.

**Performance of each stage**

We now provide detailed results on how the solution quality progresses over time, in each stage of the large-scale solution scheme. Our main finding is that there is diminishing returns in solution quality improvement when given more computation time. This implies that we can terminate the stages early without severely impacting the final solution quality, and we provide guidelines for doing so.

**Solution quality over iterations of \((RMP)\).** Figure 2.12 shows the convergence gap improvement over the iterations of \((RMP)\), where the *convergence gap* is defined as

\[
\text{Convergence gap} = \frac{v_t - v^*}{v^*} \times 100\%,
\]
where $v_t$ is the objective cost in the current iteration $t$, and $v^*$ is the optimal objective cost of (RMP) for the same problem instance and configuration. The same information is summarized in a different format in Table 2.8, which reports the average number of iterations required to close the convergence gap by a certain percentage, averaged over the 10 problem instances for each configuration.

These results indicate that solution quality improvement is concentrated in the first one-third to half of the iterations, after which the objective cost is very close to the optimal solution of (RMP), but absolute optimality takes a long time to achieve. Therefore, when computation time is limited, we may wish to terminate Stage C before (RMP) is solved to optimality.

There are many possible early termination criteria for Stage C. A simple choice is to limit the number of iterations. However, as shown in Figure 2.12 and Table 2.8, the appropriate choice of iteration limit depends on the configuration and the problem.

Another option is to set a threshold for the relative improvement in objective cost,

$$\text{Relative improvement} = \frac{v_{t-1} - v_t}{v_t} \times 100\%,$$

where, again, $v_t$ is the objective cost in iteration $t$. Figure 2.13a shows the relative improvement ($y$-axis) and the convergence gap ($x$-axis) at every iteration and problem instance, for each of the six configurations. Since there is a significant correlation between the two measures, we can use the relative improvement (which can easily be deduced from available information) as a termination criteria to obtain solutions with a good convergence gap (which is not observable, since we do not know the true optimal objective $v^*$). For example, for our problem, if we terminate Stage C as soon as the relative improvement of (RMP) is below 0.01%, then we will generally obtain a solution that is within 0.1% optimal of (RMP), regardless of the configuration and problem instance. There is one caveat though: the relative improvement may

\[18\] The convergence gap is distinguished from the optimality gap—upon convergence, the optimal solution of (RMP) is different from the optimal solution of the inventory placement problem (MIP), and is also distinct for each configuration, i.e., choice of $|\mathcal{J}|$ and aggregation method.
oscillate over iterations, although it generally follows a decreasing trend. This is shown in Figure 2.13b. Because of this, a simple threshold check may lead to premature termination. To overcome this caveat, we can consider the relative improvement across recent iterations (rather than only for the current iteration), e.g., take the running average of relative improvement, or terminate only when a certain number of consecutive iterations have met the threshold.

**Computation time over iterations of \((CG_j)\).** In Table 2.7, we observe that \(t_{CG}\) decreases with increasing \(|J|\). We now provide further diagnosis of the cause.

Figure 2.14 plots the unit computation time (per bundle) of the column generation subproblem \((CG_j)\) for each iteration. In this figure, each gray line represents the iterations of a problem instance under a configuration, and the red lines are the average of 10 instances for each configuration. The figure reveals that the column generation problems actually start out with a similar unit computation time for all configurations (around 0.2 seconds). As the iterations go on, they take progressively longer, reflecting that the column generation subproblems become more and more difficult to solve. The increase in difficulty is more significant for smaller \(|J|\), and more significant for the Random aggregation method than for K-means.

This provides another reason for terminating Stage C early. As the iterations grow, not only do the cost improvements for \((RMP)\) become less significant, as we saw above, but the \((CG_j)\) problems also take longer to solve. In other words, towards the end of Stage C, there is diminishing returns in solution quality for the investment of computation time.

**Initial solution of \((D_j)\).** In Stage D of the experiment, we arbitrarily set the optimization time limit to 300 seconds (i.e., to terminate the optimization of \((D_j)\) upon the first solution or bound update after 300 seconds). We now show that it may in fact be sufficient to terminate it even earlier, upon finding the first initial feasible solution.
Figure 2.12: In Stage C, most of the progress in (RMP) is made in the initial iterations.

1000 items, 10 instances (each line represents the iterations of one problem instance).

Table 2.8: Number of (RMP) iterations required to reach a certain convergence gap

| Gap  | $|\mathcal{J}| = 1$ | $|\mathcal{J}| = 10$ | $|\mathcal{J}| = 10$ | $|\mathcal{J}| = 100$ | $|\mathcal{J}| = 100$ | $|\mathcal{J}| = 1000$ |
|------|------------------|------------------|------------------|------------------|------------------|------------------|
|      | kmeans           | random           | kmeans           | random           | kmeans           | random           |
| 100% | 36.9             | 12.4             | 29.3             | 7.8              | 17.8             | 7.1              |
| 10%  | 92.8             | 36.0             | 71.1             | 23.4             | 43.1             | 19.4             |
| 1%   | 139.6            | 55.4             | 104.9            | 35.9             | 64.6             | 30.9             |
| 0.1% | 240.2            | 82.5             | 150.8            | 49.1             | 86.7             | 42.0             |
| 0%   | 597.7            | 159.3            | 259.0            | 76.4             | 127.0            | 67.8             |
Figure 2.13: Relative improvement is correlated with convergence gap and can be used as a termination criteria for (RMP).

(a) Relative improvement versus convergence gap

Each dot represents an iteration of a problem instance under a configuration.

(b) Relative improvement over iterations

Each line represents the iterations of a problem instance under a configuration.
Figure 2.14: The column generation subproblems ($\text{CG}_j$) take longer to compute as the iterations go on.

Table 2.9 reports the computation time and convergence gap\textsuperscript{19} in Stage D, for both the initial feasible solution, and the final solution obtained after about 300 seconds. The “Mean” and “Max” rows represent, respectively, the average and maximum gap or time across the bundles of each problem instance, which is then averaged over the 10 instances.

According to this table, the configuration of $|\mathcal{J}| = 1$ generally requires more than 300 seconds to find the initial feasible solution, and the initial solution quality is far from being optimal for ($D_j$). However, for all other configurations, the initial solution is usually found very quickly, and the gap is small. Therefore, in these cases, we can save time without sacrificing too much on solution quality by terminating Stage D as soon as an initial feasible solution for ($D_j$) is found.

Note that for the configuration of $|\mathcal{J}| = 1000$ in the last column, the initial and final time and gap are all 0 (after rounding to 2 digits). Figure 2.15 sheds light on why this is the case, by plotting the histogram of the number of bundles that have a certain number of columns in the solution produced by Stage C, which is a convex combination of columns, and serves as input to Stage D. For $|\mathcal{J}| = 1000$,

\textsuperscript{19}This is the gap relative to the optimal solution of bundle-to-item disaggregation problem ($D_j$).
most bundles have only one column in the solution, and therefore, \((D_j)\) is trivially solved— the column corresponds to a placement plan that is assigned to every item in the bundle.

### 2.4.3 Solution quality and item characteristics

In this section, we investigate properties of the final placement plan produced by the top-performing configuration, \(|\mathcal{I}| = 100\) with \(K\)-means aggregation. In particular, we would like to see if there are types of items that tend to be stocked suboptimally, compared to their uncapacitated placement plan produced by SPLP. Such an understanding could inform the development of other efficient placement heuristics.

As proxies for item features, we focus on the three parameters for random item generation described in Section 2.1.2, namely, the total demand, geographical distribution \(\alpha\), and weight. Figure 2.16 shows the relationship between the optimality gap (y-axis) and the three parameters (x-axis) of each item. In this figure, each item is represented by a dot that is color-coded by problem instance, and the per-item optimality gap is given by

\[
\text{Optimality gap of item } i = \frac{\hat{v}^i - v^i}{\hat{v}^i} \times 100%,
\]

with \(v^i\) being the cost of the item in the placement plan produced by our method.
Figure 2.15: When the number of bundles is large (e.g., $|J| = 100$ or $1000$), most bundles have no more than a few columns in the solution of Stage C. In those cases, disaggregation in Stage D is easy, and for a single column, disaggregation is trivial.

1000 items, total of 10 instances

and $v^i$ the cost of the SPLP placement plan.

From Figure 2.16, we see that there is a significant correlation between the total demand of the item and its optimality gap, but not for the other two parameters. Specifically, the large-scale solution scheme appears to achieve near-optimality by placing very low-demand items suboptimally, while other items are placed almost optimally. There is no intrinsic design in the large-scale solution scheme that accounts for this phenomenon, although it makes sense intuitively somewhat, as low-demand items contribute to a smaller fraction of the objective cost, and therefore the objective is less sensitive to their cost suboptimality.

2.4.4 A sequential placement heuristic

The results in the previous subsection suggest the viability of a sequential placement heuristic for solving the original problem at the item level. The heuristic is stated formally in Algorithm 2.4.1. It first sorts the items in decreasing order of demand, and then places them one at a time with respect to the remaining fulfillment cen-
Figure 2.16: The large-scale solution scheme tends to concentrate the suboptimality of placement on low-demand items.
ter capacities \( \{b_n\}_{N=1}^N \). The placement plan is obtained by solving the single-item Capacitated Plant Location Problem (CPLP),

\[
\begin{align*}
\min_{x^i, y^i} & \quad \sum_{n=1}^N \sum_{r=1}^R c_{nr}^i d_{nr}^i x_{nr}^i + \sum_{n=1}^N f_n^i y_n^i \\
\text{subject to} & \quad \sum_{n=1}^N x_{nr}^i = 1 \quad \forall r = 1, \ldots, R, \\
& \quad \sum_{r=1}^R d_{nr}^i x_{nr}^i \leq \bar{b}_n \quad \forall n = 1, \ldots, N, \\
& \quad x_{nr}^i \leq y_n^i \quad \forall n = 1, \ldots, N, \forall r = 1, \ldots, R, \\
& \quad x_{nr}^i \geq 0 \quad \forall n = 1, \ldots, N, \forall r = 1, \ldots, R, \\
& \quad y_n^i \in \{0, 1\} \quad \forall n = r, \ldots, N,
\end{align*}
\]  

(CPLP_i)

The formulation is identical to the uncapacitated Simple Plant Location Problem (SPLP_i) given in Section 2.4.1, except for additional capacity constraints in the third line. The remaining capacities \( \{b_n\}_{N=1}^N \) are updated after placing each item.

The sequential placement heuristic attempts to mimic the behavior of concentrating suboptimal placement on low-demand items, which the large-scale solution scheme exhibits. By proceeding in the order of decreasing demand, we expect high-demand items to be placed similarly as the (SPLP_i) placement plan. Capacity constraints only become binding as we progress toward items of lower demand.

Each instance of (CPLP_i) is relatively easy to compute. However, unlike our large-scale solution scheme, this heuristic does not benefit from parallel computing due to its sequential nature.

We test the sequential placement heuristic on the same 10 instances of 1,000 items. The average upper bound on the overall optimality gap is 2.18\%. This is worse than the best configuration of the large-scale solution scheme, and even though the difference is only slightly more than one percent, the potential savings can be huge in light of the fact that online-retail can be a billion-dollar business. However, the sequential placement heuristic only takes an average of 199.72 seconds to compute (i.e., 0.2 seconds per item), which makes it a competitive alternative, especially if
parallel computing resources are not available for speeding up the large-scale solution scheme.

It is worth pointing out that the sequential placement heuristic does not actually achieve the intended behavior shown in Figure 2.16, which may account for why the overall gap is worse than that of the large-scale solution scheme. Instead of concentrating suboptimal placement on low-demand items, it places many medium-to low-demand items suboptimally, as depicted in Figure 2.17. Moreover, the “humps” of rapid increase in suboptimality correspond to the situation where fulfillment centers run out of capacity. However, for items of extremely low demand, the placement is again near-optimal.

Although the sequential placement heuristic achieves a reasonable cost for our numerical example, we remark that such a greedy heuristic can be arbitrarily suboptimal. Consider a small example with \( N = 2 \) fulfillment centers, \( R = 1 \) region, and two items \( I = \{1, 2\} \). Suppose that each fulfillment center has capacity \( 1 + \epsilon \), where \( \epsilon > 0 \), and that the fixed costs are all zero. Suppose also that the items have a demand of \( d_1 = 1 + \epsilon, d_2 = 1 \), and that the shipping costs are \( (c_{11}^1, c_{21}^1) = (1, 2) \) for Item 1, and \( (c_{11}^2, c_{21}^2) = (1, C) \) for Item 2, where \( C > 2 \) is a large number. In other words, it is better to ship both items from the first fulfillment center, and in the case that they need to be shipped from the second fulfillment center, the additional cost for Item 2 is greater than that of Item 1. However, this is not taken into account by the sequential placement heuristic. It simply starts with Item 1, which has the greatest demand, and places it fully in the first fulfillment center; then, it places Item 2 fully in the second fulfillment center. This results in a total cost of \( 1 + \epsilon + C \). A better solution would be to place Item 2 fully in fulfillment center 1, along with \( \epsilon \) units of Item 1, and place the remaining 1 unit of Item 1 in fulfillment center 2, for a total cost of \( 3 + \epsilon \). The former cost could be arbitrarily worse than the latter, depending on the value of \( C \).
Figure 2.17: The sequential placement heuristic places items of medium-to-low demand suboptimally.

(a) Optimality gap versus demand for all 10 instances

(b) When placing items from high to low demand (i.e., moving from the right of the figure to the left), the rapid increase in suboptimality corresponds to occasions where fulfillment centers of desirable locations run out of capacity, as indicated by the red lines. (Other instances exhibit a similar behavior.)
**Algorithm 2.4.1** Sequential placement heuristic

1: \((i_1, i_2, \ldots, i_{|\mathcal{I}|}) \leftarrow \text{indices in } \mathcal{I} \text{ sorted in decreasing order of } d^i
2: \bar{b}_n \leftarrow b_n \quad \forall n = 1, \ldots, N \quad \Rightarrow \text{Fulfillment center capacity constraints}
3: \textbf{for } i \leftarrow i_1, i_2, \ldots, i_{|\mathcal{I}|} \textbf{ do}
4: \quad \text{Solve } (\text{CPLP}_i)
5: \quad \textbf{for all } n \leftarrow 1, \ldots, N \textbf{ do}
6: \quad \quad \bar{b}_n \leftarrow \bar{b}_n - \sum_{r=1}^R d_r^i x_{nm}^i \quad \Rightarrow \text{Remaining capacity}
7: \quad \textbf{end for}
8: \textbf{end for}
9: \textbf{return} \text{ Placement plans } \{(x^i, y^i)\}_{i \in \mathcal{I}}

2.4.5 Effects of capacity constraint tightness

To see how the tightness of capacity constraints affect the solution quality and computation time, we vary the amount of slackness in total capacity from the default 10\% for previous experiments to values of 5\%, 20\% and 50\%. For each capacity, we solve the same 10 sets of 1,000 items as before, using the large-scale solution scheme (with the most favorable configuration of \(|\mathcal{J}| = 100 \text{ and } K\text{-means aggregation}\)) and the sequential placement heuristic.

The resulting performance of the large-scale solution scheme is summarized in Table 2.10. It shows that tighter capacity constraints correspond to higher costs and longer computation times. The increase in computation time comes from a greater numbers of Stage C iterations, as well as increased unit computation time \(t_{RMP}\) and \(t_D\), as the slack decreases.

Figure 2.18 compares the optimality gap under various levels of capacity constraint slackness, averaged over the 10 instances, for our large-scale solution scheme (after both Stages D and E, whose costs only differ slightly) and for the sequential placement heuristic. Two observations can be drawn from the figure. First, for both methods, the objective cost decreases at a decreasing rate with increasing capacity slackness. Secondly, the sequential placement heuristic appears to be more sensitive to capacity constraint slackness than the large-scale solution scheme; this is seen from the fact that the difference in the gap of the two methods increases as capacity constraints
become tighter.

2.4.6 Effects of the fixed cost parameter

We now investigate the effect of changing the fixed cost parameter $f^i_n$. In addition to the default of $f^i_n = 1.0$, we also test the values of $f^i_n = 10$ and 100, each for 10 randomly generated sets of items, under the configuration of $|J| = 100$ with K-means aggregation.

Table 2.11 and Figure 2.19 compare the solution of the three fixed cost parameters, in terms of the costs and the number of fulfillment centers that hold each item. These results show that, as expected, the magnitude of $f^i_n$ has a direct impact on the number of fulfillment centers that holds each item. A larger fixed cost parameter leads to each item using fewer fulfillment centers, and in turn, the shipping cost increases. In other words, $f^i_n$ can be used to control the sparsity of the placement plan.

Note also that for the default case of $f^i_n = 1.0$, the total fixed cost is relatively low compared to the shipping cost. However, the majority of the items are stocked in 10 or fewer of the 88 fulfillment centers, which appears relatively few. This indicates that even without fixed costs, it may not be necessary to utilize every fulfillment center; instead, it may be sufficient to concentrate on a few that are in desirable locations.

Table 2.11: Solutions of various fixed cost values
  ($|I| = 1000$, average of 10 instances)

<table>
<thead>
<tr>
<th>$f^i_n$</th>
<th>Cost</th>
<th>Breakdown</th>
<th>Number of fulfillment centers per item</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Shipping</td>
<td>Fixed</td>
</tr>
<tr>
<td>1</td>
<td>541.42</td>
<td>530.08</td>
<td>11.34</td>
</tr>
<tr>
<td>10</td>
<td>597.59</td>
<td>553.5</td>
<td>44.09</td>
</tr>
<tr>
<td>100</td>
<td>836.44</td>
<td>614.14</td>
<td>222.3</td>
</tr>
</tbody>
</table>

2.4.7 Scaling up

Setup. To demonstrate the scalability of our large-scale solution scheme, we solve 10 problem instances with one million randomly generated items, and compare its performance with the sequential placement heuristic.
Figure 2.18: As capacity constraints become tighter, the cost increases at an increasing rate, and the optimality gap of our method outperforms the sequential placement heuristic more significantly.

Table 2.10: Performance of large-scale solution scheme for various levels of capacity constraint tightness ($|\mathcal{I}| = 1000$, average of 10 instances)

<table>
<thead>
<tr>
<th>Stage</th>
<th>Slack 5%</th>
<th>Slack 10%</th>
<th>Slack 20%</th>
<th>Slack 50%</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) Upper bound on optimality gap (%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0.89</td>
<td>0.8</td>
<td>0.66</td>
<td>0.44</td>
</tr>
<tr>
<td>D</td>
<td>0.96</td>
<td>0.86</td>
<td>0.72</td>
<td>0.47</td>
</tr>
<tr>
<td>E</td>
<td>0.94</td>
<td>0.84</td>
<td>0.70</td>
<td>0.45</td>
</tr>
<tr>
<td>(b) Total computation time (seconds)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>1572.87</td>
<td>1520.99</td>
<td>1275.7</td>
<td>999.9</td>
</tr>
<tr>
<td>D</td>
<td>3877.72</td>
<td>3339.63</td>
<td>2846.88</td>
<td>1612.87</td>
</tr>
<tr>
<td>E</td>
<td>196.12</td>
<td>205.63</td>
<td>253.87</td>
<td>416.32</td>
</tr>
<tr>
<td>Total</td>
<td>5646.71</td>
<td>5066.25</td>
<td>4376.45</td>
<td>3029.09</td>
</tr>
<tr>
<td>(c) Number of iterations</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>81.6</td>
<td>76.4</td>
<td>62.8</td>
<td>46.1</td>
</tr>
<tr>
<td>(d) Unit computation time of each stage</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_{RMP}$</td>
<td>0.09</td>
<td>0.08</td>
<td>0.07</td>
<td>0.05</td>
</tr>
<tr>
<td>$t_{CG}$</td>
<td>0.19</td>
<td>0.2</td>
<td>0.2</td>
<td>0.22</td>
</tr>
<tr>
<td>$t_D$</td>
<td>38.78</td>
<td>33.40</td>
<td>28.47</td>
<td>16.13</td>
</tr>
<tr>
<td>$t_E$</td>
<td>0.20</td>
<td>0.21</td>
<td>0.25</td>
<td>0.42</td>
</tr>
</tbody>
</table>
Figure 2.19: When the unit fixed cost is larger, fewer fulfillment centers are used, and the shipping cost increases.

(a) Distribution of the number of fulfillment centers used by the 1000 items

(b) Cost breakdown

Average of 10 instances for each fixed cost parameter
For each instance, the 1,000,000 randomly generated items are aggregated into 1,000 bundles of 1,000 items each. To avoid running K-means on such a huge dataset, we devise the following workaround: The range of the three parameters (demand, geographical distribution $\alpha$, and weight) are each divided into 10 intervals. This creates a total of 1,000 combinations of parameter intervals, and we randomly generate 1,000 items for each combination. The resulting bundles contain similar items, much like the effect of performing K-means.

In our large-scale solution scheme, we allow Stage C (column-based reformulation) to be solved to optimality. Stage D (disaggregation) is performed until the first feasible solution is found, i.e., it is terminated as soon as a feasible placement plan for each item has been constructed from the bundle’s column-based solution produced by Stage C. We omit Stage E because exploratory results indicate that it produces a relatively small improvement for the amount of time it takes; most of its computation time is spent on proving the optimality of the item’s placement plan.

**Overall comparison of methods.** Table 2.12 reports the optimality gap from the lower bound, as well as the computation time, averaged over the 10 instances. The lower bound on the optimality gap is again computed by solving the uncapacitated simple plant location problem (SPLP), which takes a total of 43.59 hours, or an average of 0.16 seconds per item.

The table shows that our large-scale solution scheme outperforms the sequential placement heuristic for the large problem instance. The average optimality gap is 0.796% for the former compared to 2.21% for the latter. This is similar to the previous example of 1,000 items. However, our method only takes 3.14 hours on average, whereas the sequential placement heuristic requires 46.05 hours (or 0.17 seconds per item). Recall also that the computation time for our method can be further reduced by parallel computing, which is not true for the sequential placement heuristic.\(^{20}\)

\(^{20}\)While beyond the scope of this text, we acknowledge that there may be more sophisticated variants of the sequential placement heuristic that would lend itself well to parallel computing. For example, with 10 parallel processors, we would likely find a similar solution by solving 10 items at a time, independent of each other, with $\frac{1}{10}$ of the available capacity allocated to each item.
Table 2.12: Performance of large-scale solution scheme and sequential placement heuristic for an instance of 1 million items

<table>
<thead>
<tr>
<th>Method</th>
<th>Optimality gap (%)</th>
<th>Computation time (hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large-scale solution scheme</td>
<td>0.796</td>
<td>3.14</td>
</tr>
<tr>
<td>Sequential placement</td>
<td>2.21</td>
<td>46.05</td>
</tr>
</tbody>
</table>

**Details for the large-scale solution scheme.** Further analysis of the large-scale solution scheme shows that Stage C takes up 3 hours of the computation time, while Stage D takes only 9.25 minutes on average. Stage C terminates after an average of 64.6 iterations. Only 0.8% of the time for Stage C is spent on solving the master problem (RMP), i.e., 1.34 seconds per iteration. The bulk of the time is spent on the column generation subproblems (CG), which takes an average of 165.15 seconds for all bundles (or 0.0165 seconds per bundle) per iteration.

Thus, if parallel processors were employed to solve the column generation subproblems and the disaggregation problems for each bundle, the computation time would be reduced by a factor roughly equal to the number of parallel processors (up to the number of bundles).

Most of the bundles only utilize one column in the solution of Stage C, for which disaggregation is trivial. Figure 2.20 shows the histogram of the number of columns each bundle uses. This is similar to the behavior shown in Figure 2.15 for 1,000 items and 1,000 bundles. As a result, the disaggregation problem in Stage D is trivial for most bundles.

