# Case-Pack Allocation in Retail Distribution Networks

Myungeun Eom
Department of Industrial and Systems Engineering
Auburn University

Alejandro Toriello
Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

August 19, 2026

**Abstract**

Many retail distribution networks use a hierarchical structure in which regional distribution centers replenish local distribution centers that are closer to customers. In such networks, products are often shipped in large quantities, as case packs, cases or pallets, to the regional distribution center, whereas local distribution centers may require quantities at the individual-unit level. Case breaking, which disaggregates case packs into individual units at the regional distribution center, enables more flexible allocation across local distribution centers. However, limited processing capacity may prevent all cases from being broken upstream. Planners must therefore decide which cases to break and how to allocate both individual units and