**Details for the sequential placement heuristic.** For one instance of a million items, Figure 2.21 shows the optimality gap for each item, where the x-axis corresponds to the item’s demand. The behavior is similar to that shown in Figure 2.17 for 1,000 items.
Figure 2.20: For 1,000,000 items aggregated into 1,000 bundles, most bundles only use one column.

Figure 2.21: The sequential placement heuristic places the first 185 items (with the highest demand) optimally, but the cost for the remainder of the items can be as high as more than 6 times the item’s optimal placement.
2.5 Conclusions

2.5.1 Summary of contributions

The contributions of this chapter can be summarized as being related to modeling, methods, and managerial implications.

For modeling, we formulate the online-retail inventory placement problem as a mixed-integer program that is general enough to capture different types of costs and constraints. We propose a large-scale solution scheme for the problem, which uses item aggregation and column generation as modeling techniques for breaking down the problem into tractable stages.

For methods, the large-scale solution scheme is an optimization method that is capable of handling extremely large mixed-integer programs. The computationally intensive stages are decomposable, and can thus take advantage of parallel computing. Moreover, it can trade-off the solution quality with the computing budget, as each stage allows for early termination with a feasible solution. The effectiveness of the solution scheme is demonstrated with large-scale numerical examples.

For managerial implications, the large-scale solution scheme is a powerful tool for assessing the impact of various factors. Our empirical analysis reveals the effect of several factors on the solution quality, including the aggregation method, item characteristics, capacity constraint tightness, and fixed cost. These insights can be used to guide the design of the fulfillment center network, as well as the choice of problem parameters to shape the placement plans obtained by the large-scale solution scheme.

2.5.2 Future directions

The primary direction of future work is to extend the large-scale solution scheme to other mixed-integer programs that have similar structures, within and beyond the context of online-retail inventory management. We illustrate the possibility of such extensions with a related problem in online retail, namely, the problem of sparsity-
constrained inventory placement.

Instead of using fixed costs to induce sparsity in the placement plan, consider the alternative of imposing sparsity constraints on the fulfillment center usage variables \( y^i = (y^i_n)^N_{n=1}, i \in I \):

\[
\begin{align*}
\min_{x,y} & \quad \sum_{i \in I} \sum_{n=1}^N \sum_{r=1}^R c_{nr}^i d_r^i x_{nr}^i \\
\text{subject to} & \quad \sum_{n=1}^N x_{nr}^i = 1 \quad \forall r = 1, \ldots, R, \forall i \in I, \\
& \quad \sum_{i \in I} \sum_{r=1}^R d^i_r x_{nr}^i \leq b_n \quad \forall n = 1, \ldots, N, \\
& \quad \sum_{r=1}^R d^i_r x_{nr}^i \leq d^i y_n^i \quad \forall n = 1, \ldots, N, \forall i \in I, \\
& \quad (y^i)_{i \in I} \in S \\
& \quad x_{nr}^i \geq 0 \quad \forall n = 1, \ldots, N, \forall r = 1, \ldots, R, \forall i \in I, \\
& \quad y_n^i \in \{0, 1\} \quad \forall n = 1, \ldots, N, \forall i \in I,
\end{align*}
\]

(2.35)

where everything is the same as in \((\text{MIP})\) except that the fixed costs in the objective is replaced by constraints (2.35). The set \( S \) can represent different types of sparsity constraints, i.e., limits on the number of fulfillment centers that carry an item.

Using (2.35) in lieu of fixed costs has the potential benefit that the set \( S \) may be easier to characterize than the fixed cost parameters in certain business contexts. However, it also has the drawback that the presence of both sparsity and capacity constraints may result in the problem being infeasible, even if the total capacity exceeds the total demand. There appears to be no simple way to check feasibility. One technique for ensuring feasibility is to add a virtual fulfillment center that is uncapacitated and does not appear in sparsity constraints, but has a high shipping cost to all regions. Then, any flow that would not fit in the actual fulfillment centers would be redirected to the virtual one, and the resulting solution could be used to diagnose the cause of infeasibility.

Below, we describe two exemplary types of sparsity constraints that lead to dif-
different problem structures, and discuss how to adapt the large-scale solution scheme to each case.

**Sparsity limits per item.** Due to considerations such as replenishment location restrictions imposed by suppliers, an online retailer may wish to limit the number of fulfillment centers carrying an item. This can be modeled by writing (2.35) as

$$\sum_{n=1}^{N} y_{n}^{i} \leq s_{i} \quad \forall i \in \mathcal{I},$$

where the sparsity limit \( s_{i} \) for each item \( i \in \mathcal{I} \) is given as input.

When performing aggregation and bundle characterization, the sparsity limit \( s_{j} \) of the bundle \( j \in \mathcal{J} \) should be taken as the minimum sparsity limit among its items, \( \min_{i \in \mathcal{I}_{j}} s_{i} \), so as to ensure the feasibility of the disaggregated placement plans for each item. For this reason, it may be a good idea to aggregate the items according to their sparsity limits, in addition to other criteria for aggregation.

The column-based reformulation of this problem has the same reduced master problem (RMP) as before, while the constraint \( \sum_{n=1}^{N} y_{n}^{j} \leq s_{j} \) is added to the column-generation subproblem (CG_j). The resulting column solution is a convex combination of columns that satisfy the sparsity constraint.

The disaggregation problem is an integer program that selects exactly one column for each item, which is different from the previous formulation that selects a convex combination of columns with fixed cost adjustments.

**Sparsity limits per fulfillment center.** A fulfillment center may also have a limit on the number of items it can carry, especially if the storage space is organized by “bins” that must each carry a unique item. In this case, we can write (2.35) as

$$\sum_{i \in \mathcal{I}} y_{n}^{i} \leq t_{n} \quad \forall n = 1, \ldots, N,$$

where the sparsity limit \( t_{n} \) for each fulfillment center \( n \) is given as input.

Unlike the previous example, these sparsity constraints cannot be decomposed
by items, but rather create a coupling of items. Therefore, they do not need to be
considered in aggregation and bundle characterization, but must be included in the
reduced master problem along with the capacity constraints. One possible approach
is to add the following constraint to \((\text{RMP})\):

$$
\sum_{j \in \mathcal{J}} \sum_{k \in \mathcal{K}_j} |\mathcal{I}_j| y_{n}^j(k) \lambda_k^j \leq t_n \quad \forall n = 1, \ldots, N,
$$

where \( |\mathcal{I}_j| y_{n}^j(k) \) is equal to the total number of items in the bundle if the column
\( k \in \mathcal{K}_j \) places items in fulfillment center \( n \), and 0 otherwise. Thus, the left-hand side
of the constraint is an estimate of the total number of items, across all bundles, that
are placed in the fulfillment center.

When performing disaggregation, we need to ensure that the number of items
allocated to each fulfillment center does not exceed the value dictated by the convex
combination of columns. This can be achieved by adding the constraints

$$
\sum_{i \in \mathcal{I}_j} y_{n}^i \leq t_n^j \quad \forall n = 1, \ldots, N,
$$

to the disaggregation problem \((\text{D}_j)\), where

$$
t_n^j := \left\lfloor \sum_{k \in \mathcal{K}_j} |\mathcal{I}_j| y_{n}^j(k) \lambda_k^j \right\rfloor
$$

is the number of items in bundle \( j \) that can be placed in fulfillment center \( n \), and \( \lfloor \cdot \rfloor \)
denotes rounding down to the nearest integer (although it is possible for some bundles
to round up as well, as long as \( \sum_{j \in \mathcal{J}} t_n^j \leq t_n \)).

In conclusion, the large-scale solution scheme can be extended to problems with
more complex types of constraints, as we have demonstrated with sparsity constraints.
Depending on the resulting structure, different stages of the solution scheme can be
adapted. Finding other applications for this large-scale solution scheme appears to
be a promising future direction.
Chapter 3

Online-retail inventory replenishment

3.1 Introduction

This part of the thesis focuses on the inventory replenishment problem for a single item. We assume that, for the specific item, we are given a set of fulfillment centers that are eligible for placement (possibly produced by the placement problem described in the previous chapter), and that there are no capacity constraints.

For order fulfillment, we focus on the myopic policy, in which the demand is satisfied at the lowest possible shipping cost, using the cheapest— which is typically the closest— fulfillment center that has available inventory and can meet the customer’s service request (e.g., two-day delivery). If the closest fulfillment center is out of stock, demand is rerouted to the next closest fulfillment center, a situation known as spillover. Demand is fulfilled as long as it is feasible to do so for the specified service request, even if it requires shipping from a distant fulfillment center at a high cost. Otherwise, the demand becomes lost sales.

While the myopic fulfillment policy is not optimal, as shown by Acimovic and Graves (2015), it is a reasonable proxy for what is commonly implemented in practice. Furthermore, for low-demand items, the myopic fulfillment policy intuitively appears to be near-optimal, as we can expect each unit sold to be replenished before the next demand arrives, in the average case.

We also assume that the fulfillment centers have deterministic lead times that
are not necessarily identical. Non-identical lead times not only reflect operational realities, but can also potentially be leveraged to reduce the overall cost through the effect of lead time staggering.

To see the potential benefit of lead time staggering, consider the small example illustrated in Figure 3.1, in which there are two fulfillment centers serving the same amount of deterministic demand. The figure compares the total system-wide inventory position over time for the case of identical zero lead times, and the case in which one fulfillment center has a lead time equal to half the length of the review period. In the latter case, the lead times are said to be staggered, and the system as a whole is replenished twice as frequently as in the former case. As a result, the system-wide inventory has less variation, and the total amount of inventory is always positive.

The minimum of the positive inventory can be interpreted to serve as the system-wide safety stock. If the demand is perturbed with a small level of stochasticity, then the staggered lead time case would likely experience less system-wide stockout than the identical lead-time case. In other words, lead time staggering provides the opportunity to achieve a higher service level with a lower amount of inventory. That being said, a longer lead time also corresponds to the need for hedging against greater uncertainty during the lead time. An important motivation of our model is to quantify the cost-benefit trade-off of non-identical lead time staggering.

3.1.1 Related work

Two main features that contribute to the complexity of our problem are that of lost sales and lead time. This combination was first addressed by Karlin and Scarf (1958), who highlight the difficulty of obtaining a closed-form solution even for a single-item, single-location problem. Since then, various assumptions and heuristic policies have been proposed, as reviewed by Zipkin (2008) and Bijvank and Vis (2011). Of particular interest to us are base-stock and constant-order policies. The base-stock policy is widely used in practice due to its simplicity, although Zipkin (2008) provides numerical evidence that this policy may be outperformed by many other heuristics. In the single-location setting, the base-stock policy also exhibits desirable properties
Figure 3.1: Lead time staggering can have a system-wide impact of reducing inventory variation while achieving a higher service level through safety stock.

(a) Identical lead times

(b) Staggered lead times

under some conditions; for example, Huh et al. (2009) shows that the best base-stock policy is asymptotically optimal when the lost sales penalty is large relative to the holding cost, and Downs et al. (2001) and Janakiraman and Roundy (2004) show that the cost is convex with respect to the order-up-to level. The constant-order policy is also simple to implement, and Goldberg et al. (2014) show that the best constant-order policy is asymptotically optimal with respect to growing lead time. These findings for the single-location problem motivate us to consider heuristic policies for our multi-location problem that have the elements of base-stock and constant-order policies.

The setup of our model is closest to that of Acimovic and Graves (2016), who also study online-retail inventory replenishment with lost sales and lead time. They show that the status-quo local base-stock policy is suboptimal and may result in persistent inventory imbalance, a phenomenon they call “whiplash.” They propose a projected base-stock policy and show how it can mitigate inventory imbalance and reduce outbound shipping costs. Our work differs from Acimovic and Graves (2016) in three ways. First, their model focuses on minimizing shipping costs given an exogenous ser-
vice level, while we consider the total operational cost, including shipping, lost sales and holding costs, in order to find the overall optimal inventory level and service level. Secondly, they assume that all fulfillment centers have the same deterministic lead time, while our model allows for non-identical deterministic lead times. Thirdly, they achieve integrality with a rounding procedure, while our model explicitly accounts for the integrality of inventory, which is an important consideration especially for low-demand items that often constitute a large proportion of the online-retail catalog.

We remark that Acimovic (2012) provides a numerical example of two fulfillment centers with different lead times that are both less than the review period, for which the local base-stock policy produces whiplash even when demand is deterministic. However, the author also notes that if some lead times are longer than a review period, the inventory imbalance seems to correct itself after a few periods. The topic of non-identical lead times is largely left for future research (see Acimovic (2012), pages 135-136).

Another stream of related work is the study of lateral transshipment, which also features multiple locations, fungible inventory, and the explicit consideration of fulfillment costs; see Paterson et al. (2011) for a review. In particular, Herer et al. (2006) study the replenishment policy that minimizes the expected average cost over an infinite horizon. Our model differs with theirs in the following ways: First and foremost, their model considers a single realization of batched demand per review period, while in our case demand arrives individually and continuously over the course of the review period, and different demand arrival order sequences result in different fulfillment costs. Secondly, in their multi-location setup, each warehouse serves a single dedicated region, and local stockouts have an isolated effect. In contrast, inventory is fully fungible in our problem, i.e., any fulfillment center may serve any region, and thus, local stockouts affect the entire system. Also, they allow backlogging, whereas we model unmet demand as being lost.
3.1.2 Chapter overview

Section 3.2 begins by formulating the online-retail inventory replenishment problem as an infinite-horizon average cost problem, and derives formulas for computing the state transition probability distribution and the expected stage cost. We then describe the standard methods for solving the dynamic program, and show empirically that these methods suffer from the curse of dimensionality, thereby motivating the necessity of approximations.

Section 3.3 describes two approximation approaches: approximation in policy space, by the use of parameterized policies, and approximation in value space, by the use of simulation. We then propose three simulation optimization methods for finding low-cost parameterized policies.

Section 3.4 provides empirical results for a realistic inventory network to establish the effectiveness of the simulation optimization methods. In particular, the method of random search with gradient-based sampling is shown to be capable of finding low-cost solutions within a few minutes. We then analyze the policies found by random search in various settings to investigate the impact of problem characteristics such as lead times, demand variability, and costs.

Finally, Section 3.5 concludes with a summary of contributions and directions for future work.

3.2 Dynamic programming model

3.2.1 Modeling assumptions

Mathematical notation. Throughout the chapter, we use the italic font $x$ to denote scalars, and the bold font or bracketed scalars $\mathbf{x} = (x_1, \ldots, x_N) = (x_n)_{n=1}^N$ to denote vectors. The subscripts of variables, e.g., $x_n$, denote indices of fulfillment centers, while the superscripts, e.g., $x^k$, denote indices of review periods. The symbols $\mathbb{R}_+^N$ and $\mathbb{Z}_+^N$ represent the $N$-dimensional space of nonnegative real numbers and nonnegative integers, respectively. The $l_1$-norm is denoted by $\|x\|_1$ and is the absolute
sum of all entries of a vector, $\sum_{n=1}^{N} |x_n|$. We use $x \leq y$ as a shorthand for $x_n \leq y_n$ for all $n = 1, \ldots, N$, and $x \geq y$ is defined similarly. We use $x < y$ as a shorthand for the condition that $x \leq y$ and $x_n < y_n$ for some $n$; $x > y$ is defined similarly. The vector $e_n \in \mathbb{Z}_+^N$ denotes the $N$-dimensional vector with 1 in the $n$-th entry and 0 elsewhere, and $0 \in \mathbb{Z}_+^N$ is the vector of all zeros.

**Periodic review and joint replenishment decisions.** The online-retail inventory system can be modeled as a set of $N$ fulfillment centers, indexed by $n = 1, \ldots, N$. We consider a single item, for which there is no capacity constraint, and whose inventory level takes on non-negative integer quantities.

We assume that inventory at every fulfillment center is reviewed periodically at a given, common review epoch. At the review epoch, the replenishment order is placed concurrently for all the fulfillment centers, and the order quantity decision is made with all the inventory positions in mind. This is referred to as a joint replenishment decision.

**Deterministic, non-identical replenishment lead times.** Let $L \in \mathbb{R}_+$ be the length of the review period, and let $L_n \in \mathbb{R}_+$ be the replenishment lead time for fulfillment center $n$. Both $L$ and $L_n$ are assumed to be given as inputs. The lead times do not have to be identical, and without loss of generality, we assume that $L_1 \leq L_2 \leq \cdots \leq L_N$. For simplicity, we formulate our model for the case where $L_N < L$, so that there is no outstanding on-order (in-transit) inventory at any fulfillment center at the review epoch. Our model and methods can be generalized to the case where some lead times exceed the review period, by augmenting the state space to include on-order inventory. For cohesion in notation, we also adopt the convention of $L_0 = 0$ and $L_{N+1} = L$ in some expressions.

**Poisson customer demand.** We assume that customers can be grouped according to a finite set of geographic regions where they are located, indexed by $r = 1, \ldots, R$. Besides the region, customers are otherwise indistinguishable in the sense that they pay the same retail price, require the same type of service (e.g., two-day
shipping), and experience the same dissatisfaction when the item is not available.

We also assume that customer demand arrives continuously throughout the course of a review period, and that demand arrival from each region follows an independent Poisson process (although our model extends to any independent and memoryless process). Let \( \lambda_r \) be the proportion of demand arrivals that originate from region \( r \), with \( \sum_{r=1}^{R} \lambda_r = 1 \). Let \( D \) be the system-wide demand arrival rate per review period. Thus, \( \lambda_r D \) is the average demand per review period in region \( r \).

Denote \( f_d \) as the probability mass function of a Poisson process with rate \( d \), i.e., for \( m \in \mathbb{Z}_+ \), \( f_d(m) = \frac{d^m e^{-d}}{m!} \) is the probability that there are exactly \( m \) demand arrivals during a time interval where the average demand is \( d \). Let \( F_d \) denote the corresponding cumulative density function, i.e., \( F_d(m) \) is the probability that there are no more than \( m \) demand arrivals during this period.

**Myopic fulfillment policy.** We assume that all fulfillment centers are able to meet customer service requests at every region. For example, for a two-day shipping service, we assume that any fulfillment center can ship the item to any region within two days (although some pairs of fulfillment center and customer region may require a transportation mode that incurs an extremely high cost, such as air shipment.) As a result, inventory is fully fungible, and demand will be satisfied as long as there is inventory at some fulfillment center. We assume that demand arrivals are satisfied by the cheapest fulfillment center with on-hand inventory, which we refer to as the myopic fulfillment policy. If there is no inventory in the system, demand is lost.

**Holding, stockout and shipping costs.** Let \( c_u \geq 0 \) denote the stockout (or underage) cost of each unit of unmet demand. This typically equals the unit sales profit plus a lost sales penalty, which is a monetary value reflecting the loss of customer goodwill. Let \( c_o \geq 0 \) be the holding (or overage) cost, charged to every unit of on-hand inventory when a replenishment order arrives. In other words, \( c_o \) reflects the opportunity cost for holding one unit of inventory for the length of the review period. These costs are assumed to be given and identical for all fulfillment centers, although
it is straightforward to extend our model to the case of non-identical stockout and holding costs. There is no fixed setup costs for either ordering or holding inventory.

Let $c_{nr} \geq 0, n = 1, \ldots, N, r = 1, \ldots, R$, be a given table of per-unit shipping costs from fulfillment center $n$ to customer in region $r$. We assume that for every $r$, $\arg\min_{n \in \mathcal{N}} c_{nr}$ is unique for every nonempty subset among fulfillment centers $\mathcal{N} \subset \{1, \ldots, N\}$, i.e., there is a tie-breaking rule for fulfillment centers that have the same shipping cost to region $r$, so that the myopic fulfillment policy always makes a consistent choice. We also assume that $c_{nr} \leq c_n$ for all $n$ and $r$, which is to say that from the perspective of a single demand, there is no incentive to withhold any available on-hand inventory from being shipped and sold.

**Sequence of events.** In each review period, the following cost-incurring events take place. First, at the review epoch, inventory is reviewed jointly at all fulfillment centers, and replenishment orders are placed and billed.

As time goes on, demand arrives continuously throughout the review period, and is either fulfilled instantly according to the myopic policy, which incurs some shipping cost, or lost if there is a system-wide stock-out, which incurs a stockout cost.

Also, at every lead time $L_n, n = 1, \ldots, N$, just before the replenishment order for fulfillment center $n$ arrives, the pre-replenishment on-hand inventory at fulfillment center $n$ is charged a holding cost. The order is then received and added to the on-hand inventory.

We remark that our method of accounting for holding cost is different than the convention for most single-location inventory models, where the holding cost is charged at the review epoch. For the multi-location setting, charging the holding cost at the time of replenishment accurately captures the amount of safety stock, i.e., the excess on-hand inventory held to prevent stockouts. In contrast, if holding cost were charged according to the inventory at the review epoch, then fulfillment centers that were replenished more recently would inaccurately receive greater penalty for its inventory. As an example, consider a system of $N = 2$ fulfillment centers with lead times $L_1 = 1, L_2 = 6$ and a review period of $L = 7$ days. At any given review epoch,
the second fulfillment center has just received its replenishment orders one day ago, while the first fulfillment center has had six days to deplete its replenishment order. Thus it would be unfair to charge the same holding cost to the on-hand inventory of both fulfillment centers at the review epoch.

We also note that another valid approach for measuring inventory is to charge a holding cost on the average of pre- and post-replenishment inventory at the replenishment lead time. The two metrics simply differ by the expected number of units sold per review period, which can easily be computed.\(^1\)

### 3.2.2 Dynamic program formulation

**Infinite-horizon average-cost model**

The online-retail inventory replenishment problem can be formulated as an infinite-horizon average cost dynamic program, in which each stage represents a review period and is indexed by \(k\). The state \(x^k \in X = \mathbb{Z}_+^N\) is the vector of inventory positions at the \(N\) fulfillment centers at the review epoch prior to the replenishment decision. The control \(u^k \in U(x^k) = \mathbb{Z}_+^N\) is the vector of joint replenishment order quantities, calculated based on \(x^k\).

A replenishment policy \(\mu : x \mapsto u \in U(x)\) is a function that maps the state variable to the control variable. Since we are interested in the steady-state behavior of an infinite-horizon problem, we focus our attention on *stationary* policies, i.e., policies that are the same for all review periods. Our goal as the online-retail decision maker is to find a stationary policy that minimizes the infinite-horizon average cost of the inventory replenishment problem.

We characterize the infinite-horizon average cost in three steps. First, we define the expected cost and transition probability distribution in a single stage (i.e., a review period). Next, we consider a finite-horizon dynamic program with multiple stages. Finally, we extend the expressions to an infinite horizon.

Given a state \(x\) and a control \(u\), define \(g(x, u)\) as the expected stage cost over a

---

\(^1\)For a given policy, let \(q\) be the expected stockout cost per period. Then \(\frac{q}{c_w}\) is the expected number of lost sales per period, and \(D - \frac{q}{c_w}\) is the expected number of units sold per review period.
single review period, and define $\mathbb{P}(x, u, y)$ as the probability of transitioning to state $y \in X$ at the end of the review period. Note that $\mathbb{P}(x, u, y)$ is nonzero if and only if $0 \leq y \leq x + u$, because $x + u$ is the maximum possible inventory position at the end of the review period under Poisson demand.

Now consider a dynamic program with $K$ stages, indexed by $k = 0, 1, \ldots, K - 1$. Let $J_{\mu}^{k,K}$ denote the finite-stage total cost from stage $k$ up to and including stage $K - 1$, for a specified policy $\mu$. Define the terminal cost as $J_{\mu}^{K,K}(x) = 0$ for all $x \in X$. Then, the finite-stage total cost for a given policy $\mu$ can be expressed recursively as

$$J_{\mu}^{k,K}(x^k) = g(x^k, u^k) + \sum_{x^{k+1} \in X} \mathbb{P}(x^k, u^k, x^{k+1}) J_{\mu}^{k+1,K}(x^{k+1} \mid u^k = \mu(x^k)), \quad k = 0, \ldots, K - 1,$$

where the first term is the expected cost of stage $k$ and the second term is the expected cost-to-go for future stages. As noted previously, the summation is in fact taken over all $x^{k+1}$ such that $0 \leq x^{k+1} \leq x^k + u^k$. The value of $J_{\mu}^{0,K}(x^0)$ is the total cost for the $K$-stage problem, for a given policy $\mu$ and initial state $x^0$.

The infinite-horizon average cost for policy $\mu$ can then be defined by taking $K$ to infinity:

$$g_{\mu} := \limsup_{K \to \infty} \frac{1}{K} J_{\mu}^{0,K}(x^0). \quad (3.1)$$

As we show in Section 3.2.4, $g_{\mu}$ does not depend on the initial state $x^0$, and is only a function of the policy $\mu$.

Our ultimate goal is to solve the optimization problem

$$\min_{\mu} g_{\mu}. \quad (3.2)$$

The computation of $g_{\mu}$ relies on the characterization of the expected stage cost $g(x, u)$ and the state transition probability distribution $\mathbb{P}(x, u, y)$. This is the focus of the remainder of the section.
Segmentation by lead time

For a given review period, let \( \mathbf{x} \) be the vector of on-hand inventory levels at the review epoch, and let \( \mathbf{u} \) be the replenishment order quantity placed at the review period. In addition, let \( \mathbf{z}_n \) be the random vector of pre-replenishment on-hand inventory levels at the lead time \( L_n, n = 1, \ldots, N \), and let \( \mathbf{z}'_n = \mathbf{z}_n + u_n \mathbf{e}_n \) be the vector of post-replenishment on-hand inventory levels.

Due to the memoryless property of the demand arrival process, \( \mathbf{z}_n \) depends only on \( \mathbf{z}'_{n-1} \), and not on the inventory levels before lead time \( L_{n-1} \). Therefore, we can decompose the review period into independent segments, using the lead times as breakpoints. We shall henceforth refer to the time interval of \( [L_n, L_{n+1}) \) as segment \( n \), where \( n = 0, 1, \ldots, N \). (Recall that we adopt the convention of \( L_0 = 0 \) and \( L_{N+1} = L \).) This segmentation technique is illustrated in Figure 3.2.

For every segment \( n = 1, \ldots, N \), let \( g_n(\mathbf{z}_n, u_n) \) be the expected cost during segment \( n \), which is a function of the initial inventory \( \mathbf{z}_n \) and the replenishment quantity \( u_n \) received at fulfillment center \( n \). Let \( \mathbb{P}_n(\mathbf{z}'_n, \mathbf{z}_{n+1}) \) be the probability of ending up with on-hand inventory level \( \mathbf{z}_{n+1} \) after serving demand in segment \( n \), given the post-replenishment inventory level \( \mathbf{z}'_n \) (which we recall is equal to \( \mathbf{z}_n + u_n \mathbf{e}_n \)). We defer the detailed characterization of \( \mathbb{P}_n \) and \( g_n \) to the two subsections below. For now, assuming these values are available, we focus on how to combine them together to determine the overall \( \mathbb{P} \) and \( g \).

To derive \( \mathbb{P} \) from \( \mathbb{P}_n \), some auxiliary expressions are needed. Let \( \mathbb{P}_n, n = 1, \ldots, N \), be the composite transition probability distribution of the interval \( [L_n, L) \), i.e., \( \mathbb{P}_n(\mathbf{z}'_n, \mathbf{y}) \) is the probability of ending up with inventory level \( \mathbf{y} \) at the end of the review period, given that the post-replenishment inventory level at lead time \( L_n \) is \( \mathbf{z}'_n \). This composite transition probability can be expressed recursively with the segment transition probabilities \( \mathbb{P}_n \) as follows:
\[
\bar{P}_N(z'_N, y) = P_N(z'_N, y), \\
\bar{P}_n(z'_n, y) = \sum_{z_{n+1}: 0 \leq z_{n+1} \leq z'_n} P_n(z'_n, z_{n+1})\bar{P}_{n+1}(z'_{n+1}, y), \quad n = N - 1, N - 2, \ldots, 1.
\]

The first line is the trivial case of the last segment, and the second line expresses \( \bar{P}_n \) in terms of \( \bar{P}_{n+1} \) by summing over all possible intermediate states \( z_{n+1} \) (recall that \( z'_{n+1} = z_{n+1} + u_{n+1}e_{n+1} \)). We then have

\[
P(x, u, y) = \sum_{z_1: 0 \leq z_1 \leq x} P_0(x, z_1)\bar{P}_1(z'_1, y),
\]

where \( P_0(x, z_1) \) is the probability of transitioning from \( x \) to \( z_1 \) in segment 0, and \( z'_1 = z_1 + u_1e_1 \).

To perform this calculation given \( x, u \) and \( y \leq x + u \), we begin by computing \( \bar{P}_N(z'_N, y) \) for every \( z'_N \in X, z'_N \leq y \). We then compute \( P_n(z'_n, z_{n+1}) \) in sequential order from \( n = N - 1 \) down to 1, using \( u \) to determine the appropriate set of \( z'_n \) to compute. Finally, \( P(x, u, y) \) is obtained from the last expression.

Similarly, let \( \bar{g}_n, n = 1, \ldots, N \), be the expected cost for the interval \([L_n, L]\). These quantities can be calculated recursively using \( g_n \):

\[
\bar{g}_N(z_N, u) = g_N(z_N, u_N), \\
\bar{g}_n(z_n, u) = g_n(z_n, u_n) + \sum_{z_{n+1}: 0 \leq z_{n+1} \leq z'_n} P_n(z'_n, z_{n+1})\bar{g}_{n+1}(z_{n+1}, u), \quad n = 1, \ldots, N - 1,
\]

Again, the first line is the trivial case of the last segment, and the second line expresses \( \bar{g}_n \) as the segment cost \( g_n \) plus the expected cost-to-go from segment \( n + 1 \) to the end of the period, where \( z'_n = z_n + u_ne_n \).

Then, the expected stage cost \( g(x, u) \) can be obtained by

\[
g(x, u) = g_0(x) + \sum_{z_1: 0 \leq z_1 \leq x} P_0(x, z_1)\bar{g}_1(z_1, u),
\]
where \( g_0(x) \) is the expected cost in segment 0.

Figure 3.2: Online-retail inventory replenishment can be modeled as a dynamic program (DP), in which the stage transition probability distribution is obtained by segmenting the review period using lead times as breakpoints. The derivation is similar for the expected stage cost.

Characterization of segment transition probabilities

In this section, we derive the expression for the segment transition probability distribution \( P_n \) through a three-step process. We begin by quantifying the effect of a single demand arrival on the on-hand inventory level. Next, we compute the conditional transition probability for a fixed number of demand arrivals. Finally, we combine the conditional probabilities into the segment transition probability using the demand distribution.

For simplicity, we deviate from the previous notation and let \( x, y \in X \) be variables that denote inventory positions in general (rather than being associated with a certain time). The segment \( n \) transition probability \( P_n(x, y) \) is the probability of ending up at inventory position \( y \) at the end of segment \( n \), given post-replenishment inventory position \( x \) at the beginning of the segment.
Effect of a single demand arrival. Given state $x \in X$, $x \neq 0$, let

$$A(x) = \{n = 1, 2, \ldots, N \mid x_n > 0\}$$

be the set of fulfillment centers with available on-hand inventory. For each $n \in A(x)$, define the set of regions served by $n$ under state $x$ as

$$R_n(x) = \left\{ r \in \{1, 2, \ldots, R\} \mid c_{nr} = \min_{n' \in A(x)} c_{n'r} \right\}, \quad n \in A(x).$$

Let $p(x, x - e_n)$ be the probability that the inventory level vector transitions from $x$ to $x - e_n$ as a result of the next demand arrival. (Recall that $e_n \in \mathbb{Z}_+^N$ is the vector with 1 in the $n$-th entry and 0 elsewhere.) This is equal to the probability that the next demand comes from a region in $R_n(x)$. Since the demand arrival process in each region is an independent Poisson process, we have

$$p(x, x - e_n) = \sum_{r \in R_n(x)} \lambda_r, \quad n \in A(x), x \neq 0, \quad (3.3)$$

where we recall that $\lambda_r$ is fraction of demand arrivals that originate from region $r$.

Note that $R_n(x)$ may be an empty set for some $n \in A(x)$, which implies that $n$ is not the cheapest fulfillment center for any customer region under state $x$. In this case, $p(x, x - e_n) = 0$, which indicates that it is impossible to transition from state $x$ to state $x - e_n$. Note also that if $x = e_n$, then $A(x) = \{n\}$ is a singleton, which leads to $R_n(x) = \{1, \ldots, R\}$ and $p(x, x - e_n) = p(e_n, 0) = 1$.

Conditional transition probability. Let $\hat{P}(x, y)$ denote the conditional probability of transitioning from initial inventory $x$ to inventory level $y \in X, y \leq x$, conditioned on there being exactly $\|x - y\|_1$ demand arrivals. Then $\hat{P}$ can be written
The first line of this expression is the base case for recursion. It states trivially that conditioned on having no demand arrivals, the probability of remaining at \( x \) is 1. In the second line, the conditional probability of transitioning to \( y \) is expressed as the sum over fulfillment centers that may serve the first demand arrival, \( n \in A(x - y) \). Any demand sequence that results in a transition from \( x \) to \( y \) must pass through some inventory level \( x - e_n \), where \( n \) is such that \( y_n < x_n \) (or equivalently, \( n \in A(x - y) \)), as a result of the first demand arrival.

An example illustrating the use of (3.4) is given below.

**Example.** Consider a system with \( N = 2 \) fulfillment centers. Let \( x = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \) and \( y = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \). We show how to apply (3.4) to compute \( \hat{P}(x, y) \), the conditional probability for transitioning from \( x \) to \( y \) given that there are exactly two demand arrivals.

After the first demand is served, the system ends up at one of two possible states: \( z_1 = \begin{bmatrix} 3 \\ 0 \end{bmatrix} \) or \( z_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \). According to (3.4), the conditional probabilities of transitioning from these two respective states to \( y \) are

\[
\hat{P}(z_1, y) = p(z_1, y)\hat{P}(y, y) = p(z_1, y),
\]
\[
\hat{P}(z_2, y) = p(z_2, y)\hat{P}(y, y) = p(z_2, y).
\]

The probability of transitioning from \( x \) to \( z_1 \) and to \( z_2 \) are given by \( p(x, z_1) \) and \( p(x, z_2) \), respectively.

The conditional probability \( \hat{P}(x, y) \) can be found by adding the probabilities
of demand sample paths through \( z_1 \) and \( z_2 \), i.e.,

\[
\hat{P}(x, y) = p(x, z_1)\hat{P}(z_1, y) + p(x, z_2)\hat{P}(z_2, y)
\]

\[
= p\left(\left[\begin{array}{c} z_1 \\ \end{array}\right], \left[\begin{array}{c} 0 \\ \end{array}\right]\right) + p\left(\left[\begin{array}{c} z_2 \\ \end{array}\right], \left[\begin{array}{c} 1 \\ \end{array}\right]\right).
\]

All the values of \( p(\cdot, \cdot) \) can be obtained from (3.3).

**Segment transition probability.** Due to the fact that demand is memoryless, the system-wide demand arrival during the \( n \)-th segment is a Poisson process with rate

\[
D_n = D \times \frac{L_{n+1} - L_n}{L}.
\]

Let \( x \) be the post-replenishment on-hand inventory at breakpoint \( L_n \), and let \( y \) be the pre-replenishment on-hand inventory at breakpoint \( L_{n+1} \). The probability of transitioning from \( x \) to \( y \), as a result of serving demand arrivals in the \( n \)-th segment, is given by multiplying the conditional probability \( \hat{P}(x, y) \) with the probability that there are exactly \( \|x - y\|_1 \) demand arrivals,

\[
\mathbb{P}_n(x, y) = \begin{cases} 
  f_{D_n}(0), & x = y, \\
  f_{D_n}(\|x - y\|_1) \cdot \hat{P}(x, y), & y < x \neq 0,
\end{cases}
\]  

(3.5)

**Characterization of expected segment cost**

The cost of segment \( n \), denoted \( g_n(x, u) \), is the sum of the shipping, stockout and holding costs incurred during the time of \( [L_n, L_{n+1}) \). We explain each cost component below. Throughout this subsection, for a given \( n \), let \( x \) denote the pre-replenishment on-hand inventory at breakpoint \( L_n \), and let \( u \) denote the order quantity placed in the beginning of the review period. Thus, the post-replenishment inventory position at the start of segment \( n \) is \( x + u_ne_n \).

**Holding cost.** Immediately before replenishment at breakpoint \( L_n \), a holding cost \( c_o \) is charged to every unit of on-hand inventory at fulfillment center \( n \). The holding
cost of the \( n \)-th segment is thus

\[
\text{Segment } n \text{ holding cost } = c_ox_n. \tag{3.6}
\]

**Stockout cost.** Let \( q = \|x\|_1 + u_n \) be the system-wide post-replenishment inventory position, immediately after fulfillment center \( n \) is replenished. Lost sales will occur during segment \( n \) if and only if the number of demand arrivals exceeds \( q \). Each unit of unmet demand is charged an underage cost of \( c_u \). The expected stockout cost is thus

\[
\text{Segment } n \text{ expected stockout cost } = c_u \cdot \sum_{m=q+1}^{\infty} (m - q) f_{D_n}(m) \tag{3.7}
\]

\[
= c_u \left( D_n - \sum_{m=1}^{q} m f(m) - q[1 - F_{D_n}(q)] \right) \tag{3.8}
\]

where the second line is more suitable for actual computation, and can be easily verified using the fact that \( D_n \) is the expected total demand during the segment.

Note that the segment stockout cost only depends on the system-wide inventory in the system, and is indifferent to the allocation of inventory across individual fulfillment centers. Indeed, because inventory is fungible, a local stockout at a fulfillment center can be compensated by inventory in other fulfillment centers. Therefore, the system-wide stockout probability only depends on the total system-wide inventory.

**Shipping cost.** Define \( s_m(z) \) as the expected shipping cost of the \( m \)-th demand arrival fulfilled by inventory position \( z \), conditioned on there being at least \( m \) demand arrivals, \( 0 \leq m \leq \|z\|_1 \). We derive a recursive expression for calculating \( s_m(z) \), which we then use for computing the expected shipping cost of the segment.

Let \( z = x + u_n e_n \) be the initial post-replenishment inventory in segment \( n \). For the very first demand arrival in the segment, assuming that there is one, the expected
shipping cost is

\[
s_1(z) = \sum_{r=1}^{R} \lambda_r \left( \min_{n \in A(z)} c_{nr} \right),
\]

which is the weighted average of the cheapest shipping costs, given that the current set of fulfillment center with available inventory is \(A(z)\).

Now suppose there are at least two demand arrivals. When the second demand arrives, the inventory position will be \(z - e_n\) for some \(n \in A(z)\). We can think of this demand arrival as the first demand arrival under initial inventory \(z - e_n\), thereby expressing its expected shipping cost as

\[
s_2(z) = \sum_{n \in A(z)} p(z, z - e_n) s_1(z - e_n).
\]

Similarly, for the \(m\)-th demand arrival, \(m \geq 2\), conditioned on its existence, the expected shipping cost can be computed recursively:

\[
s_m(z) = \sum_{n \in A(z)} p(z, z - e_n) s_{m-1}(z - e_n), \quad m = 2, 3, \ldots, \|z\|_1.
\]

Then, given any post-replenishment initial inventory \(z\) in segment \(n\), we can compute the expected segment shipping cost by multiplying each \(s_m(z)\) with the probability of there being at least \(m\) demand arrivals during this segment,

\[
\text{Segment } n \text{ expected shipping cost} = \sum_{m=1}^{\|z\|_1} s_m(z) [1 - F_{D_n}(m - 1)].
\]

Note that in the actual implementation, since \(z\) can take on any value in \(X\), it may be more computationally efficient to construct the table of values of \(s_m(z)\), \(z \in X, m = 1, \ldots, \|z\|_1\) off-line and store it in memory, so that the values can be used repeatedly. To construct the table, we begin by calculating \(s_1(z)\) for every \(z\) using (3.9). Then, in the order of increasing \(\|z\|_1\), compute \(s_m(z)\) for every \(z\) and every \(m = 2, \ldots, \|z\|_1\). The order of computation ensures that all values on the right-hand
side of (3.10) will be available when needed.

3.2.3 Bounds

This section describes methods to estimate bounds on the state space and on the optimal cost. For a numerical example that illustrates the calculation of these bounds, see Section 3.2.5.

Existence of a bounded state space

So far, we have kept a general definition of the state space $X$ and control space $U$. We now show that in fact, it is sufficient to consider a finite state and control space.

Indeed, since the expected demand is finite, as we increase the on-hand inventory level at each fulfillment center, the expected shipping and stockout costs per stage are bounded, while the expected holding cost grows unbounded. Therefore, there exists finite inventory levels $M_n \in \mathbb{Z}_+, n = 1, \ldots, N$, such that the state space

$$X = \{ x \in \mathbb{Z}_+^N \mid x_n \leq M_n \ \forall n = 1, \ldots, N \}$$  \hspace{1cm} (3.12)

contains the recurrent states of the optimal policy, i.e., the control space

$$U(x) = \{ u \in \mathbb{Z}_+^N \mid x + u \in X \}, \quad \forall x \in X$$

is such that the set of policies $\{ \mu : x \mapsto u \in U(x) \}$ contains the optimal policy.

Having a bounded state space is useful because it limits the set of policies we need to optimize over. The natural question that follows is, how do we obtain the inventory bounds $\{M_n\}_{n=1}^N$? Below, we show that these bounds can be estimated by means of a conservative base-stock policy.
Upper bounds on the cost and inventory position

Consider the base-stock policy

$$\bar{\mu}_n(x) = (M_n - x_n)^+, \quad n = 1, \ldots, N,$$  \hspace{1cm} (3.13)

where $(\cdot)^+$ indicates projection unto the set of non-negative numbers. In other words, this is a base-stock policy that always brings the inventory position (which includes the on-hand and on-order inventory) back up to the inventory bounds $\{M_n\}_{n=1}^N$. It is a feasible policy, and therefore its cost is an upper bound on the optimal cost (3.2).

The policy (3.13) provides a way to estimate inventory position bounds $\{M_n\}_{n=1}^N$. Specifically, if we choose $\{M_n\}_{n=1}^N$ so that (3.13) is a conservative ordering policy that over-orders and overstocks, then the $\{M_n\}_{n=1}^N$ values would also serve as an upper bound on the inventory position for recurring states of the optimal policy.

We can set (3.13) to a conservative policy by making the following two assumptions:

1. Spillovers are disallowed. In other words, we assume that a demand is lost if it cannot be satisfied by the fulfillment center closest to its region of origin. As a result, the cost of a local stockout is higher than it actually is when spillover is permitted, and therefore, the fulfillment centers will hold more inventory than is necessary with spillovers.

2. In addition to disallowing spillovers, we also assume when calculating the base stock level that any shortages result in backorders, rather than in lost sales. Under this assumption, the expected on-hand inventory level is the base-stock level minus the expected demand during lead time. The resulting base stock level will hence be larger than that for the case with lost sales. (In the case of lost sales, the expected on-hand inventory is the base-stock level minus the actual demand served during the lead time, which is equal to the expected demand during lead time minus lost sales.)

The first assumption allows each $M_n$ to be optimized independently, since the
fulfillment centers are dedicated to non-overlapping local demand. The second assumption allows us to use the single-period Newsvendor model to optimize $M_n$.

The Newsvendor model states that for a given time period, let $f$ be the probability distribution function, and $F$ the cumulative distribution function, of the total demand during this period. Also, let $c_1$ be average cost per unit inventory leftover at the end of the time period, and let $c_2$ be underage cost per unit of unmet demand during the time period. Then the expected total cost, as a function of the initial inventory $q$, is

$$c_1 \sum_{d=0}^{q} (q-d)f(d) + c_2 \sum_{d=q}^{\infty} (d-q)f(d).$$

It can be shown that the cost is minimized at the smallest integer $q$ such that $F(q) \geq \frac{c_2}{c_1 + c_2}$. The value of $\frac{c_2}{c_1 + c_2}$ is commonly referred to as the critical fractile. It reflects the target local service level that the optimal inventory $q$ should achieve, i.e., the probability that there are no lost sales during the period.

To apply the Newsvendor model for a given fulfillment center $n$, we need to determine its demand distribution and critical fractile, assuming no spillovers. Let $R_n$ be the set of regions whose closest fulfillment center is $n$, i.e., $R_n = \{ r \in R \mid n = \text{argmin}_{r'} c_{n'r} \}$. Then the local demand of fulfillment center $n$ follows a Poisson distribution with an average demand per period of

$$d_n = D \cdot \sum_{r \in R_n} \lambda_r.$$

The critical fractile at fulfillment center $n$ is given by $\frac{c_2}{c_1 + c_2} = \frac{c_u - c_n}{c_o + c_u - c_n}$, where $c_1 = c_o$ is the average cost (i.e., the holding, cost), and $c_2 = c_u - c_n$ is the cost difference between a local stockout and a shipped demand, with

$$c_n = \frac{\sum_{r \in R_n} \lambda_n c_{nr}}{\sum_{r \in R_n} \lambda_n}$$

being the expected shipping cost for each demand fulfilled by $n$.

\footnote{Recall that we assume the existence of a tie-breaking rule that ensures the uniqueness of the minimizer $\text{argmin}_{n'} c_{n'r}$, so that $\{R_n\}_{n=1}^N$ form a partition of $R$.}
Finally, note that for the base-stock policy (3.13), the time period to which we apply the Newsvendor model is in fact the lead time plus the review period (and not just the review period). In other words, the base-stock level $M_n$ reflects the amount of inventory that minimize the cost over the period of $L + L_n$, for which the cumulative demand distribution is $F_{(1 + L_n)}$. Therefore, each $M_n$ should be chosen as the smallest amount of inventory that achieves the target local service level indicated by the critical fractile, i.e.,

$$M_n = \min \left\{ q \in \mathbb{Z}_+ \mid F_{(1 + L_n)}(q) \geq \frac{c_u - c_n}{c_o + c_u - c_n} \right\}, \quad n = 1, \ldots, N. \quad (3.14)$$

We shall henceforth refer to (3.13) as the Newsvendor base-stock policy, and use it as a basis for comparison with more sophisticated optimization methods we develop in the next section. From informal conversations with practitioners in the online-retail industry, this policy appears to be a reasonable proxy for policies implemented in practice. Its main appeal is the computational simplicity, but the drawback is that it carries more inventory than necessary.

**Lower bound on the cost**

We now show that a lower bound on the optimal cost (3.2) is in fact the minimum expected shipping cost, given by $\bar{s} \cdot D$, where $D$ is the expected demand and

$$\bar{s} = \sum_{n=1}^{N} \sum_{r \in R_n} \lambda_n c_{nr}, \quad (3.15)$$

is the expected per-unit shipping cost if demand in every region is always served by its closest fulfillment center.

The argument is simple. For any given policy, every unit of demand either incurs a shipping cost, for which $\bar{s}$ is a lower bound by definition, or a stockout cost, for which $\bar{s}$ is also a lower bound by the assumption that $c_u \geq c_{nr}$ for every $n, r$. Therefore, the expected per-period shipping and stockout cost of any policy is bounded below by $\bar{s} \cdot D$. This is a lower bound on the overall cost per period, as it does not consider
the holding cost, which is nonnegative.

The bound can be quite loose, as it does not utilize either the unit holding cost \( c_o \) or the unit stockout cost \( c_u \). Nevertheless, it provides a simple way of bounding the optimality gap of a policy, and will come in handy for large-scale problems for which the optimal policy is difficult to compute.

### 3.2.4 Exact solution methods

#### Optimality conditions

Under our assumption of Poisson demand, there is a nonzero probability of being in the system-wide stockout state \( x = 0 \) at any period under any policy. Therefore, the infinite-horizon dynamic program may be thought of as a series of finite-horizon problems segmented by visits to state 0. Based on this observation, Bertsekas (2005) shows that the infinite-horizon average-cost problem can be transformed into a series of stochastic shortest path problems. By Bertsekas (2012), Proposition 5.1.3, there exists an optimal stationary policy for the problem. By Bertsekas (2005), Proposition 7.4.1, the optimal policy (3.2) has the following properties:

- The optimal average cost is the same for all initial states.

- The optimal average cost, denoted \( g^* \), together with some function \( h^*: X \to \mathbb{R} \), satisfies Bellman’s equation

\[
g^* + h^*(x) = \min_{u \in \mathcal{U}(x)} \left\{ g(x, u) + \sum_{y \leq x+u} \mathbb{P}(x, u, y)h^*(y) \right\}, \tag{3.16}
\]

for all \( x \in X \).

- If \( \mu(x) \) attains the minimum in (3.16) for all \( x \in X \), the stationary policy \( \mu \) is optimal. In addition, out of all functions \( h^* \) satisfying this equation, there is a unique function for which \( h^*(0) = 0 \).

- If a scalar \( g \) and a function \( h: X \to \mathbb{R} \) satisfy (3.16), then \( g \) is the optimal average cost per stage for all initial states.
Given a stationary policy $\mu$ with corresponding average cost per stage $g_\mu$, there is a unique function $h_\mu : X \to \mathbb{R}$ such that $h_\mu(0) = 0$ and

$$g_\mu + h_\mu(x) = g(x, \mu(x)) + \sum_{y \leq x + \mu(x)} \mathbb{P}(x, \mu(x), y)h_\mu(y)$$  \hspace{1cm} (3.17)

for all $x \in X, x \neq 0$.

These properties imply that the infinite-horizon average cost of a stationary policy can be evaluated by solving (3.17), and that the optimal stationary policy can be found by solving Bellman’s equation (3.16). These implications provide the foundation for the solution methods described in the following subsection.

**Solution methods**

We now present three solution methods that produce the exact optimal solution of our problem. The first two are the standard formulations of relative value iteration (RVI) and policy iteration (PI), which are well-known methods for solving the infinite-horizon average-cost dynamic program, and are described in detail in Bertsekas (2005). The third is a hybrid formulation of RVI and PI that we developed and found to be more efficient for our problem in practice.

Recall that formulas for the expected stage cost $g(x, y)$ and the state transition probability distribution $\mathbb{P}(x, u, y)$ are given in Section 3.2.2, and the values can be either pre-computed and stored as a table, or computed on-the-fly. In our implementations, we do the latter to prevent running out of memory.

**Relative value iteration (RVI).** Let $h_0^0(x) = 0$, for all $x \in X$, be the initial function. For every iteration $t = 0, 1, \ldots$, compute

$$g^{t+1} = \min_{u \in U(0)} \left\{ g(0, u) + \sum_{y \leq u} \mathbb{P}(0, u, y)h^t(y) \right\},$$  \hspace{1cm} (3.18)

$$h^{t+1}(x) = \min_{u \in U(x)} \left\{ g(x, u) + \sum_{y \geq x+u} \mathbb{P}(x, u, y)h^t(y) \right\} - g^{t+1}.$$  \hspace{1cm} (3.19)
The method is called relative value iteration due to the fact that the value of $h^t$ is maintained relative to a special state (the state 0 in our case), which keeps $h^t$ bounded. If the process converges, then $g^t$ approaches $g^\ast$. Note that the process is not guaranteed to converge (see discussion on page 432 of Bertsekas (2005)), although in our numerical experiments we have not encountered a non-convergent instance. In our implementation, we terminate the process when the maximum value change between iterations is below $10^{-10}$, i.e., if

$$\delta^t := \max \left\{ |g^{t+1} - g^t|, \max_{x \in X} |h^{t+1}(x) - h^t(x)| \right\} \leq 10^{-10}. \quad (3.20)$$

**Policy iteration (PI).** Let $\mu^0(x) = 0$ for all $x \in X$ be the initial stationary policy. For every iteration $t = 0, 1, \ldots$, perform the following two steps:

1. **Policy evaluation:** Find $g^t$ and $h^t$ such that for all $x \in X$,

$$g^t + h^t(x) = g(x, \mu^t(x)) + \sum_{y \leq x + \mu^t(x)} \mathbb{P}(x, \mu^t(x), y) h^t(y). \quad (3.21)$$

(3.21) is a system of linear equations, but the large number of equations prevents us from solving it directly. Instead, as shown in Bertsekas (2005), the solution can be obtained by repeatedly applying (3.18) and (3.19), with the minimization operator replaced by $\mu^t(x)$, until the maximum change in values between iterations is below some tolerance level ($10^{-10}$ in our case).

2. **Policy improvement:** Compute $\mu^{t+1}$, which is such that for all $x \in X$,

$$\mu^{t+1}(x) \in \arg\min_{u \in U(x)} \left\{ g(x, u) + \sum_{y \leq x + u} \mathbb{P}(x, u, y) h^t(y) \right\}. \quad (3.22)$$

We do this by checking every possible control in $U(x)$. If $\mu^{t+1} = \mu^t$, terminate; otherwise, go back to the policy evaluation step. It can be shown that policy iteration terminates with the optimal policy after a finite number of iterations (Bertsekas (2005), Proposition 7.4.2).
Hybrid RVI-PI. Let \( h^0(x) = 0 \) be the initial function. For iterations \( t = 0, 1, \ldots \), perform the following steps:

1. **Value iteration stage**: Given \( h^t \), perform (3.18), (3.19). In each iteration, set \( \mu_t \) to be the minimizer of (3.18) and (3.19), and terminate when \( \mu^t = \mu^{t+1} \). In other words, perform RVI until it converges on a policy (rather than until the function values converge). Go to step 2.

2. **Policy evaluation stage**: Given \( \mu^t \), solve (3.21) to obtain \( g^t \) and \( h^t \). At this point, if \( \mu^t \) attains the minimum of Bellman’s equation (3.16), it is the optimal solution; terminate. Otherwise, go to step 1.

This hybrid can be viewed as a variant of RVI, with step 2 being a boost in the update of function values \( h^t \) in hope of arriving at convergence more quickly. Alternatively, it can also be thought of as a variant of PI, with step 1 being a modified policy improvement step that yields a more significant policy update. While there are no theoretical guarantees on convergence or convergence rates, we found the hybrid implementation to produce the optimal policy faster than the aforementioned standard formulations of RVI or PI. The improvement is likely due to the fact that much fewer iterations are needed for the hybrid method than either RVI or PI, although each iteration may take slightly longer.

**Computational complexity**

The size of this state space serves as a lower bound on the number of operations required to compute the optimal solution using the aforementioned DP solution methods. Recall that one way to bound the state space is to use (3.12), with inventory bounds \( \{M_n\}_{n=1}^N \) given by (3.14). This bounded state space has a size of \( |X| = M_1 M_2 \ldots M_N \), which can be computationally prohibitive when the number of fulfillment centers or the expected demand is large.
3.2.5 Empirical analysis

In this section, we present numerical results for the aforementioned dynamic programming model and the solution methods. We design experiments to investigate the following questions:

1. What is the performance of the three solution methods, RVI, PI and hybrid RVI-PI?

2. How does the performance change with the scale of the problem?

3. What does the optimal policy look like?

4. What is the effect of non-identical lead times?

Below, we first describe a small numerical example on which the experiments are conducted, and illustrate how to compute its bounds according to Section 3.2.3. In the subsequent sections, we then provide an empirical analysis for each of the questions of interest.

A small numerical example

Consider a small online-retail inventory system that has $N = 2$ fulfillment centers and $R = 2$ customer regions. The overage and underage costs are $c_o = $1 and $c_u = $12, respectively. The shipping costs are given by $c_{11} = c_{22} = $5 and $c_{12} = c_{21} = $6. The review period is one week (7 days), and the average weekly demand is 1 for each region, for a system-wide total demand of $D = 2$ units per week.

The two fulfillment centers are symmetrical except for their lead times. Unless otherwise stated, the default lead times in our experiments are $L_1 = 3$ days and $L_2 = 5$ days.

Computation of bounds

For the small numerical example, if we assume that spillovers are not allowed, then the shipping cost is $c_n = 5$ for all the demand that is fulfilled, and the expected
Table 3.1: Comparison of the optimal policy and Newsvendor base-stock policy

<table>
<thead>
<tr>
<th>Policy</th>
<th>Overall cost</th>
<th>Shipping</th>
<th>Stockout</th>
<th>Holding</th>
<th>Lost sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td>13.037</td>
<td>9.766</td>
<td>1.205</td>
<td>2.066</td>
<td>0.1004</td>
</tr>
<tr>
<td>Newsvendor base-stock</td>
<td>13.422</td>
<td>9.971</td>
<td>0.467</td>
<td>2.983</td>
<td>0.0389</td>
</tr>
</tbody>
</table>

weekly demand is \( d_n = 1 \), for both fulfillment centers \( n = 1, 2 \). Therefore, both have a Newsvendor critical fractile of

\[
\frac{c_u - c_n}{c_u + c_a - c_n} = \frac{12 - 5}{1 + 12 - 5} = 0.875,
\]

and the inventory bounds are given by (3.14) as

\[
M_1 = \min \left\{ q \in \mathbb{Z}_+ \mid F_{(1+\frac{5}{d_1})q}(q) \geq 0.875 \right\} = 3,
\]

\[
M_2 = \min \left\{ q \in \mathbb{Z}_+ \mid F_{(1+\frac{5}{d_2})q}(q) \geq 0.875 \right\} = 3.
\]

The cost\(^3\) of the Newsvendor base-stock policy (3.13) can then be found by policy evaluation (Equation (3.21)). The results are given in Table 3.1. In this case, it happens to be only 2.95% above the optimal cost. As expected, the holding cost is higher than optimal. Note also that the average number of lost sales per period is given by \( \frac{0.467}{12} = 0.0389 \), which corresponds to a service level of \( \frac{2-0.0389}{2} = 0.981 \). This is much higher than the critical fractile of 0.875, and also higher than \( F_{(1+\frac{5}{d_1})q}(M_1) = 0.943 \) and \( F_{(1+\frac{5}{d_2})q}(M_2) = 0.905 \), due to the fact that local stockouts may indeed spill over to other fulfillment centers.

For the lower bound, the minimum expected shipping cost is \( \bar{s} = 0.5 \times 5 + 0.5 \times 5 = \$5 \) per unit, which gives a lower bound of \( \bar{s} \cdot D = \$10 \) on the overall cost. This is 23.3% below the optimal cost, showing that the lower bound can indeed be quite loose.

\(^3\)This is the infinite-horizon average cost under the actual conditions of the problem, i.e., assuming spillovers, lost sales and real-time demand fulfillment, and not to be confused with the assumptions made when calculating the Newsvendor policy.
Comparison of solution methods

For our numerical example, it turns out that the recurrent states of the optimal policy are contained in the state space given by (3.12) with at most \( M = 3 \) units per fulfillment center. We apply the three exact DP solution methods on this state space, and compare their performance in terms of convergence over time.

Figure 3.3 shows convergence behavior of the three methods, including the average cost estimate, \( g^t \), and the maximum change in value function, \( \delta^t \) (defined in (3.20)), for each iteration \( t \). In the figure, each dot represents an iteration, and the methods are marked with different colors.

The following observations can be made for each method:

- Relative value iteration (RVI) settles upon a good estimate of \( g^t \) in a few iterations, but the value function \( h : X \to \mathbb{R} \) takes a relatively long time to converge. This is because each iteration of RVI takes a relatively long time to compute (recall that it updates \( h(x) \) by optimizing over all possible policies \( u \in U(x) \) for every \( x \in X \) in every iteration).

- Policy iteration (PI) terminates with the optimal policy after checking 4 policies, as shown by the sawtooth behavior of \( \delta^t \). The rise in each sawtooth indicates that a new policy has been found in the policy improvement step, and the subsequent decrease corresponds to the policy evaluation step. (Recall that an iteration of the policy evaluation step is much faster than an iteration of RVI, because it focuses on one policy rather than optimizes over all possible policies.)

- The hybrid RVI-PI method follows the same trajectory as RVI until it converges on a policy, and then applies policy evaluation (as shown in the rapid drop of \( \delta^t \)). In this instance, the first convergent policy found by RVI happens to be optimal; however, this need not be the case in general.

In conclusion, the hybrid method appears to get the best of both worlds by combining RVI and PI. Like RVI, it maintains a good estimate of \( g^t \), and like PI, it focuses its computational efforts on evaluating a single promising policy rather than checking
all possible policies. We observe similar behavior in exploratory tests on other small numerical examples.

**Scaling up**

To assess the scalability of the exact DP methods, we test various state space sizes by varying the maximum amount of inventory per fulfillment center $M$ from 1 to 9, i.e., by solving for the optimal policy on the state space

$$X = \{x \in \mathbb{Z}_+^N \mid x_n \leq M, n = 1, \ldots, N\}.$$  

Figure 3.4 shows the computation time and objective cost for different values of $M$.

The results highlight the trade-off associated with the size of the state space. In this example, the computation time for all the methods grows quickly with $M$, showing that they can be computationally prohibitive for a large state space. However, when $M < 3$, the state space is too small and does not contain all the recurrent states of the actual optimal policy, and therefore, the objective cost is suboptimal.

For this small example, $M = 3$ gives us the optimal policy in a short computation time. In practice, however, with greater expected demand and more fulfillment centers, the smallest $M$ that includes all optimal recurrent states will likely be too large to compute with the exact solution method. This motivates the need for large-scale solution schemes, which we propose in the next section.

**Optimal solution structure**

Table 3.2 shows the optimal policy for the small numerical example, i.e., the optimal order quantity $u = (u_1, u_2)$ as a function of the state $x = (x_1, x_2)$. As expected, the optimal order quantity decreases as the amount of on-hand inventory increases. However, beyond that, the optimal policy has a rather complex form. The policy for each fulfillment center $n$ is not only a function of its own inventory position $x_n$, but that of other fulfillment centers as well. In other words, the fulfillment centers affect each other, and thus to be treated as a whole; decoupling them for individual
Figure 3.3: Among the three exact DP solution methods described, the hybrid method captures the best of both worlds by combining relative value iteration (RVI) and policy iteration (PI).

(a) Average cost estimate $g^t$ over time

(b) Maximum value change $\delta^t$ over time$^a$

$^a$Each dot represents an iteration $t$, and the dotted line is the termination threshold of $10^{-10}$. 

151
Figure 3.4: The exact DP methods are computationally prohibitive when the state space is too large. However, the state space may need to be large in order to contain the optimal solution.

(In this example, the size of the state space grows exponentially with $M$, the maximum amount of inventory per fulfillment center. The optimal solution is captured by $M \geq 3$.)

(a) Computation time for various state space choices

(b) Objective cost for various state space choices
optimization does not capture the full complexity of the problem.

That being said, to maintain tractability, it may still be a good idea to consider decoupled policies of the form $u_n = \mu_n(x_n)$ as an approximation, and account for the interdependence of fulfillment centers by optimizing all the $\mu_n$ at the same time. Figure 3.5 depicts the relationship of $u_n$ and $x_n$ when the inventory position in the other fulfillment center is fixed. Some of these resemble base-stock policies (e.g., $\mu_1$ for $x_2 = 2, 3$, and $\mu_2$ for $x_1 = 3$, all with a base-stock level of 2); some can be described as a base-stock policy that is capped at some constant-order level (e.g., $\mu_1$ for $x_2 = 0$, and $\mu_2$ for $x_1 = 0, 1$, both of which have a base-stock level of 3 and a constant ordering limit of 2). Both of these are promising classes of parameterized policies, and we further investigate them in the next section.

Benefits of lead time staggering

We now provide numerical evidence that staggered lead times can improve the overall performance. In particular, we pose the following question: Is there any instance where the overall cost is improved when we increase the lead time of some fulfillment centers in order to stagger the lead times? For a single fulfillment center, a shorter lead time corresponds to a lower cost because it means having less uncertainty to hedge against. However, in the online-retail setting where multiple fulfillment centers carry fungible inventory, it may be worthwhile to set a longer lead time for some fulfillment centers so as to reap the benefits of lead time staggering.

To illustrate this effect, we compare the expected costs of the default scenario of staggered lead times, $L_1 = 3, L_2 = 5$, with the case of identical lead times, $L_1 = L_2 = 3$. The overall cost and its breakdown are given in Table 3.3.

According to the table, the expected cost for staggered lead times is lower than when $L_2$ is shorter and identical to $L_1$. The cost breakdown further reveals that case of staggered lead times has lower stockout and holding costs, i.e., it achieves a higher service level with less safety stock, compared to the case of identical lead times. This provides evidence that the benefit of lead time staggering may outweigh the cost that comes with increasing lead times.
Table 3.2: Optimal policy of the small numerical example

(a) \( u_1 = \mu_1(x_1, x_2) \)

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(b) \( u_2 = \mu_2(x_1, x_2) \)

<table>
<thead>
<tr>
<th>( x_2 )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 3.5: The optimal policy has a structure that resembles the hybrid of base-stock and constant-order policies.

(a) \( u_1 = \mu_1(x_1) \) for fixed values of \( x_2 \)

(b) \( u_2 = \mu_2(x_2) \) for fixed values of \( x_1 \)

Table 3.3: Staggered lead times results in lower costs

<table>
<thead>
<tr>
<th>Lead times</th>
<th>Overall cost</th>
<th>Shipping</th>
<th>Stockout</th>
<th>Holding</th>
</tr>
</thead>
<tbody>
<tr>
<td>( L_1 = 3, L_2 = 5 ) (staggered)</td>
<td>13.037</td>
<td>9.766</td>
<td>1.205</td>
<td>2.066</td>
</tr>
<tr>
<td>( L_1 = L_2 = 3 ) (identical)</td>
<td>13.131</td>
<td>9.687</td>
<td>1.253</td>
<td>2.190</td>
</tr>
</tbody>
</table>

154
3.3 Approximation schemes for large-scale problems

As established in the previous section, the exact DP solution methods suffer from the curse of dimensionality when the state space is large, and the structure of the optimal solution can be difficult to interpret or implement. In this section, we address these complexities by developing two approximation schemes in the respective spaces of policies and function values. Section 3.3.1 presents three classes of policies that can be described with parameters; by restricting our attention to these classes of policies, policy optimization is simplified into the problem of finding the best policy parameters. Section 3.3.2 describes the use of simulation to estimate the infinite-horizon average cost, which is much more scalable than exact evaluation. In Section 3.3.3, we combine these two approximation schemes and propose three simulation-based optimization methods for finding near-optimal policies.

3.3.1 Approximation in policy space: Policy parameterization

In this section, we introduce three classes of parameterized policies. The first two, base-stock and constant-basestock hybrid policies, are both of the form \( u_n = \mu_n(x_n) \), i.e., the order quantity of a fulfillment center \( n \) only depends on its own inventory position \( x_n \), and not on the inventory position of other fulfillment centers. The third one, projected base-stock, has the form of \( u_n = \mu_n(x) \), i.e., the order quantity is a function of all fulfillment center inventory positions. All three classes can be described by a relatively small set of parameters. Restricting to these classes of policies reduces the optimization problem to the more tractable problem of searching for the best parameters.

Base-stock policies

A base-stock policy is of the form

\[
\mu_n(x_n) = (b_n - x_n)^+,
\]
where the parameter $b_n \in \mathbb{Z}_+$ is the base-stock (or order-up-to) level, and $(\cdot)^+$ is the projection operator onto the nonnegative space. In other words, if the inventory position $x_n$ is below the base-stock level, order an amount that brings it up to the base-stock level; otherwise, order zero.

The base-stock policy is widely used in practice because of its simplicity. The optimal policy for a single-warehouse problem, under the assumption of zero lead time, zero fixed ordering cost, and backorders, is a base-stock policy (see Bertsekas (2005), Section 4.2). It has also been shown to have desirable properties in other settings (see, for example, Janakiraman and Roundy (2004) and Huh et al. (2009)).

When optimizing over the class of base-stock policies, it is reasonable to limit ourselves to the choice of $b_n \in \{0, 1, \ldots, M_n\}$ for every fulfillment center $n$, where $M_n$ is the Newsvendor base-stock policy parameter given by (3.14). Since the Newsvendor base-stock policy is a conservative policy that tends to hold more than the optimal amount of inventory, the optimal base-stock policy is unlikely to have base-stock levels that are greater.

The main drawback of base-stock policies is that it may order too much in the case where inventory is low relative to lead time demand, i.e., when a local stockout is likely to happen before the replenishment order arrives. For this reason, base-stock policies are more suitable when unmet demand results in backorders rather than lost sales. In our situation, due to lost sales, we are motivated to consider more complex policies, described below, that overcome the drawback of base-stock policies.

**Constant-basestock hybrid policies**

Empirical results from the previous section suggests that, in the optimal policy, the ordering quantity is capped at a constant order level when the inventory level is low. This phenomenon can be explained by observing that in every case where the on-hand inventory is low at the review epoch, then all the inventory will likely be used up before the replenishment order arrives. Therefore, all such cases are likely to have the same future states beyond the replenishment lead time, and should thus have the same optimal order quantities. In other words, the optimal order policy is to bring
the inventory position back to the best point for “starting out fresh” at the lead time. This is also consistent with the finding of Goldberg et al. (2014) that in the single-warehouse case, constant-order policies are asymptotically optimal with respect to large lead times.

This leads us to consider the following hybrid of constant and base-stock policies:

\[
\mu_n(x_n) = \min(\beta_n, (b_n - x_n)^+) = \begin{cases} 
  \beta_n, & x_n \leq b_n - \beta_n, \\
  (b_n - x_n)^+, & x_n > b_n - \beta_n. 
\end{cases}
\]

This class of policies has two parameters for each fulfillment center \(n\): the base-stock level \(b_n\) and the constant-order level \(\beta_n\). When \(x_n\) is small (namely, when \(x_n \leq b_n - \beta_n\)), the policy is a constant-order policy; otherwise, it is a base-stock policy.

To obtain a bound on the optimal constant-base stock policy parameters, we may again use \(M_n\) as the upper bound on \(b_n\) for every \(n\). As for \(\beta_n\), recall that it reflects the desired amount of inventory after the next inventory replenishment order arrival, and is to last until the next replenishment, i.e., after one review period. Therefore, similar to (3.14), an upper bound on \(\beta_n\) is the amount of inventory that achieves a conservatively high service level (by assuming no spillovers) for one review period:

\[
\beta_n \leq M_n' = \min \left\{ q \in \mathbb{Z}_+ \mid F_{d_n}(q) \geq \frac{c_u - c_n}{c_o + c_u - c_n} \right\}, \quad n = 1, \ldots, N. \tag{3.23}
\]

**Projected base-stock policies**

Yet another class of policies is the projected base-stock policy proposed in Acimovic and Graves (2016), which is characterized by the set of projected base-stock levels \(\beta_n\) for all \(n\), i.e., the desired inventory position at each fulfillment center after replenishment. The order quantity for fulfillment center \(n\) is

\[
\mu_n(x) = (\beta_n - x'_n)^+,
\]

where \(x'_n\) is the expected inventory position at fulfillment center \(n\) at its lead time \(L_n\), right before replenishment.
$x'_n$ is a function of both $x$ and $u_{n'} = \mu_{n'}(x)$ for $n' < n$, the order quantities of all the fulfillment centers that were replenished before $L_n$. It is nontrivial to characterize because of lead times and spillovers. For the case of identical lead times, Acimovic and Graves (2016) suggest a heuristic for computing $x'_n$ that uses a transportation problem to approximate the effect of spillovers. In our case of non-identical lead times, the values of $x'_n$ can be computed exactly from the dynamic programming formulation, or for larger problems, by simulation, as we describe in Section 4.

Note that we have deliberately chosen the notation $\beta_n$ for both the projected base-stock level and the constant-order level of the constant-basestock hybrid policy. This highlights the fact that both quantities reflect the desired inventory position immediately after the replenishment order is received at the lead time. It has the same upper bound given by (3.23).

3.3.2 Approximation in value space: Simulation

In this section, we present a discrete event simulation for evaluating a given policy. In addition to effectively estimating the infinite-horizon average cost, we also develop a method of estimating “gradient-like” information that indicates a descent direction in which we are likely to find policies of lower costs. The simulation results are validated on a small numerical example.

Discrete event simulation design

We now describe a discrete event simulation for evaluating the cost of a given replenishment policy $\mu = (\mu_n)_{n=1}^N$. Recall from Section 3.2.2 that each review period is divided into $N + 1$ segments, indexed by $n = 0, 1, \ldots, N$, using lead times $L_n$ as breakpoints, and that $L_0$ and $L_{N+1}$ denote the beginning and the end of a review period (i.e., the current and the next review epochs), respectively.

Algorithm 3.3.1 provides details on how the activities in a review period are simulated. There are three types of discrete events: replenishment order placement, replenishment order reception, and demand arrival. The first type occurs on a regu-
lar basis at the review epoch, and the second at the lead times, while demand arrivals take place during each segment.

Algorithm 3.3.2 provides a computationally efficient method for generating a sequence of demand arrivals. This algorithm takes advantage of the fact that Poisson processes are additive and memoryless, so that we could first sample the total demand for the entire review period, and then sample the segment and region of each demand arrival. This requires only one sample of the Poisson random variable, and is generally faster than independently sampling multiple independent Poisson processes for each segment and region.

Let $k$ be a nonnegative integer that denotes the index of a period, and let $\hat{g}_k^\mu = \hat{g}_s + \hat{g}_u + \hat{g}_o$ be the overall cost simulated by Algorithm 3.3.1 for period $k$, where $\hat{g}_s, \hat{g}_u, \hat{g}_o$ are the simulated shipping, stockout (underage) and holding (overage) costs, respectively. The sample average of $K$ periods is given by

$$\bar{g}_K^\mu = \frac{1}{K} \sum_{k=1}^{K} \hat{g}_k^\mu.$$ 

By the law of large numbers, as $K$ approaches infinity, $\bar{g}_K^\mu$ converges to the infinite-horizon average cost $g_\mu$ of policy $\mu$. This class of simulations that focuses on the behavior of a single replication in the limit of infinite periods is called steady-state simulations, in contrast with the alternative of transient simulations where the subject of interest is the sample average taken over multiple replications of finite length.

In practice, since we have to terminate the simulation at some point, the choice of $K$ becomes an important issue. How many periods do we need to simulate before we are confident with representing the true cost by the sample average? Related to this is the question of how to establish the confidence interval of the sample average, which relies on having a good estimator of the variance of the sample average, i.e., the variance of $\lim_{K \to \infty} \bar{g}_K^\mu$, treating the steady-state simulation outcome. (Note that this is distinct from the variance of $\hat{g}_k^\mu$, which is the simulated cost of each period.) One popular variance estimator is the non-overlapping batch means estimator, which divides the simulation periods into batches of $m$ periods, computes the sample average
of each batch, i.e., the batch mean, and use the variance of the batch means as the variance estimator. It can be shown that the bias of the non-overlapping batch means variance estimator approaches zero as the batch size $m$ grows large; see Goldsman and Nelson (2006) for more details on variance estimators.

The appropriate choice of simulation length $K$ and batch size $m$ depends on the problem. We illustrate the trade-offs empirically in Section 3.4 with a numerical example.

**Algorithm 3.3.1** Discrete event simulation for a single period

1: **Inputs:** Inventory position $x$ at the end of the previous period; replenishment policy $\mu$
2: $\hat{g}_s \leftarrow 0$, $\hat{g}_u \leftarrow 0$, $\hat{g}_o \leftarrow 0$ $\Rightarrow$ Simulated shipping, stockout and holding costs, respectively
3: $u = \mu(x)$ $\Rightarrow$ Vector of replenishment orders for each fulfillment center
4: Generate $\{d_n\}_{n=0}^N$ using Algorithm 3.3.2
5: Perform `DemandFulfillment($d_0$)` $\Rightarrow$ Segment 0
6: for $n = 1, \ldots, N$ do $\Rightarrow$ Segment $n$
7: $\hat{g}_o \leftarrow \hat{g}_o + c_o x_n$ $\Rightarrow$ Charge holding cost
8: $x_n \leftarrow x_n + u_n$ $\Rightarrow$ Replenishment order arrives
9: Perform `DemandFulfillment($d_n$)`
10: end for
11: return $x, \hat{g}_s, \hat{g}_u, \hat{g}_o$

12: **procedure** `DemandFulfillment($d_n$)`
13: for $r$ in $d_n$ do $\Rightarrow$ Let $n_1, \ldots, n_N$ be the fulfillment center indices sorted by increasing shipping costs to $r$, i.e., $c_{n_1,r} \leq c_{n_2,r} \leq \cdots \leq c_{n_N,r}$
14: for $n = n_1, \ldots, n_N$ do
15: if $x_n > 0$ then $\Rightarrow$ Able to fulfill demand
16: $\hat{g}_s \leftarrow \hat{g}_s + c_{nr}$
17: $x_n \leftarrow x_n - 1$
18: return
19: end if
20: end for
21: $\hat{g}_u \leftarrow \hat{g}_u + c_u$ $\Rightarrow$ System-wide stockout
22: end for
23: end procedure
**Algorithm 3.3.2** Poisson demand sequence generation

**Require:** \( D \), the expected total demand per review period

1: \textbf{for} \( n = 0, \ldots, N \) \textbf{do} \\
2: \hspace{1em} \( d_n \leftarrow \) empty sequence \( \Rightarrow \) Sequence of regions from which demand originates \\
3: \textbf{end for} \\
4: Sample \( d \), the total number of demand arrivals, from the distribution \( \text{Poisson}(D) \) \\
5: \textbf{for} \( i = 1, \ldots, d \) \textbf{do} \\
6: \hspace{1em} Sample \( n_i \), the time segment in which the demand arrives, from the distribution \\
7: \hspace{2em} \( \mathbb{P}(n_i = n) = \frac{L_{n+1} - L_n}{L}, n = 0, \ldots, N \) \\
8: \hspace{1em} Sample \( r_i \), the region from which the demand originates, from the distribution \\
9: \hspace{2em} \( \mathbb{P}(r_i = r) = \lambda_r, r = 1, \ldots, R \) \\
10: \textbf{end for} \\
11: \textbf{return} \( \{d_n\}_{n=0}^N \) \\

**Gradient-like information**

In the space of continuous variables, gradients give rise to a class of iterative optimization methods that are extensively studied and widely used in both the deterministic and stochastic settings. We are interested in extending the idea of gradients to develop discrete optimization via simulation (DOvS) methods that take advantage of gradient-like information. In this section, we describe how gradient-like information can be obtained as a natural by-product of discrete event simulation, and in Section 3.3.3, we propose one way of applying gradient-like information to the DOvS method of random search to improve its efficiency.

The notion of gradient is not well defined in discrete space. Nevertheless, it is still possible to extend continuous-space gradient estimation techniques, such as those reviewed in Fu (2006), to estimate some gradient-like information for our problem. Particularly pertinent to our case is the technique of *perturbation analysis*, which
is the method of dividing the underlying system into several subsystems, estimating the input-output sensitivity of each subsystem by hypothetically perturbing its inputs, and combining the perturbation effects to estimate the overall gradient. Predicated upon some analytical knowledge of the problem, perturbation analysis has the advantage of being more computationally efficient than indirect gradient estimation techniques (such as finite difference) that require additional simulation effort. See Ho and Cao (1991) for a comprehensive study on perturbation analysis for discrete event dynamic systems, and Glasserman and Tayur (1995) for an example of applying perturbation analysis to the estimation of gradient-like (or sensitivity) information of base-stock levels, and thereby enabling their optimization, in a multi-echelon inventory system.

For the online-retail inventory replenishment problem, we would like to estimate the gradient-like information, or sensitivity, of the infinite-horizon average cost $g_\mu$ of a parameterized policy $\mu = \mu(\cdot; \theta)$ with respect to changes in the vector of policy parameters $\theta$, where $\theta \in \mathbb{Z}^N_+$ for base-stock and projected base-stock policies, and $\theta \in \mathbb{Z}^{2N}_+$ for constant-base-stock policies. Let $\theta_0 \in \mathbb{Z}$ be an element of $\theta$ (where the index 0 can represent any given entry), and let $\frac{\partial g_\mu}{\partial \theta_0} \in \mathbb{R}$ denote the gradient-like information of $g_\mu$ with respect to the single parameter $\theta_0$. We derive the estimation of $\frac{\partial g_\mu}{\partial \theta_0}$ with two steps of approximation. The first step is to approximate it with the sample mean taken from $K$ review periods of simulation,

$$
\frac{\partial g_\mu}{\partial \theta_0} \approx \frac{1}{K} \sum_{k=1}^{K} \frac{\partial \hat{g}_\mu^k}{\partial \theta_0},
$$

where $\hat{g}_\mu^k$ is the simulated cost of policy $\mu$ in the $k$-the period.

The second step is to apply perturbation analysis to estimate each $\frac{\partial g_\mu^k}{\partial \theta_0}$ term. We divide the underlying system into two independent subsystems: one with $\theta$ as input and replenishment orders $\{u_k\}_{k=1}^{K}$ as output, and the other with $\{u_k\}_{k=1}^{K}$ as input and $\hat{g}_\mu^k$ as output. The gradient-like information, or input-output sensitivity, of the former system can be derived by the definition of $\mu$, and the gradient-like information of the latter can be estimated by simulation. Combining the two then yields $\frac{\partial \hat{g}_\mu^k}{\partial \theta_0}$. 

162
Next, we provide a more formal treatment of how \( \frac{\partial \hat{g}_k}{\partial \theta_0} \) is computed from gradient-like information of the two subsystems. For a given period \( k \), there are two sets of replenishment orders that have a direct impact on the cost \( \hat{g}_k \): the order made in the previous period, \( u_{k-1} \), which affects the cost from the review epoch to the replenishment lead time; and the order made in the current stage, \( u_k \), received by each fulfillment center at its own lead time, and affects the cost thereafter. Thus, we can approximate \( \frac{\partial \hat{g}_k}{\partial \theta_0} \) by

\[
\frac{\partial \hat{g}_k}{\partial \theta_0} = \sum_{n=1}^{N} \left( \frac{\partial \hat{g}_k}{\partial u_{k-1,n}} \frac{\partial u_{k-1,n}}{\partial \theta_0} + \frac{\partial \hat{g}_k}{\partial u_{k,n}} \frac{\partial u_{k,n}}{\partial \theta_0} \right),
\]

where the term \( \frac{\partial \hat{g}_k}{\partial u_{k,n}} \) is the sensitivity of the \( k \)-th period cost \( \hat{g}_k \) with respect to \( u_{k,n} \), the \( k \)-th period order quantity of fulfillment center \( n \), and \( \frac{\partial u_{k,n}}{\partial \theta_0} \) is the sensitivity of \( u_{k,n} \) with respect to the parameter \( \theta_0 \). The terms related to \( u_{k-1,n} \) are defined similarly.

The \( \frac{\partial \hat{g}_k}{\partial u_{k,n}} \) terms can be estimated by augmenting Algorithm 3.3.1 with the following accounting operations: When any cost is charged, consider the marginal cost that would have been added if \( u_{k,n} \) (or \( u_{k-1,n} \)) were incremented by 1, and include it in \( \frac{\partial \hat{g}_k}{\partial u_{k,n}} \). There are three cases:

- If any holding cost is charged (Algorithm 3.3.1, Line 7), it implies that the fulfillment center has not experienced a local stockout since the last replenishment, and the marginal cost of ordering one more unit is \( c_o \).

- If there is a local stockout at some fulfillment center \( n \) (i.e., if Algorithm 3.3.1, Line 15 does not hold), the marginal cost of ordering one more unit for fulfillment center \( n \) is \( c_{nr} - c_{nr'} \), where \( r \) is the region and \( n' \) is the next closest fulfillment center, regardless of whether or not \( n' \) ends up fulfilling the demand. Note that if \( n' \) is unable to fulfill demand, it in turn incurs a marginal cost that is included in \( \frac{\partial \hat{g}_k}{\partial u_{k,n'}} \), based on the cost it would have saved compared with the next closest fulfillment center after it, and so on.

- If the above local stockout occurs at the farthest fulfillment center \( n_N \), then
there is a system-wide stockout, and the marginal cost of ordering one more unit for fulfillment center \( n_N \) is \( c_{n_N r} - c_u \).

These accounting operations are based on simulation results that are already available, so they require no additional simulation efforts and very little computational overhead.

The \( \frac{\partial u_{k,n}}{\partial \theta_0} \) terms can be computed analytically by the definition of \( u_{k,n} = \mu_n(x_k) \). For example, for the class of base-stock policies, we have \( \mu_n(x_k) = (b_n - x_{k,n})^+ \). Therefore, for the parameter \( \theta_0 = b_n \), we have

\[
\frac{\partial \mu_n(x_k)}{\partial b_n} = \begin{cases} 
1, & x_{k,n} \leq b_n, \\
0, & x_{k,n} > b_n.
\end{cases}
\]

In other words, if the parameter \( b_n \) were to increase by 1, then the order quantity would also increase by 1 if \( x_{k,n} \leq b_n \); otherwise, if \( x_{k,n} > b_n \) it would stay the same.

**Validation**

To validate our simulation design, we evaluate a collection of base-stock policies for the small numerical example described in Section 3.2.5, and compare results with exact policy evaluation using dynamic programming.

Specifically, we consider the base-stock policies with base-stock levels \( (b_1, b_2) \), where \( b_1 \) and \( b_2 \) are integers ranging from 0 to 5. The combination produces a total of 36 base-stock policies. For each policy, we evaluate it exactly using DP Equation (3.21), and also simulate it for 10,000 periods, with an additional 1,000 initial periods discarded to mitigate transient effects. On a personal laptop computer, the average evaluation time per base-stock policy is 1.22 seconds for exact DP evaluation, and 0.45 seconds for simulation evaluation. For all 36 base-stock policies evaluated, the simulated cost is within 1.8\% of the exact cost.

Figure 3.6 shows a side-by-side comparison of the exact and simulated costs as a function of policy parameters. In the figure, \( b_1 \) and \( b_2 \) are plotted along the \( x \) and \( y \) axes, respectively. The color of each grid indicates the infinite-horizon average cost of the corresponding base-stock policy. In both cases, the best base-stock policy is
\((b_1, b_2) = (2, 3)\) and is marked with an asterisk. Contour lines are added to aid the visualization and comparison of the discrete cost functions. For simulation evaluation, the red arrows indicate the negative direction and magnitude of gradient-like information for each policy.

The figure shows that our discrete event simulation produces an accurate approximation of cost function values. The gradient-like information also correctly points to a descent direction, and the magnitude reflects the “steepness” of the function, much like the gradient of a continuous function. The validity of the gradient-like information allows us to apply it in the optimization of policies, as we present in the next section.

Figure 3.6: For base-stock policies of a small numerical example, simulation accurately estimates of the cost function, and the gradient-like information (red arrows) correctly indicates a descent direction of the cost function.

3.3.3 Simulation optimization of parameterized policies

We now introduce three simulation-based optimization methods to find near-optimal policy parameters. The first method, random search, performs the most comprehensive search of the entire space of discrete feasible solutions. Our main contribution is that we enhance the state-of-the-art random search approach with gradient-like infor-
mation, thereby making it more efficient. The second method heuristically generates a relatively small subset of discrete solutions, and chooses the best among them using a standard simulation-based selection-of-the-best procedure. The third method uses a probabilistic rounding technique to relate to the discrete solution space with continuous parameters, which can be optimized more easily, for example with stochastic gradient descent. While the first method is expected to find the best solution efficiently, the other two methods each have their strengths and may be favorable under some circumstances, which we discuss in detail below.

Random search with gradient-based sampling

In discrete optimization via simulation (DOvS), when the feasible space is large, state-of-the-art methods rely on random search. Most random search methods operate by maintaining one (or more) current solution(s) that is iteratively updated via comparison with another candidate solution, selected according to some sampling scheme. Different random search methods are distinct in their features of (1) the candidate sampling scheme, (2) the method of comparing the current and candidate solutions, and (3) the way of estimating the optimal solution. See Andradóttir (2006a) and Nelson (2010) for reviews on random search.

Our goal is not to compare the wide variety of available random search methods and find the best for our problem. Instead, we choose to build upon a specific existing framework called Randomized Balanced Explorativ and Exploitatativ Search with Estimation (R-BEESE), proposed by Andradóttir and Prudius (2009), which uses a random coin toss in each iteration to determine whether candidate sampling is conducted according a global (explorativ) or local (exploitatativ) distribution. R-BEESE is an appropriate choice not only because it has been established to be competitive, but also because the framework is well-suited for enhancement with gradient-like information.

We propose an extension of R-BEESE that has gradient-based sampling as part of the local sampling scheme. A formal description of the method is given in Algorithm 3.3.3. Some key features are as follows:
Algorithm 3.3.3 R-BEESE with gradient-based sampling

Require: Global sampling distribution $G$; local sampling distributions $L_1$ (neighborhood) and $L_2$ (gradient-based), with respective sampling probabilities $p_1$ and $p_2$; number of periods $n_0$ to simulate per solution per iteration

1: $t \leftarrow 0$ \hfill \Comment{Iteration counter}
2: Draw a random sample from $G$ as the current solution $\theta_t$
3: Simulate $\theta_t$ for $n_0$ periods
4: $\Theta \leftarrow \{\theta_t\}$ \hfill \Comment{The set of visited solutions}
5: while stopping criteria not satisfied do
6: Draw a $[0,1)$-uniform random variable $p$ independent of everything else
7: if $p \in [0, p_1)$ then \hfill \Comment{Neighborhood sampling}
8: Draw a random sample $\theta'$ from $L_1(\theta_t)$
9: else if $p \in [p_1, p_1 + p_2)$ then \hfill \Comment{Gradient-based sampling}
10: Draw a random sample $\theta'$ from $L_2(\theta_t)$
11: else \hfill \Comment{Global sampling}
12: Draw a random sample $\theta'$ from $G$
13: end if
14: Simulate $\theta'$ for $n_0$ additional periods and update $\hat{g}(\theta')$
15: $\theta^* \leftarrow \argmin_{\theta \in \Theta} \hat{g}(\theta)$
16: Simulate $\theta^*$ for $n_0$ additional periods and update $\hat{g}(\theta^*)$
17: $\theta_{t+1} \leftarrow \argmin_{\theta = \theta_t, \theta', \theta^*} \hat{g}(\theta)$
18: $\Theta \leftarrow \Theta \cup \{\theta'\}$
19: $t \leftarrow t + 1$
20: end while
21: $\Theta' \leftarrow \{\theta \in \Theta \mid n(\theta) \geq n_0 \sqrt{t}\}$ \hfill \Comment{$n(\theta)$ is the number of periods $\theta$ has been simulated}
22: $\hat{\theta}^* \leftarrow \begin{cases} \argmin_{\theta^*} \hat{g}(\theta), & \Theta' \neq \emptyset \\ \theta_t, & \Theta' = \emptyset \end{cases}$ \hfill \Comment{Optimal solution estimate}
23: return $\hat{\theta}^*$

(1) $\theta_t$ denotes the current solution in iteration $t$; $\theta'$ denotes the candidate solution; $\theta^*$ denotes the best solution visited so far; and $\hat{g}(\theta)$ denotes the estimated cost (sample average) of a solution $\theta$.

(2) Lines 6-14 describe the randomized candidate sampling scheme. We use the global sampling distribution $G$ that selects each parameter from integers between
0 and the Newsvendor upper bound uniformly at random. We consider two local sampling distributions, both of which depend on the current solution \( \theta \in \mathbb{Z}_+^N \):

- Neighborhood sampling distribution \( L_1(\theta) \): Select a solution from the set

\[
\{ \theta \pm e_n \mid n = 1, \ldots, N \} \cap \mathbb{Z}_+^N
\]

uniformly at random. In other words, randomly pick a fulfillment center \( n \), and flip a fair coin to decide whether to increment or decrement its parameter; all other parameters remain unchanged. If the resulting parameter is negative, repeat the process until a valid policy is found.

- Gradient-based sampling distribution \( L_2(\theta) \): Let \( \nabla \in \mathbb{R}_+^N \) be the gradient-like information of \( \theta \), and denote its \( n \)-th element by \( \nabla_n \) (which is equal to \( \frac{\partial \theta_n}{\partial \theta_n} \) in the notation of Section 3.3.2). Let \( \|\nabla\|_\infty = \max_n |\nabla_n| \). Generate the candidate solution \( \theta' \) independently as follows: For every \( n = 1, \ldots, N \), if

\[
\frac{|\nabla_n|}{0.5} < \frac{\|\nabla\|_\infty}{1.5},
\]

then set \( \theta'_n = \theta_n \). Otherwise, draw a number \( p_n \) uniformly at random from the interval \([0, \|\nabla\|_\infty]\), and let

\[
\theta'_n = \begin{cases} 
\theta_n & \text{if } p_n > |\nabla_n|, \\
\theta_n - \text{sign}(\nabla_n) & \text{otherwise},
\end{cases}
\]

where the sign operator takes on the value of 1 if \( \nabla_n > 0 \), –1 if \( \nabla_n < 0 \), and 0 if \( \nabla_n = 0 \).

In other words, \( \theta' \) is generated by adjusting each entry of \( \theta \) in the descent direction estimated by \( \nabla \). We limit the magnitude of the adjustment to be at most 1 in each entry. The probability that the \( n \)-th entry is adjusted depends on \( |\nabla_n| \). The entry that is largest in magnitude is always changed by 1, while entries that are relatively small, according to (3.25), are never adjusted.
The condition (3.25) ensures that after rounding, \( \theta' \) is approximately in the descent direction of \( \theta \).

Intuitively, if (3.25) holds for some \( n \) but we adjust \( \theta'_n \) by 1, then in order to remain consistently in the estimated gradient descent direction, the entry of \( \theta' \) with the largest gradient-like information magnitude should in fact be adjusted by more than 1, which is contrary to the adjustment limit of 1.

We remark that it is possible to consider greater adjustment limits, or other ways of using gradient-like information in candidate sampling. We leave this as an open topic for future research.

(3) Lines 15-16 is the estimation step (the last “E” in “R-BEESE”). By taking further samples of the best solution among visited solutions, we improve the accuracy of the estimation of its cost. This, in conjunction with current solution update and optimal solution estimation steps below, helps us focus simulation efforts on the most promising solutions.\(^4\)

(4) Line 17 updates the current solution by comparing the estimated costs of the current, candidate and best solutions. We prefer this straightforward comparison approach for its simplicity and efficiency, although it does not have the statistical guarantees of a rigorous selection-of-the-best algorithm. (For an example of using selection-of-the-best in random search, see Pichitlamken et al. (2006).)

(5) Lines 21-22 present the optimal solution estimate. As suggested by Andradóttir and Prudius (2009) and Andradóttir (2006b), we estimate the optimal solution with the best solution from the subset \( \Theta' \) that has been simulated sufficiently often. If \( \Theta' \) is empty, then the current solution is used.

The best choice of parameters \( p_1, p_2 \) and \( n_0 \) depends on the problem. Our exploratory tests suggest that \( p_1 = p_2 = \frac{1}{3} \) and \( n_0 = 5,000 \) periods form a reasonable

\(^4\)Note that the original R-BEESE proposed by Andradóttir and Prudius (2009) randomizes between candidate sampling (lines 6-14) and estimation (lines 15-16) using an additional parameter \( \alpha \), rather than perform both in every iteration as we do here. Our current version is devised to avoid having to tune \( \alpha \).
configuration that is robust to different instances of our problem. We terminate the search process after 200 iterations, observing that the optimal solution estimate improves relatively slowly thereafter. Appendix B.1 provides empirical evidence that gradient-based sampling improves the efficiency of R-BEESE.

Note that due to the inherent randomness of R-BEESE, a different solution is produced every time we run the algorithm, even if it starts at the same initial solution and executed for the same number of iterations. In our experiments, the final solutions obtained by independent trials are seldom the same, but they all have low costs. This is consistent with our previous observation that the cost function is flat near the optimum, and so there may be many near-optimal solutions.

**Target local service level heuristic**

Recall from Section (3.2.3) that the Newsvendor base-stock policy parameters are obtained by Equation (3.14) using critical fractiles that reflect conservatively high local service levels. This begs the question of what the optimal target local service levels should be. Unfortunately, optimizing the target local service levels for all fulfillment centers is just as difficult as optimizing the parameterized policy itself, because for each fulfillment center, the relationship between the target service level and parameterized policy is uniquely defined.

Nevertheless, we can simplify the problem by imposing the condition that all fulfillment centers have a common target local service level, denoted by $\alpha$, which is the service level at each fulfillment center if there were no spillovers. Then, the problem is reduced to the optimization of the single parameter $\alpha$. We refer to this as the *target local service level heuristic*, or the $\alpha$ heuristic in short.

Given a common target local service level $\alpha$, we define the corresponding base-stock policy parameters as

$$b_n(\alpha) = \min \left\{ q \in \mathbb{Z}_+ \left| F_{(1+\frac{L}{P})d_n}(q) \geq \alpha \right. \right\}, \quad n = 1, \ldots, N,$$

where $F_{(1+\frac{L}{P})d_n}$ is cumulative distribution function for the local demand in the dura-
tion of the lead time plus review period. In other words, \( b_n(\alpha) \) is the minimum amount of inventory required to achieve a local service level of \( \alpha \) assuming that spillovers are not allowed. This interpretation is similar to that of (3.14). Similarly, for constant-basestock hybrid policies and projected base-stock policies, the parameters are given by

\[
\beta_n(\alpha) = \min\{q \in \mathbb{Z}_+ \ | \ F_{d_n}(q) \geq \alpha\}, \quad n = 1, \ldots, N.
\]

The target local service level heuristic is to select the best policy from the collection of policies that correspond to all possible values of \( \alpha \), or the set of \( \alpha \) policies in short, given by

\[
\Theta = \{\theta(\alpha) \ | \ \alpha \in [0, 1]\},
\]

where \( \theta(\alpha) = (b_n(\alpha))_{n=1}^N \) for base-stock policies, \( \theta(\alpha) = (b_n(\alpha), \beta_n(\alpha))_{n=1}^N \) for constant-base-stock hybrid policies, and \( \theta(\alpha) = (\beta_n(\alpha))_{n=1}^N \) for projected base-stock policies.

Before proceeding to describe the optimization of \( \alpha \) policies, we highlight a few clarifications on the concept of the target local service level:

- The local service level is distinct from the system service level. There is no such distinction in the classical single-warehouse model, but the situation is more complex with a network of fulfillment centers. The local service level of a fulfillment center is defined as the probability of not experiencing a local stockout during a review period, whereas the system service level is the probability of not experiencing a system-wide stockout that results in lost sales. Note that local stockouts may result in demand spillovers to other fulfillment centers, but not necessarily in lost sales. Therefore, the system service level is typically higher than the local service levels.

- The target local service level is distinct from the actual local service level. The former assumes that spillovers are not allowed, and is used for analytical calculations; the latter depends on the actual performance of the system and may be difficult to characterize, but can be measured with simulation. Because of spillovers, each fulfillment center usually ends up serving more demand and
experiencing local stockouts more often than the target service level projects, so the actual local service level is typically lower than the target local service level at each fulfillment center.

• The Newsvendor base-stock policy is not necessarily a special case contained in \( \Theta \). In the Newsvendor base-stock policy, each fulfillment center has a different target local service level, which is given by its critical fractile \( \frac{c_u - c_n}{c_o + c_u - c_n} \) and depends on the local expected shipping cost \( c_n \). In contrast, every policy in \( \Theta \) has the same target local service levels for all fulfillment centers.

To generate the set \( \Theta \), note that it is not necessary to check every possible value of \( \alpha \in [0, 1] \), because an interval of \( \alpha \) values may correspond to the same policy, and also because a very high service level (\( \alpha \) arbitrarily close to 1) is unlikely to be optimal. For example, for the class of base-stock policies, consider the break points

\[
S_n = \left\{ F\left(1+\frac{\alpha}{T_n}\right) d_n(q) \mid q = 0, 1, \ldots, M_n \right\}, n = 1, \ldots, N,
\]

where \( M_n \) is the upper bound on the base-stock level at fulfillment center \( n \) given by (3.14). The collection of breakpoints, \( S = \bigcup_{n=1}^{N} S_n \), divide the range of \( \alpha \in [0, \max S] \) into intervals, each of which corresponds to a unique base-stock policy. Similarly, for projected base-stock policies, the breakpoints are given by

\[
S'_n = \left\{ F_{d_n}(q) \mid q = 0, 1, \ldots, M'_n \right\}, n = 1, \ldots, N,
\]

where \( M'_n \) is given by (3.23). For constant-base-stock policies, the breakpoints are the union\(^5\) of all \( S_n \) and \( S'_n \).

To find the optimal policy in \( \Theta \), we apply an efficient selection-of-the-best procedure called \( \mathcal{KN}+ \), proposed by Kim and Nelson (2006). This procedure is guaranteed to select a policy whose cost is no more than \( \delta \) (which is called the indifference zone) above the best policy with a given probability of correct selection \( PCS \). It operates

---

\(^5\)To be precise, since there is redundancy in the way the constant-base-stock policy is defined (specifically, \( c_n = 0 \) results in zero order quantity regardless of the value of \( b_n \)), it is in fact sufficient to consider the union of breakpoints \( S'_n \) and \( \{ s \in S_n \mid s \geq \min S'_n \} \) for all \( n \).
by iteratively eliminating inferior policies and taking additional samples of the remaining candidates, until only one policy is left. We find $\delta = 0.1$ and $PCS = 0.95$ to be a reasonable choice of parameters for our purposes.

The target local service level heuristic would have an advantage over random search if the set of $\alpha$ policies $\Theta$ contains near-optimal solutions and is not too large to optimize over. The size of $\Theta$ depends on the problem, and scales with the bounds $\{M_n\}_{n=1}^N$ (and $\{M'_n\}_{n=1}^N$). When the bounds are large, one may wish to consider alternative optimization methods that are more efficient than $KN^+$, for example, one-dimensional optimization methods that search for the best value of $\alpha$. Exploratory numerical results (see Figure B.2a in Section 3.4) suggest that as $\alpha$ increases from 0 to 1, the cost of policies first decreases and then increases, which is a structure that can be exploited.

As for whether or not $\Theta$ contains near-optimal solutions, we can expect the affirmative when near-optimal solutions indeed have a common target local service level. This is the case, for example, when the additional cost incurred by spillovers is roughly the same for all fulfillment centers, which is often true in practice. Figure 3.7 gives a small numerical example that illustrates the effect of spillover cost on the optimality of $\alpha$ policies.

**Randomized rounding and stochastic gradient descent**

The two optimization methods above directly optimize over a set of discrete policies. We now propose an alternative approach that works with discrete policies indirectly, through the use of continuous parameters.

The approach relies on the technique of *randomized rounding*, which is first proposed by Raghavan and Thompson (1987) and widely used in the design and analysis of approximation algorithms for combinatorial optimization. Applied to our problem, the idea is to relax the integrality constraint of policy parameters, but still maintain the integrality of order quantities by rounding them according to a probability distribution.

Specifically, for a given policy parameter $\theta_0 \in \mathbb{Z}_+$, let $\tilde{\theta}_0 \in \mathbb{R}_+$ be its continuous
Figure 3.7: The set of α policies considered by the target local service level heuristic may not contain the optimal solution.

(a) For the small numerical example given in Section 3.2.5, the set of α base-stock policies (marked in red) contains the optimal base-stock policy \((b_1, b_2) = (2, 3)\) (marked with a white asterisk).

(b) Now consider the same small numerical example, but with the shipping cost \(c_{21}\) (i.e., between fulfillment center 2 and region 1) increased from $6 to $12; everything else is kept as before. This makes it more costly for demand to spill over from fulfillment center 1 (which serves region 1) to fulfillment center 2. As a result, the optimal solution is now \((b_1, b_2) = (3, 2)\) (marked with a white asterisk), where fulfillment center 1 holds more inventory than the original optimal solution, and fulfillment center 2 holds less. The new optimal solution is not contained in the set of α policies.
relaxation. Then, when computing the order quantity at the review epoch, take

$$\theta_0 = \begin{cases} 
\lceil \tilde{\theta}_0 \rceil, & \text{with probability } [\tilde{\theta}_0] - \tilde{\theta}_0, \\
\lfloor \tilde{\theta}_0 \rfloor, & \text{with probability } \tilde{\theta}_0 - [\tilde{\theta}_0],
\end{cases}$$

where the operators \([\cdot]\) and \([\cdot]\) indicate rounding down and rounding up to the nearest integer, respectively. The randomized rounding is done independently for each review epoch and each parameter.

The relaxation of parameters allows us to optimize them with the stochastic gradient descent method, which iteratively updates the parameters by

$$\theta_0 \leftarrow \theta_0 - a \frac{\hat{g}_{\mu}}{\partial \theta_0},$$

where \(\frac{\hat{g}_\mu}{\partial \theta_0}\) is the gradient-like information given in (3.24), and \(a \geq 0\) is a step size of our choice.

Randomized rounding and stochastic gradient descent are relatively easy to implement and interpret. However, the main drawback of the simple randomized rounding scheme described above is that it ignores any correlations between fulfillment center order quantities. Thus, it is not unlikely to end up with suboptimal policies after rounding. For example, in Figure 3.7a above, the relaxation \((\tilde{b}_1, \tilde{b}_2) = (2.5, 2.5)\) rounds to integral policies \((2, 3), (3, 2), (2, 2), (3, 3)\) with equal probability, but the first two policies are better than the last two. To use randomized rounding policies in practice, it may be necessary to consider correlated rounding schemes, or impose other guardrails and post-processing procedures to protect against suboptimal rounding.

### 3.4 Empirical analysis of approximation schemes

In this section, we present empirical results on a realistic inventory network to demonstrate the effectiveness of the approximation schemes proposed in the previous section.

In Sections 3.4.1-3.4.3, we first describe the numerical example, determine the appropriate simulation configuration, and establish that random search with gradient-
based sampling is the most efficient solution method among the three simulation optimization methods proposed in Section 3.3.3.

In the subsequent sections, we focus on the analysis of final solutions found by random search. Recall that random search produces different solutions given its random nature and given different initial solutions. We replicate random search 100 times, each with a different initial feasible solution, and treat the 100 final solutions as being drawn from a population of low-cost solutions. Analyzing these final solutions gives insights on the the appropriate choice of parameterized policies, the effects of lead time staggering, the correlations between fulfillment centers, and the cost sensitivity with respect to problem characteristics such as demand variability, overage cost, and underage cost parameters.

3.4.1 A realistic numerical example

To demonstrate the effectiveness of simulation for problems of realistic sizes, we create a numerical example from synthetic data based on publicly-available information of an actual online-retail inventory system. We remark that the solution methods we propose are not specific to this example, but are readily applicable to other problem instances provided that the same types of data are available.

The example consists of 15 fulfillment centers and 98 customer regions located across continental United States. The remainder of this section provides further details on how the numerical example is constructed.

Fulfillment centers. The 15 fulfillment centers used in our numerical example are positioned at publicly known Amazon fulfillment center locations.\(^6\) There are 88 fulfillment centers in the dataset as of December 2015, and we select a subset that collectively achieves the lowest possible expected shipping cost, under the simplifying assumption that spillovers are not permitted. Specifically, we solve the following

mixed integer linear program:

$$\begin{align*}
\min_{x,y} & \quad \sum_{n=1}^{N} \sum_{r=1}^{R} \lambda_r c_{nr} x_{nr} \\
\text{subject to} & \quad \sum_{n} x_{nr} = 1 \quad \forall r \\
& \quad 1 - x_{nr} \geq y_{n'} \quad \forall n' \text{ such that } c_{n'r} \leq c_{nr}, \forall n, \forall r \\
& \quad x_{nr} \leq y_{n} \quad \forall n, \forall r \\
& \quad \sum_{n} y_{n} = N \\
& \quad x_{nr}, y_{n} \in \{0, 1\} \quad \forall n, \forall r
\end{align*}$$

In this formulation, the binary decision variable $x_{nr}$ takes on the value of 1 if and only if region $r$ is served by fulfillment center $n$, and the binary decision variable $y_{n}$ takes on the value of 1 if and only if fulfillment center $n$ is selected. Recall from Section 3.2.1 that $\lambda_r$ is the fraction of demand originating from region $r$, and that $c_{nr}$ is the shipping cost from fulfillment center $n$ to region $r$, both of which are discussed in more detail below. The objective is to minimize the expected shipping cost to fulfill a unit demand. The first constraint states that every region is served by exactly one fulfillment center. The second constraint ensures that the fulfillment center serving a region is indeed the closest among all selected fulfillment centers (i.e., for every $r$, if $x_{nr} = 1$ for some $n$, then all $n'$ that are closer to the region $r$ than $n$ cannot be selected.) The third constraint says that a fulfillment center may serve a region only if it is selected. The fourth constraint controls the number of fulfillment centers selected, in our case $N = 15$.

The above problem can be solved to optimality within 40 seconds on a personal laptop using Gurobi Optimizer version 6.0.5. The resulting fulfillment center selection is illustrated in Figure 3.8.
Customer regions. The 98 customer regions are created with data from the 2010 US census.\footnote{United States Census Bureau, 2010 Census Gazetteer Files: Counties. http://www.census.gov/geo/maps-data/data/gazetteer2010.html. Accessed in December 2015.} The raw data contains 3221 counties, which we group into 98 regions by dropping the last two digits of the GEOID.\footnote{Geographic identifiers, or GEOIDs, are numerical codes that uniquely identify geographic areas for census data purposes. See https://www.census.gov/geo/reference/geoidentifiers.html for more details. While GEOIDs are generally assigned by geographical location, we remark that dropping the last two digits of the GEOID does not necessarily produce groups of counties that are contiguous with each other, but merely serves as a proxy.} This produces 102 regions (i.e., groups of counties), from which we remove the six that are not in continental US (one from Hawaii, two from Puerto Rico, and three from Alaska) and divide the greater Los Angeles counties (which has more than twice the population of the second largest region) into three regions (Los Angeles county, counties east of LA, and counties west of LA), resulting in 98 final regions. The longitude and latitude of each region is the population-weighted average of its counties. We assume that the demand in each region is proportional to its population. The location and demand of the regions are illustrated in Figure 3.8.

Shipping cost. The shipping cost structure is modeled after the two-day shipping services of UPS.\footnote{United Parcel Service of America, Inc. https://www.ups.com/.} We assume that the shipping cost is a function of the distance on the sphere of the Earth, which can be calculated from longitude and latitude values using the haversine formula. To capture the nonlinearity of shipping costs as a function of distance, we further assume that ground shipment is used for distances no greater than 1,000 miles; otherwise, second-day air shipment is used. The shipment mode threshold of 1,000 miles is determined by observing the two-day radius on a limited number of UPS ground service map queries.\footnote{UPS, U.S. Ground Maps. https://www.ups.com/maps. Accessed in October 2016.}

We then fit an affine function to each of the shipment modes for a 1-pound package according to 2016 UPS Daily Rates data.\footnote{UPS, Daily Rate and Service Guide. https://www.ups.com/content/us/en/shipping/cost/zones/daily_rates.html. Accessed in October 2016.} The data is given by zones, and we align the smallest (Zone 2) and largest (Zone 8) zones with a distance of 0 and 3,000 miles,
Figure 3.8: For empirical analysis, we construct a realistic numerical example that consists of 15 fulfillment centers and 98 customer regions. The default lead time scenario $L=\text{west}$ corresponds to having a supplier on the west coast.

Red triangles indicate fulfillment centers, indexed by $n = 1, \ldots, 15$. The numbers under the triangles are the lead times in days. The dots represent customer regions, and are sized according to the regional demand, which is assumed to be proportional to population.

respectively. The resulting shipping cost function is illustrated in Figure 3.9, and the actual shipping cost values $\{c_{nr} \mid n = 1, \ldots, N, r = 1, \ldots, R\}$ range from $\$7.02$ to $\$24.72$ per unit.

While this example only models two-day fulfillment service, we remark that it is possible to capture multiple classes of fulfillment services by creating additional regions for them, each with its own set of demand distributions and shipping costs.

**Stockout and holding costs.** We set the stockout cost to $c_u = \$25$ per unit and the holding cost to $c_o = \$0.25$ per unit per week, which satisfies our assumption that $c_{nr} \leq c_u + c_o$ for all $n$ and $r$ (i.e., that it is always cost-effective to fulfill a demand). This could correspond to a one-pound product, for example, that has a selling price of $\$50$ and a profit margin of $\$25$. 
System-wide demand. By default, the system-wide demand follows a Poisson process with a weekly average of $D = 10$.

Lead times. By default, we assume that the lead times are as depicted in Figure 3.8. The lead time increases from 1 to 6 days linearly from west to east. This corresponds to having a supplier on the west coast. For ease of future reference, this scenario is labeled $L=west$. The review period is assumed to be 7 days.

Bounds. For this numerical example, the lower bound on the cost is 71.071, calculated from Equation (3.15). The Newsvendor base-stock policy (3.13) has a cost of approximately 81.073 (evaluated by simulation, as described below), which is 14.07% above the lower bound. The base-stock levels for the fulfillment centers are, from west ($n = 1$) to east ($n = 15$),

$$(M_n)^{15}_{n=1} = (2, 3, 4, 2, 5, 4, 3, 5, 4, 4, 3, 5, 5, 3).$$

Figure 3.10 indicates the regional demand that is served by each fulfillment center in the computation of the Newsvendor base-stock policy, when spillover is assumed to be disallowed.

Random initial solutions. For the methods of random search and randomized rounding with stochastic gradient descent, we replicate the experiment 100 times with randomly generated initial solutions. Each random initial solution is generated by independently drawing each policy parameter uniformly at random from integers between 0 and its upper bound ($M_n$ or $M'_n$ for each $n = 1, \ldots, 15$).

3.4.2 Simulation configuration

Before conducting experiments, we need to address the questions posed in Section 3.3.2 regarding how the simulation should be set up: How many simulation periods is sufficient for obtaining a “good” estimate of the true infinite-horizon average cost? And how do we establish that an estimate is good?
Figure 3.9: The shipping cost function is a piecewise-linear function of distance, fitted according to UPS data for 1-pound packages.

![Graph showing shipping cost function for UPS data.](image)

Figure 3.10: A simple heuristic for comparison, the Newsvendor base-stock policy, is computed by assuming that each region is served by its closest fulfillment center, and that spillovers are not allowed. Each region is connected to its closest fulfillment center with a line. The black numbers under the triangles are the expected local demand at each fulfillment center.

![Map showing Newsvendor base-stock policy.](image)

Figure 3.11 shows results for using simulation to evaluate the Newsvendor base-stock policy of our numerical example, which is representative of similar behavior.
observed with other policies. For this policy, each review period takes less than \(1.33 \times 10^{-4}\) seconds to simulate on a personal laptop computer.

**Simulation length.** The black line in Figure 3.11a indicates the sample average over the simulation periods. As we can see, the sample average exhibits some transient behavior, but appears to converge quickly.

Based on these results, for future experiments, we discard the first 1,000 periods for their transient behavior, and take the sample average of the next 20,000 periods as the cost estimate.

**Batch size.** Even though the sample average converges to the true infinite-horizon average cost, it is never going to stay constant, but will fluctuate around the true cost. So if we wish to use the sample mean as an estimate of the true cost after a finite number of simulation periods, it is helpful to have an understanding of how much the sample mean might fluctuate around the true cost. This is the idea behind the *confidence interval*.\(^\text{12}\)

Computation of the confidence interval relies on having an estimate of the variance of the sample mean. We use the *batch means* variance estimator, described in Section 3.3.2. The batch means over the simulated periods for different batch sizes is shown in Figure 3.11 in different colors. For example, for a batch size of 1, the batch means are the simulated costs of each period, which exhibits significant variation from period to period, by nature of the dynamics of the inventory system. As we increase the batch size, the batch means become better estimates of the infinite-horizon average cost, and the bias of its variance, compared to the true steady-state simulation variance, also approaches zero (see Goldsman and Nelson (2006), Equation (3)).

\(^\text{12}\)The confidence interval is defined as an interval in which the outcome of an experiment would land with a given probability, if the the experiment were to be repeated and the outcome treated as a random variable. For example, if we observed \(N\) outcomes of an experiment, denoted \(x_1, x_2, \ldots, x_N \in \mathbb{R}\), and we do not know the probabilistic distribution of the experiment outcome \(x_n\), then we can calculate the sample average, \(\bar{x} = \frac{1}{N} \sum_{n=1}^{N} x_n\), and the standard deviation of the samples, \(\sigma = \sqrt{\frac{1}{N} \sum_{n=1}^{N} (x_n - \bar{x})^2}\). Then, the 95% confidence interval is given by \(\bar{x} \pm \frac{t \sigma}{\sqrt{N}}\), where \(t = 12.706\) is the value such that \(1 - F_{n-1}(t) = 1 - 0.95\), where \(F_{n-1}\) is the cumulative distribution function of the Student’s t-distribution with \(n - 1\) degrees of freedom.
It turns out that for our simulation, the choice of batch size does not have a significant impact on the confidence interval, as long as the simulation length is much longer than the batch size. Recall that the length of the confidence interval is proportional to the standard deviation of batch means, and inversely proportional to the square root of the number of batches. Thus, for a fixed number of simulation periods, a larger batch size corresponds to better quality variance estimates (a smaller numerator) and fewer batches (a smaller denominator). For our simulation, these two effects roughly balance each other out. This is shown in Figure 3.12.

However, the choice of batch size is still important for the quality of the batch means variance estimator. Having a good variance estimator is particularly important for the target local service level heuristic, because it uses a selection-of-the-best procedure that relies on the variance estimator. If we choose a batch size that is too small, the variance would be overestimated, which would increase the computational effort of the selection-of-the-best procedure. On the other hand, if the batch size is too large, then it takes longer to simulate a batch of periods in order to update the variance estimator. For our problem, the batch size of 100 periods appears to be a reasonable choice for the target local service level heuristic.

\section*{3.4.3 Performance of simulation optimization methods}

\textbf{Setup.} In this section, we present empirical results on the performance of the three simulation-based policy optimization methods proposed in Section 3.3.3. The performance of a method is measured by the cost of the best policy achieved over time.

The setup of each method is as follows:

- Random search with gradient-based sampling, described in Algorithm 3.3.3, is implemented with $p_1 = p_2 = \frac{1}{3}$ and $n_0 = 5,000$, i.e., in each iteration, there is an equal probability of using neighborhood, gradient-based, or global sampling to generate a candidate, and the candidate as well as the best solution are each sampled 5,000 periods. For each new solution encountered, an initial 1,000
Figure 3.11: Exploratory results suggest that in our case, a reasonable configuration for steady-state simulation is a length of 10,000 periods (after discarding the initial 1,000 periods of transient effects), and a batch size of 100 for the batch means variance estimator.

(a) The black line indicates the sample average, which has some transient behavior but converges quickly. The dots represent batch means, with each color corresponding to a different batch size. We can see that the variance of batch means decreases with batch size.

(b) A larger batch size results in a smaller variance in batch means. (It has been shown that as the batch size continues to increase, the batch means variance estimator approaches the true variance of the steady-state simulation.)
Figure 3.12: The confidence interval for different batch sizes is comparable if the simulation length is much longer than the batch size (in this case, say, over 10,000 periods).

- The target local service level heuristic uses the $KN^+$ selection-of-the-best procedure, which is implemented with an indifference zone $\delta = 0.1$, probability of correct selection $PCS = 0.95$, batch means variance estimator of batch size 100 periods, and an initial sample of 50 batches (5,000 periods) for each $\alpha$ policy (after an initial 1,000 period is discarded to mitigate transient effects). We allow the procedure to run until only one candidate is left.

- For randomized rounding with stochastic gradient descent, in each iteration, the current solution is simulated for 5,000 periods (after an initial 1,000 period is discarded to mitigate transient effects). We perform 500 iterations of stochastic gradient descent, and the step size is $a = 0.1$.

For random search and randomized rounding with stochastic gradient descent, the per-iteration results reported below reflect the average across the 100 trials for each
iteration (i.e., the average cost and computation time for that iteration across the 100 trials).

**Overall performance comparison.** Figure 3.13 shows the optimality gap (relative to the lower bound) of the best policies, produced by the three methods over time. For the random search and stochastic gradient descent methods, the gap at each iteration is the average of 100 replicated experiments with random initial solutions. For the target service level, for each iteration, we plot the cost estimate of every policy that has not been eliminated. (In this case, there are 71 $\alpha$ policies to begin with; those with optimality gap over 20% are omitted from the figure.)

The figure shows that random search with gradient-based sampling produces the best policy in the shortest amount of time. The target service level heuristic also finds a policy of comparable cost, but it takes a long time to determine which is the best policy among the many $\alpha$ policies. Stochastic gradient descent converges quickly, but the policy is not as low in cost, possibly due to the fact that randomized rounding ignores fulfillment center correlations and can result in suboptimal policies, as discussed in Section 3.3.3.

Some additional details on the target local service level heuristic are given in Figure B.2 in Appendix B.2.

### 3.4.4 Performance of parameterized policies

**Setup.** We now focus on the final solutions produced by independent trials of random search with gradient-based sampling. Each trial starts with a different random initial feasible solution, and random search is executed for 200 iterations. We perform 100 trials for each of the three policy classes introduced in Section 3.3.1, namely, base-stock policies, constant-basestock hybrid policies, and projected base-stock policies. Each final solution is then evaluated with another independent simulation for 20,000 periods, and the outcome is recorded as an estimate of the true cost.
Among the three simulation-based optimization methods proposed, random search is the most competitive, producing the best solutions in the shortest time.

Comparison of costs. Table 3.5 reports the overall cost and the breakdown of cost components for each policy class, averaged over final solutions of the 100 trials. According to this table, the three classes of policies achieve similar near-optimality in terms of costs. Thus, if we are only concerned with finding low-cost solutions, there appears to be no reason to forgo the simplest class of base-stock policy, and use the more complex constant-basestock hybrid or projected base-stock policies.

Comparison of computational efforts. Table 3.4 reports the computation time for evaluating a policy of each class, which is a reasonable proxy for computational efforts. Recall that policy evaluation is performed by 21,000 periods of simulation (the first 1,000 being discarded for transient effects). Constant-basestock hybrid policies have some computational overhead compared to base-stock policies, as it has twice the number of parameters. The computation time for projected base-stock policies is an order of magnitude greater, due to the computation of the projected inventory level. However, this could potentially be accelerated if parallel processors are available.
Table 3.4: Average policy evaluation time

<table>
<thead>
<tr>
<th>Policy class</th>
<th>Time (sec)</th>
<th>Mean</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base-stock</td>
<td>1.37</td>
<td>0.034</td>
<td></td>
</tr>
<tr>
<td>Constant-basestock hybrid</td>
<td>1.78</td>
<td>0.088</td>
<td></td>
</tr>
<tr>
<td>Projected base-stock</td>
<td>15.49</td>
<td>0.44</td>
<td></td>
</tr>
</tbody>
</table>

Comparison of order quantity fluctuations. The base-stock policy may be competitive in terms of cost and computational effort, but the other two classes policies have other advantages. One such advantage is that constant-basestock hybrid or projected base-stock policies account for the case of a local stockout before the replenishment order arrives, and avoid ordering too much in such situations. In contrast, the base-stock policy ignores this effect, and can lead to persistent inventory imbalance and significant order quantity fluctuations.

Figure 3.14 show two ways to identify the policies’ differing abilities for mitigating order quantity fluctuations, including the coefficient of variation and relative difference. For a given policy, let \( \{u_n^k\}_{k=0}^{K-1} \) be the sequence of orders placed for fulfillment center \( n \) for the \( K \) simulated periods \( k = 0, \ldots, K - 1 \), and denote their average and standard deviation as \( \bar{u}_n = \frac{1}{K} \sum_{k=0}^{K-1} u_n^k \) and \( \sigma(u_n) = \sqrt{\frac{1}{K} \sum_{k=0}^{K-1} (u_n^k - \bar{u}_n)^2} \), respectively. Then the coefficient of variation is given by

\[
\text{Coefficient of variation of } \{u_n^k\}_{k=0}^{K-1} = \frac{\sigma(u_n)}{\bar{u}_n}, \quad n = 1, \ldots, N,
\]

and we define the relative difference as

\[
\text{Relative difference of } \{u_n^k\}_{k=0}^{K-1} = \frac{1}{K-1} \sum_{k=1}^{K} \frac{|u_n^k - u_{n-1}^k|}{\bar{u}_n}, \quad n = 1, \ldots, N.
\]

The coefficient of variation measures the overall fluctuation of the order quantities, and the relative difference measures the fluctuation between two consecutive periods. For each fulfillment center, we then take the average of these measures across the 100 solutions of the same policy class (after removing the policies with \( \bar{u}_n = 0 \)), and plot the results in the figure.
From Figure 3.14, we see that order quantity fluctuations for the constant-basestock hybrid and the projected base-stock policies are lower than that of the base-stock policy for the majority of the fulfillment centers (including \( n = 2, 3, 5, 6, 8, 9, 10, 12, 13, 14 \)), and comparable for the rest.

Table 3.5: Average cost of policies found by random search

<table>
<thead>
<tr>
<th>Policy class</th>
<th>Cost</th>
<th>Overall</th>
<th>Breakdown</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Average</td>
<td>Std</td>
</tr>
<tr>
<td>Base-stock</td>
<td>75.288</td>
<td>0.229</td>
<td></td>
</tr>
<tr>
<td>Constant-basestock hybrid</td>
<td>75.322</td>
<td>0.238</td>
<td></td>
</tr>
<tr>
<td>Projected base-stock</td>
<td>75.355</td>
<td>0.219</td>
<td></td>
</tr>
</tbody>
</table>

3.4.5 Lead time staggering effects

Setup. For the small numerical example in Section 3.2.5, we saw from Table 3.3 that lengthening some lead times to create staggering between fulfillment centers may result in a lower cost. We now design an experiment to investigate this effect for our realistic numerical example.

In this experiment, we test five lead time scenarios that vary in the level of staggering. In each scenario, the fulfillment center lead times alternate, from west to east, between 2 days and \( L' \geq 2 \) days. That is, the lead time for the westernmost fulfillment center, BFI, is set to 2 days, while the lead time of the second westernmost fulfillment center, OAK, is set to \( L' \); similarly, the lead time of ONT is always 2 days, while the lead time in PHX is \( L' \). The five scenarios corresponds to \( L' = 2, 3, 4, 5 \) and 6 days. This is illustrated in Figure 3.15.

These scenarios allow us to test whether the benefits of lead time staggering outweighs the cost associated with longer lead times. If spillovers were not allowed and the fulfillment centers were independent, a longer lead time has a strictly negative impact on the cost, and we would expect the cost to increase as \( L' \) increases. However, since the fulfillment centers are interdependent, lead time staggering may actually improve the cost.

For each lead time scenario, we perform 100 independent trials of random search
Figure 3.14: Compared to base-stock policies, constant-basestock hybrid policies and projected base-stock policies are better able to mitigate fluctuations in order quantities, measured by the coefficient of variation and the relative difference.

(a) Coefficient of variation

(b) Relative difference
Figure 3.15: We compare the scenario of identical lead times of 2 days (labeled $L=$same2) to staggered lead time scenarios (labeled $L=$stag3 to $L=$stag6) in which some fulfillment center lead times are longer than 2 days.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>$L'$ (days)</th>
</tr>
</thead>
<tbody>
<tr>
<td>same2</td>
<td>2</td>
</tr>
<tr>
<td>stag3</td>
<td>3</td>
</tr>
<tr>
<td>stag4</td>
<td>4</td>
</tr>
<tr>
<td>stag5</td>
<td>5</td>
</tr>
<tr>
<td>stag6</td>
<td>6</td>
</tr>
</tbody>
</table>

with base-stock policies. The final solutions are then evaluated by another independent simulation. The average cost of the final solutions are shown in Table 3.6, which indicates that the overall cost of staggered lead time scenarios are generally lower than the scenario $L=$same2 where all lead times are 2 days.

**Results.** Figure 3.16 provides greater evidence for the benefit of lead time staggering. In this figure, we plot the stockout and holding costs per period for the final solutions, with each scenario marked by a different color. We see that the staggered scenarios exhibit a better trade-off frontier between stockout and holding costs. In other words, compared to identical lead times, staggered lead time scenarios are able to achieve better service levels with the same amount of inventory. However, there is a diminishing benefit as $L'$ continues to increase; indeed, the frontier for $L' = 5$ and $L' = 6$ are almost indistinguishable.

These results show that it can indeed be cost-effective to lengthen the lead times of some fulfillment centers to create staggering. However, the right level of staggering depends on the problem, and the cost savings from staggered lead times and the cost increase from longer lead times need to be held in the right balance.
Figure 3.16: Staggered lead time scenarios achieve a better trade-off frontier between stockout and holding costs.

Table 3.6: Cost of various lead time scenarios

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Cost Overall</th>
<th>Breakdown</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Average</td>
<td>Std</td>
</tr>
<tr>
<td>same2 (L' = 2)</td>
<td>75.370</td>
<td>0.159</td>
</tr>
<tr>
<td>stag3 (L' = 3)</td>
<td>75.242</td>
<td>0.161</td>
</tr>
<tr>
<td>stag4 (L' = 4)</td>
<td>75.154</td>
<td>0.172</td>
</tr>
<tr>
<td>stag5 (L' = 5)</td>
<td>75.158</td>
<td>0.153</td>
</tr>
<tr>
<td>stag6 (L' = 6)</td>
<td>75.174</td>
<td>0.165</td>
</tr>
</tbody>
</table>
3.4.6 Correlations between fulfillment centers

**Motivation.** In all our experiments, different trials of random search rarely produce the same final solution, yet all final solutions have similar costs. This reveals that there is a lot of low-cost solutions for the inventory replenishment problem. We now demonstrate that these near-optimal solutions have some common structural properties, and show how the structure could help us characterize near-optimal solutions and identify them more easily.

We should expect the replenishment policies of different fulfillment centers to be correlated, due to the fact that spillovers introduce interdependence between fulfillment centers, which is an important feature of online retail inventory systems. For example, consider the setting where there is a cluster of fulfillment centers such that spillovers frequently take place within the cluster, but rarely with other fulfillment centers outside the cluster. Then, there is an inventory pooling effect for fulfillment centers in the cluster. The cost of a replenishment policy may be sensitive to the total amount of pooled inventory, but there is likely some flexibility as to how to allocate the pooled inventory between the fulfillment centers in the cluster.

Note that such correlations between fulfillment centers cannot be identified by simply considering the geographical location of fulfillment centers. Indeed, the frequency of spillovers depends not only on the distance between fulfillment centers, but also on other factors such as the geographical distribution of customer demand, the shipping cost to customers, fulfillment center lead times, and more.

All of these factors are implicitly considered in the process of producing near-optimal solutions by random search. By treating the final solutions as samples drawn from a distribution, we can study structure of the distribution and identify correlations between fulfillment centers.

**Distance between clusters of fulfillment centers.** We now define a notion of “distance” between two clusters of fulfillment centers, which will be useful later for identifying fulfillment center correlations with clustering.

Let \( t = 1, \ldots, T \) be the index for independent trials of random search with
gradient-based sampling, and let \( \mathbf{b}^t = (b_1^t, \ldots, b_N^t) \) be the base-stock policy parameters of the final solution that is produced. Each \( \mathbf{b}^t \) can be thought of as the realization of an \( N \)-dimensional random variable. The correlation between \( b_n^t \) for different \( n \) corresponds to the correlation between fulfillment centers.

For a cluster of fulfillment centers \( \mathcal{N} \subseteq \{1, \ldots, N\} \), let

\[
B_{\mathcal{N}} = \left( \sum_{n \in \mathcal{N}} b_n^t \right)_{t=1}^T
\]

be the \( T \) samples of policy parameters summed over the fulfillment centers in \( \mathcal{N} \). Loosely speaking, there is a strong interdependence between fulfillment centers within this cluster, i.e., if spillover occurs frequently between them, then we can expect \( B_{\mathcal{N}} \) to exhibit less variability than samples of the individual fulfillment centers \( (b_n^t)_{t=1}^T, n \in \mathcal{N} \).

For two clusters \( \mathcal{N}_1, \mathcal{N}_2 \subseteq \{1, \ldots, N\} \), we define their distance as

\[
\text{DISTANCE}(\mathcal{N}_1, \mathcal{N}_2) = \text{Covariance} \left( \frac{B_{\mathcal{N}_1}}{\|B_{\mathcal{N}_1}\|}, \frac{B_{\mathcal{N}_2}}{\|B_{\mathcal{N}_2}\|} \right), \quad \mathcal{N}_1, \mathcal{N}_2 \subseteq \{1, \ldots, N\}, \quad (3.26)
\]

where \( \| \cdot \| \) indicates the 2-norm of a vector. Note that under this definition, the distance between two clusters can be negative. A negative distance indicates that the respective inventory placed in the two clusters are negatively correlated, i.e., one is high when the other is low. This negative correlation suggests that there may be inventory pooling between the two clusters, so that in near-optimal solutions, when there is sufficient inventory in one cluster, the other can operate with less inventory.

This is not the only way of defining distances between fulfillment center clusters, but one that we have found to produce meaningful clustering results.

An example for this definition of distance is given below.

**Example.** Consider an instance with \( N = 3 \) fulfillment centers, and suppose we are given \( T = 2 \) samples of near-optimal solutions, \( \mathbf{b}^1 = (b_1^1, b_1^2, b_3^1) = (0, 1, 2) \), \( \mathbf{b}^2 = (b_1^2, b_2^2, b_3^2) = (1, 0, 2) \). Using these two samples, we wish to determine which
pair of fulfillment centers are the most strongly correlated.

For the clusters $\mathcal{N} = \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 2, 3\}$, we have

\[
B_{\{1\}} = (0, 1), \quad B_{\{2\}} = (1, 0), \quad B_{\{3\}} = (2, 2),
\]
\[
B_{\{1,2\}} = (0 + 1, 1 + 0) = (1, 1),
\]
\[
B_{\{1,3\}} = (0 + 2, 1 + 2) = (2, 3),
\]
\[
B_{\{2,3\}} = (1 + 2, 0 + 2) = (3, 2).
\]

Note that $B_{\{1,2\}}$ has a lower variability than either $B_{\{1\}}$ or $B_{\{2\}}$, which indicates that the policy parameters of fulfillment centers 1 and 2 are correlated. On the other hand, the variability of $B_{\{1,3\}}$ does not decrease compared to $B_{\{1\}}$, which indicates that fulfillment centers 1 and 3 are less correlated; the same is true with fulfillment centers 2 and 3.

Similar observations can be made from the distance between clusters $\{1\}$ and $\{2\}$, which is

\[
\text{Distance}(\{1\}, \{2\}) = \text{Covariance}((0, 1), (1, 0)) = -0.5,
\]

while the distances between $\{1\}, \{3\}$ and between $\{2\}, \{3\}$ are both

\[
\text{Distance}(\{1\}, \{3\}) = \text{Covariance}((0, 1), (2, 2)) = 0.
\]
\[
\text{Distance}(\{2\}, \{3\}) = \text{Covariance}((1, 0), (2, 2)) = 0.
\]

In other words, under this notion of distance, fulfillment centers 1 and 2 are “closer” with each other than with fulfillment center 3.

**Agglomerative hierarchical clustering.** To identify fulfillment center correlations, we apply the method of hierarchical clustering, detailed in Algorithm 3.4.1. The method is *agglomerative* in the sense that it begins with each fulfillment center being its own cluster, and iteratively merges pairs of clusters in a greedy manner until only one cluster is left. In each iteration, the pair of clusters with the smallest
custom-defined “distance” is merged, where the distance is defined above.

**Algorithm 3.4.1** Agglomerative hierarchical clustering of fulfillment centers

1. $\mathcal{C} \leftarrow \{\{1\}, \{2\}, \ldots, \{N\}\}$
2. **while** $|\mathcal{C}| > 1$ **do**
   3. $(\mathcal{N}_1, \mathcal{N}_2) \leftarrow \arg\min\limits_{\mathcal{N}_1, \mathcal{N}_2 \subset \{1, \ldots, N\}: \mathcal{N}_1 \neq \mathcal{N}_2} \text{DISTANCE}(\mathcal{N}_1, \mathcal{N}_2)$ \(\Rightarrow\) DISTANCE defined in (3.26)
   4. $\mathcal{C} \leftarrow (\mathcal{C}\backslash\{\mathcal{N}_1, \mathcal{N}_2\}) \cup (\mathcal{N}_1^* \cup \mathcal{N}_2^*)$ \(\Rightarrow\) Replace two clusters with a merged cluster
5. **end while**

**Results.** Figure 3.17 shows the result of hierarchical clustering in the form of a dendrogram. The iterations of Algorithm 3.4.1 can be recreated by reading the dendrogram from the bottom to the top. Each leaf of the dendrogram is a fulfillment center, and the branches represent clusters of fulfillment centers identified in the hierarchical clustering process. Each horizontal line that joins two branches are plotted at the distance ($y$-axis) between the two clusters found in Line 3 of Algorithm 3.4.1.

We remark again that while highly correlated fulfillment centers are often located close to each other physically, the reverse may not be true. For instance, consider fulfillment centers indexed 6, 7, 8, 9, 10 and 12. On the map, 8 and 9 appear to be of similar distance to the two pairs of 6, 7 and 10, 12. However, because of the location of customer regions (see Figure 3.8 above), the set of fulfillment centers $\{6, 7, 8, 9\}$ have a higher correlation with each other and are clustered together before they are with 10, 12.

**Implications.** The result of hierarchical clustering is useful in at least two ways, for optimization and for management. From an optimization perspective, it provides a basis for hierarchical decision making, through the aggregation of fulfillment cen-

---

13 For example, in this case, the first iteration of Algorithm 3.4.1 identifies fulfillment centers 14 and 15 as the pair with the minimum distance of $-0.0015$, and merges the two; in the second iteration, fulfillment centers 2 and 3 are identified with the minimum distance $-0.0010$ and thus merged; in the third iteration, fulfillment center 13 is merged with the cluster $\{14, 15\}$; and so on.
ters. For example, instead of optimizing policy parameters for \( N = 15 \) fulfillment centers all at once, we can run Algorithm 3.4.1 until only 5 clusters are left, and optimize policy parameters for the 5 clusters of aggregated fulfillment centers. Then, using the aggregate solution, optimize the detailed fulfillment center policies for each cluster separately. This type of hierarchical decision is a way to ensure that problems encountered at every level of the hierarchy is tractable.

On a related note, from a managerial perspective, hierarchical clustering also provides insights on appropriate organizational structures for fulfillment centers, especially in light of new types of facilities that interface with fulfillment centers. For example, some online retailers may wish to build cross docks that collect replenishment orders from vendors and distribute to several nearby fulfillment centers, which allows for ordering in bulk to reduce replenishment fixed costs. Other retailers may wish to add outbound service centers that consolidates items from multiple fulfillment centers, combining them into fewer packages to ship to customers. These types of facilities appear to be a growing trend for online retailers as their networks continue to expand. In such cases, hierarchical clustering results can inform the location of these new facilities, and which fulfillment centers they should serve.

### 3.4.7 Sensitivity analysis

**Setup.** In this section, we investigate the impact of demand variability, overage cost \( c_o \), and underage cost \( c_u \) on the quality of low-cost solutions found by random search, and find that they each have a nonlinear effect on the overall cost. As before, we take the final solutions of 100 independent trials of random search, evaluate them with a new simulation for 20,000 iterations, and report the average cost.

**Demand variability.** So far, we’ve assumed that the demand follows a Poisson process, which has a variance that is equal to the mean. In this experiment, we test negative binomial demand distributions, and adjust the parameters to create variance-to-demand ratios of 2, 4 and 8. (The average demand is kept at the default 10 units over the review period of 7 days.) Note that the negative binomial demand
Figure 3.17: Correlations between fulfillment centers can be identified by hierarchical clustering, in which the distance measures are computed from the set of near-optimal replenishment policies.

(a) Hierarchical cluster dendrogram for default lead time scenario $L=west$

(b) Locations of fulfillment centers and regions
is not memoryless, and the dynamic programming formulation no longer applies; nevertheless, simulation evaluation is still valid.

Figure 3.18a shows the costs of the different demand scenarios. We see that as the variance increases relative to the mean, the stockout and holding cost both increase, i.e., a worst service level is achieved despite having more inventory; the overall cost also increases. The change in shipping cost is relatively insignificant (note the scale of the $y$-axis).

**Underage cost.** We test the cases where the per-unit underage (stockout) cost is increased from the default value of $c_u = \$25$ to $\$50$ and $\$100$, while everything else is the same as the default setting. The comparison of overall and breakdown costs are shown in Figure 3.18b.

Note that the stockout cost increases at a lower rate than the increase in $c_u$, which means that as the stockout cost increases, the stockout probability of near-optimal policies decreases, i.e., the service level increases. This is in line with our intuition.

**Overage cost.** Similar to the above We test the cases where the per-unit overage (holding) cost is increased from the default value of $c_o = 0.25$ to $0.5$ and $1.0$, while everything else is the same as the default setting. The comparison of overall and breakdown costs are shown in Figure 3.18c.

Again, similar to the above, and in accordance with intuition, the holding cost decreases at a lower rate than the increase in $c_o$, which indicates that as holding cost increases, less inventory is held in near-optimal policies.

### 3.5 Conclusions

#### 3.5.1 Summary of contributions

Our contributions can be organized according to the aspects of modeling, methods, and managerial implications.
Figure 3.18: Demand variability, overage, and underage costs have a nonlinear effect on the overall cost.

(a) Higher demand variability leads to greater stockout and holding costs.

(b) Comparison of underage (stockout) cost settings

(c) Comparison of overage (holding) cost settings
For modeling, the online-retail inventory replenishment problem is a discrete optimization problem that can be formulated as a dynamic program. The cost function, which includes the three components of shipping, stockout and holding, is difficult to characterize due to demand spillovers, nonlinear shipping costs, and non-identical fulfillment center lead times. Despite the complexity, we provide a precise characterization of the cost function, leveraging the assumption of memoryless demand to decompose each period by the lead times and to derive the expected shipping cost. Empirical analysis on small numerical examples suggest that the dynamic programming methods suffers from the curse of dimensionality, and that the optimal policy has little structure besides being a decreasing function of the inventory position.

For solution methods, we propose a scalable solution scheme by introducing two approximation methods: policy parameterization and simulation. Parameterized policies simplify the solution space; simulation bypasses the complexity of the cost function. We extend the idea of perturbation analysis to our simulation so as to estimate gradient-like information that is useful for optimization. We then devise three simulation optimization methods for finding low-cost parameterized policies, and test them on a numerical example of realistic size and structure. In particular, we accelerate a state-of-the-art random search method with gradient-based sampling, and show that it is able to find low-cost solutions within a few minutes.

For managerial implications, our empirical analysis on a realistic numerical example show that, first, base-stock policies achieve similar costs as more complex classes of policies, but the latter may be employed if there is a need to mitigate fluctuation in order quantities. Secondly, it may be beneficial to strategically lengthen the lead times of some fulfillment centers to create staggering. Thirdly, hierarchical clustering of fulfillment centers helps identify correlations between fulfillment centers, which could inform network design decisions. Last but not least, demand variability and per-unit overage or underage cost parameters have a nonlinear effect on the overall cost.
3.5.2 Future directions

Directions for future work can also be organized by modeling, methods, and managerial implications.

Regarding the model, an interesting extension would be to account for stochasticity in lead times, which often arises in reality. In this case, stochasticity becomes another cause for lead time staggering, and its benefits could outweigh the cost of increased uncertainty. Also, it would also be useful to obtain tighter lower bounds that utilize the stockout and holding cost parameters.

Regarding the approximate solution methods, we have not fully explored parameterized policies for which the order quantities depend on the entire vector of inventory positions rather than just on the corresponding fulfillment center. Such policies could explicitly capture the correlation between fulfillment centers, and may lead to lower costs.

We also believe that the techniques we developed for obtaining gradient-like information and using it to accelerate random search are readily applicable to other problems. Our next step is to test the effectiveness of these techniques on other classic problems in simulation optimization.

Regarding managerial implications, an important finding is that there are many low-cost solutions for the replenishment problem. One future direction may be to determine which solution to use, in light of other considerations or performance measures. For example, referring back to the inventory placement problem in the previous chapter, we may wish to choose low-cost replenishment policies jointly for items in the same bundle, so as to satisfy the capacity allocated to the bundle by the placement problem.

Finally, our empirical analysis results show that it may be worthwhile further exploring the effect of staggered lead times, fulfillment center correlations, and the interaction between the two. Relevant questions include, but are not limited to: What is the optimal configuration of lead times? How do fulfillment center correlations inform the inventory network design? How does the level of lead time staggering
impact the correlations between fulfillment centers? These topics are not only of academic interest, but are also important to the operations of online-retail inventory systems in practice.
Appendix A

Supplementary materials for inventory placement empirical analysis

A.1 Detailed results for MIP experiments

This section supplements Section 2.2.3 with detailed results for each problem instance and each MIP formulations. The main purpose is to show that all problem instances exhibit similar qualitative behavior in performance.
Figure A.1: Objective and bound over time for different Cuts settings ($|I| = 10$, for every combination of problem instances (labeled 0-9) and formulations)
Figure A.2: MIP gap over time for different Cuts settings ($|\mathcal{I}| = 10$, for every combination of problem instances (labeled 0-9) and formulations.
Figure A.3: Objective and bound over time for different formulations

\(|\mathcal{I}| = 10\), for 10 instances labeled 0-9
Figure A.4: MIP gap over time for different formulations
\(|\mathcal{I}| = 10\), for 10 instances labeled 0-9

\(\text{Computation time (seconds), log scale}\)
Figure A.5: MIP performance on medium-sized problem instances

(a) Objective and bound
(b) MIP gap

100 items, 10 instances

<table>
<thead>
<tr>
<th>Instance</th>
<th>Formulation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>in_cap/flow</td>
</tr>
<tr>
<td></td>
<td>in_cap/fac</td>
</tr>
<tr>
<td></td>
<td>in_strong/flow</td>
</tr>
<tr>
<td></td>
<td>in_strong/fac</td>
</tr>
<tr>
<td></td>
<td>sos_sum/flow</td>
</tr>
</tbody>
</table>

Computation time (seconds)
Figure A.6: MIP performance on large problem instances

(a) Objective and bound
(b) MIP gap
Figure A.7: MIP gap at various time intervals for different levels of capacity constraint tightness
\(|I| = 10, \text{ for 10 instances labeled 0-9}\)
A.2 Detailed results for large-scale solution scheme experiments

This section supplements Section 3.4 with detailed results for each problem instance and each configuration of the large-scale solution scheme. The main purpose is to provide further information on the variability in performance between different problem instances.

Tables A.1 and A.2 report the optimality gap and total computation time, respectively, after Stages C, D and E of the large-scale solutions scheme, for each problem instance. Table A.3 report the number of Stage C iterations required to solve the relaxed column-based formulation to optimality. At the bottom of each table, we summarize the mean, standard deviation (SD) and coefficient of variation (CoV, defined as the standard deviation divided by the mean) of each column. Table (2.6) in the main text is taken from the “Mean” rows of each table below.

Table A.4 lists the total computation time, summed over all iterations for each instance, for the two optimization methods (RMP) and (CG) in Stage C. This information reveals that most of the computation effort in Stage C is spent on the column generation subproblems (CG). The unit computation time in Table 2.7 is computed by dividing the “Sum” rows by the total number of iterations obtained from Table A.3.
Table A.1: Optimality gap after each stage, $|\mathcal{I}| = 1000$

(a) Optimality gap (%) after Stage C

| Instance | Configuration | $|\mathcal{I}| = 1$ | $|\mathcal{I}| = 10$ | $|\mathcal{I}| = 10$ | $|\mathcal{I}| = 100$ | $|\mathcal{I}| = 100$ | $|\mathcal{I}| = 1000$ |
|----------|---------------|---------------------|-------------------|-------------------|-------------------|-------------------|-------------------|
|          |               | kmeans              | random            | kmeans            | random            | kmeans            | random            |
| 0        |               | 2.04                | 1.16              | 2                 | 0.82              | 1.8               | 0.75              |
| 1        |               | 2.02                | 1.12              | 1.98              | 0.82              | 1.76              | 0.75              |
| 2        |               | 2.12                | 1.16              | 2.07              | 0.83              | 1.83              | 0.76              |
| 3        |               | 2.01                | 1.08              | 1.94              | 0.75              | 1.74              | 0.69              |
| 4        |               | 2.08                | 1.12              | 2.03              | 0.79              | 1.81              | 0.73              |
| 5        |               | 2.04                | 1.13              | 1.95              | 0.76              | 1.75              | 0.7              |
| 6        |               | 2.04                | 1.17              | 1.98              | 0.84              | 1.82              | 0.78              |
| 7        |               | 2.06                | 1.16              | 2.01              | 0.82              | 1.78              | 0.76              |
| 8        |               | 1.98                | 1.07              | 1.94              | 0.79              | 1.74              | 0.72              |
| 9        |               | 2.04                | 1.15              | 1.99              | 0.82              | 1.81              | 0.76              |
| Mean     |               | 2.04                | 1.13              | 1.99              | 0.8               | 1.78              | 0.74              |
| SD       |               | 0.04                | 0.03              | 0.04              | 0.03              | 0.03              | 0.03              |
| CoV      |               | 1.88%               | 3.08%             | 2.07%             | 3.76%             | 1.93%             | 3.93%             |

(b) Optimality gap (%) after Stage D

| Instance | Configuration | $|\mathcal{I}| = 1$ | $|\mathcal{I}| = 10$ | $|\mathcal{I}| = 10$ | $|\mathcal{I}| = 100$ | $|\mathcal{I}| = 100$ | $|\mathcal{I}| = 1000$ |
|----------|---------------|---------------------|-------------------|-------------------|-------------------|-------------------|-------------------|
|          |               | kmeans              | random            | kmeans            | random            | kmeans            | random            |
| 0        |               | 15.28               | 1.18              | 1.97              | 0.88              | 1.86              | 0.8               |
| 1        |               | 15.05               | 1.15              | 2.01              | 0.87              | 1.83              | 0.8               |
| 2        |               | 1.93                | 1.18              | 2.09              | 0.89              | 1.9               | 0.81              |
| 3        |               | 14.98               | 1.12              | 1.97              | 0.82              | 1.81              | 0.73              |
| 4        |               | 1.89                | 1.11              | 2.04              | 0.85              | 1.88              | 0.78              |
| 5        |               | 15.34               | 1.15              | 1.96              | 0.82              | 1.82              | 0.75              |
| 6        |               | 14.98               | 1.17              | 1.99              | 0.89              | 1.89              | 0.82              |
| 7        |               | 1.88                | 1.19              | 2.01              | 0.88              | 1.86              | 0.81              |
| 8        |               | 14.96               | 1.09              | 1.94              | 0.84              | 1.81              | 0.77              |
| 9        |               | 14.89               | 1.18              | 1.99              | 0.88              | 1.88              | 0.81              |
| Mean     |               | 11.12               | 1.15              | 2                 | 0.86              | 1.85              | 0.79              |
| SD       |               | 6.36                | 0.03              | 0.04              | 0.03              | 0.03              | 0.03              |
| CoV      |               | 57.23%              | 3%                | 2.18%             | 3.18%             | 1.84%             | 3.77%             |
(c) Optimality gap (%) after Stage E

| Instance | Configuration | $|\mathcal{I}| = 1$ | $|\mathcal{I}| = 10$ | $|\mathcal{I}| = 10$ | $|\mathcal{I}| = 100$ | $|\mathcal{I}| = 100$ | $|\mathcal{I}| = 1000$ |
|----------|---------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
|          |               | kmeans          | random          | kmeans          | random          | kmeans          | random          |
| 0        | 10.23         | 1.15            | 1.88            | 0.86            | 1.78            | 0.79            |
| 1        | 10.22         | 1.11            | 1.9             | 0.86            | 1.73            | 0.79            |
| 2        | 1.84          | 1.14            | 1.99            | 0.87            | 1.81            | 0.8             |
| 3        | 10.26         | 1.08            | 1.88            | 0.8             | 1.73            | 0.72            |
| 4        | 1.8           | 1.08            | 1.94            | 0.83            | 1.79            | 0.77            |
| 5        | 10.37         | 1.11            | 1.86            | 0.8             | 1.73            | 0.73            |
| 6        | 10.13         | 1.14            | 1.91            | 0.88            | 1.81            | 0.82            |
| 7        | 1.78          | 1.15            | 1.91            | 0.86            | 1.75            | 0.79            |
| 8        | 10.15         | 1.05            | 1.84            | 0.83            | 1.72            | 0.76            |
| 9        | 10.06         | 1.14            | 1.89            | 0.86            | 1.8             | 0.79            |
| Mean     | 7.68          | 1.12            | 1.9             | 0.84            | 1.77            | 0.78            |
| SD       | 4.06          | 0.04            | 0.04            | 0.03            | 0.04            | 0.03            |
| CoV      | 52.79%        | 3.14%           | 2.22%           | 3.36%           | 2.07%           | 4.04%           |

Table A.2: Total computation time for each stage, $|\mathcal{I}| = 1000$

(a) Total computation time (seconds) for Stage C

| Instance | Configuration | $|\mathcal{I}| = 1$ | $|\mathcal{I}| = 10$ | $|\mathcal{I}| = 10$ | $|\mathcal{I}| = 100$ | $|\mathcal{I}| = 100$ | $|\mathcal{I}| = 1000$ |
|----------|---------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
|          |               | kmeans          | random          | kmeans          | random          | kmeans          | random          |
| 0        | 1331.61       | 706.7           | 5207.44         | 1659.25         | 6914.59         | 15979.89        |
| 1        | 1724.34       | 493.66          | 4880.13         | 1507.17         | 7901.96         | 13910.52        |
| 2        | 1519.8        | 454.15          | 3893.91         | 1566.21         | 5741.81         | 13339.9         |
| 3        | 1558.35       | 819.33          | 3662.44         | 1697.4          | 6358.34         | 13326.21        |
| 4        | 1395.52       | 639.27          | 3611.37         | 1476.19         | 6034.71         | 12058.39        |
| 5        | 1367.11       | 594.06          | 3108.45         | 1269.46         | 6078.94         | 11536.32        |
| 6        | 1183.75       | 900.92          | 4524.2          | 1630.33         | 6503.76         | 12865.17        |
| 7        | 1346.87       | 261.23          | 3464.81         | 1621.56         | 6686.23         | 13991.73        |
| 8        | 1326.98       | 609.1           | 3697.24         | 1449.98         | 7434.75         | 12892.31        |
| 9        | 1188.87       | 883.13          | 2874.68         | 1332.44         | 6939.43         | 16266.68        |
| Mean     | 1394.32       | 636.15          | 3892.47         | 1521            | 6659.45         | 13616.71        |
| SD       | 166.52        | 201.7           | 753.31          | 141.58          | 664.45          | 1521.73         |
| CoV      | 11.94%        | 31.71%          | 19.33%          | 9.31%           | 9.98%           | 11.18%          |
(b) Total computation time (seconds) for Stage E

| Instance | Configuration | $|J| = 1$ | $|J| = 10$ | $|J| = 10$ | $|J| = 100$ | $|J| = 100$ | $|J| = 1000$ |
|----------|---------------|-----------|-------------|-------------|-------------|-------------|-------------|
|          |               | kmeans    | random      | kmeans      | random      | kmeans      | random      |
| 0        |               | 3484.55   | 361.39      | 427.83      | 202.8       | 416.23      | 174.99      |
| 1        |               | 5605.43   | 350.62      | 416.28      | 204.75      | 374.91      | 169.8       |
| 2        |               | 669.8     | 361.11      | 439.83      | 198.28      | 384.66      | 174.99      |
| 3        |               | 3280.64   | 288.34      | 375.88      | 190.07      | 411.03      | 171.62      |
| 4        |               | 620.86    | 294.42      | 426.79      | 193.79      | 357.09      | 202.26      |
| 5        |               | 3032.05   | 308.89      | 433.46      | 211.71      | 430.36      | 184.91      |
| 6        |               | 3448.15   | 334.96      | 398.72      | 227.29      | 370.22      | 176.78      |
| 7        |               | 675.04    | 329.13      | 439.22      | 241.72      | 371.9       | 179.21      |
| 8        |               | 3035.36   | 289.2       | 415.43      | 194.3       | 396.21      | 195.37      |
| 9        |               | 4537.77   | 290.93      | 424.66      | 191.63      | 394.58      | 183.88      |
| Mean     |               | 380.08    | 2412.1      | 2966.58     | 3339.63     | 3099.63     | 18.8        |
| SD       |               | 19.61     | 329.33      | 121.22      | 681.82      | 551.78      | 7.33        |
| CoV      |               | 5.16%     | 13.65%      | 4.09%       | 20.42%      | 17.8%       | 39%         |

(c) Total computation time (seconds) for Stage E

| Instance | Configuration | $|J| = 1$ | $|J| = 10$ | $|J| = 10$ | $|J| = 100$ | $|J| = 100$ | $|J| = 1000$ |
|----------|---------------|-----------|-------------|-------------|-------------|-------------|-------------|
|          |               | kmeans    | random      | kmeans      | random      | kmeans      | random      |
| 0        |               | 3484.55   | 361.39      | 427.83      | 202.8       | 416.23      | 174.99      |
| 1        |               | 5605.43   | 350.62      | 416.28      | 204.75      | 374.91      | 169.8       |
| 2        |               | 669.8     | 361.11      | 439.83      | 198.28      | 384.66      | 174.99      |
| 3        |               | 3280.64   | 288.34      | 375.88      | 190.07      | 411.03      | 171.62      |
| 4        |               | 620.86    | 294.42      | 426.79      | 193.79      | 357.09      | 202.26      |
| 5        |               | 3032.05   | 308.89      | 433.46      | 211.71      | 430.36      | 184.91      |
| 6        |               | 3448.15   | 334.96      | 398.72      | 227.29      | 370.22      | 176.78      |
| 7        |               | 675.04    | 329.13      | 439.22      | 241.72      | 371.9       | 179.21      |
| 8        |               | 3035.36   | 289.2       | 415.43      | 194.3       | 396.21      | 195.37      |
| 9        |               | 4537.77   | 290.93      | 424.66      | 191.63      | 394.58      | 183.88      |
| Mean     |               | 2838.97   | 320.9       | 419.81      | 205.63      | 390.72      | 181.38      |
| SD       |               | 1696.6    | 30.21       | 19.74       | 16.92       | 23.27       | 10.47       |
| CoV      |               | 59.76%    | 9.41%       | 4.7%        | 8.23%       | 5.95%       | 5.77%       |
Table A.3: Number of iterations in Stage C, $|I| = 1000$

| Instance | Configuration | $|I| = 1$ kmeans | $|I| = 10$ random | $|I| = 100$ kmeans | $|I| = 100$ random | $|I| = 1000$ |
|----------|---------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| 0        |               | 583             | 161             | 259             | 76              | 123             | 70              |
| 1        |               | 635             | 154             | 271             | 75              | 133             | 69              |
| 2        |               | 596             | 147             | 275             | 74              | 120             | 67              |
| 3        |               | 601             | 163             | 250             | 73              | 122             | 63              |
| 4        |               | 595             | 172             | 256             | 76              | 123             | 66              |
| 5        |               | 613             | 163             | 247             | 75              | 131             | 66              |
| 6        |               | 572             | 175             | 265             | 79              | 132             | 73              |
| 7        |               | 582             | 141             | 248             | 78              | 130             | 68              |
| 8        |               | 602             | 151             | 268             | 80              | 132             | 66              |
| 9        |               | 598             | 166             | 251             | 78              | 124             | 70              |
| Mean     |               | 597.7           | 159.3           | 259             | 76.4            | 127             | 67.8            |
| SD       |               | 17.6            | 10.88           | 10.2            | 2.27            | 5.01            | 2.82            |
| CoV      |               | 2.94%           | 6.83%           | 3.94%           | 2.97%           | 3.95%           | 4.16%           |

Table A.4: Total computation time breakdown for Stage C, $|I| = 1000$

(a) Total computation time (seconds) spent on (RMP)

| Instance | Configuration | $|I| = 1$ kmeans | $|I| = 10$ random | $|I| = 100$ kmeans | $|I| = 100$ random | $|I| = 1000$ |
|----------|---------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| 0        |               | 1.19            | 1.29            | 3.67            | 5.85            | 15.3            | 123.94          |
| 1        |               | 1.32            | 1.21            | 3.84            | 5.79            | 16.19           | 111.24          |
| 2        |               | 1.25            | 1.13            | 3.93            | 5.41            | 14.12           | 122.88          |
| 3        |               | 1.33            | 1.42            | 3.39            | 5.87            | 16.66           | 103.91          |
| 4        |               | 1.28            | 1.31            | 3.34            | 5.65            | 15.2            | 122.54          |
| 5        |               | 1.19            | 1.33            | 3.12            | 5.24            | 16.86           | 100.91          |
| 6        |               | 1.02            | 1.41            | 3.67            | 6.47            | 17.1            | 121.42          |
| 7        |               | 1.24            | 1.2             | 3.42            | 5.98            | 17.18           | 114.47          |
| 8        |               | 1.2             | 1.16            | 3.6             | 5.68            | 16.33           | 103.46          |
| 9        |               | 1.13            | 1.17            | 3.3             | 5.51            | 15.07           | 134.19          |
| Sum      |               | 12.15           | 12.63           | 35.28           | 57.45           | 160.02          | 1158.94         |
(b) Total computation time (seconds) spent on \((\mathbf{CG}_j)\)

| Instance | Configuration | \(|J| = 1\) kmeans | \(|J| = 10\) random | \(|J| = 10\) kmeans | \(|J| = 10\) random | \(|J| = 100\) kmeans | \(|J| = 100\) random | \(|J| = 1000\) |
|----------|---------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| 0        |               | 1330.42 | 705.41 | 5203.77 | 1653.4 | 6899.29 | 15855.94 |
| 1        |               | 1723.01 | 492.45 | 4876.29 | 1501.37 | 7885.76 | 13799.27 |
| 2        |               | 1518.55 | 453.02 | 3889.98 | 1560.8 | 5727.68 | 13217.01 |
| 3        |               | 1557.03 | 817.91 | 3659.05 | 1691.53 | 6341.67 | 13222.29 |
| 4        |               | 1394.24 | 637.96 | 3608.03 | 1470.53 | 6019.51 | 11935.84 |
| 5        |               | 1365.91 | 592.74 | 3105.33 | 1264.22 | 6062.08 | 11435.39 |
| 6        |               | 1182.73 | 899.51 | 4520.53 | 1623.86 | 6486.66 | 12743.73 |
| 7        |               | 1345.63 | 260.02 | 3461.39 | 1615.57 | 6669.05 | 13877.25 |
| 8        |               | 1325.78 | 607.93 | 3693.64 | 1444.29 | 7418.41 | 12788.84 |
| 9        |               | 1187.74 | 881.96 | 2871.37 | 1326.92 | 6924.36 | 16132.47 |
| Sum      |               | 13931.03 | 6348.92 | 38889.37 | 15152.49 | 66434.47 | 135008.04 |
Appendix B

Supplementary materials for inventory replenishment empirical analysis

B.1 R-BEESE with and without gradient-based sampling

This appendix supplements Section 3.3.3 by providing empirical evidence that gradient-like information improves the efficiency of the R-BEESE random search method. We perform R-BEESE, with and without gradient-based sampling, on the default setting of the realistic numerical example presented in Section 3.4.1. Figure B.1 shows the simulated cost over time for R-BEESE, averaged over 100 independent trials. The method finds low-cost solutions more quickly with gradient-based sampling.
Figure B.1: Gradient-based sampling improves the efficiency of R-BEESE.
B.2 Detailed empirical results for the target local service level heuristic

This appendix supplements Section 3.4.3 with further results for the target local service level heuristic.

Figure B.2a shows the cost function of $\alpha$ policies, with $\alpha$ values on the $x$-axis. The cost is a piecewise-constant function of $\alpha$, where each piecewise-constant interval corresponds to the same policy. The lowest range of $\alpha$ correspond to the policy of all zeros, in which the stockout cost completely dominates the cost function. As $\alpha$ increases, the stockout cost decreases, while the holding cost increases. The shipping cost, however, does not exhibit a monotonic trend. It first increases with $\alpha$, due to an increase in sales, and then decreases, as there is more inventory in the system and thus customers are more likely to be served by closer fulfillment centers.

The optimal target service level is marked with an asterisk. The cost function appears to be quite flat around the minimum, indicating that there is a collection of near-optimal solutions. Note also that the optimal value of $\alpha$ is not very high. Indeed, because inventory is fully fungible across fulfillment centers, a high actual system service level can be achieved with relatively lower target local service levels.

Figure B.2b plots the number of policies over time that are still being considered by the $KN+$ selection-of-the-best procedure. The figure shows that a lot of solutions are eliminated quickly in the beginning, leaving only a few close competitors that take a lot more computational efforts to differentiate. Thus, if we just want to obtain a near-optimal solution, it is possible to terminate the process early and just arbitrarily pick one of the competitive candidates.
Figure B.2: Detailed empirical results for the target local service level heuristic

(a) Costs of $\alpha$ base-stock policies

(b) Number of $\alpha$ policies still in competition over time
Bibliography


Cornuéjols, Gérard, Ranjani Sridharan, Jean-Michel Thizy. 1991. A comparison of
heuristics and relaxations for the capacitated plant location problem. *European
journal of operational research* 50(3) 280–297.

Costa, Alysson M. 2005. A survey on benders decomposition applied to fixed-charge


relaxation methods for multicommodity capacitated fixed charge network design. *Discrete Applied Mathematics* 112(1) 73–99.


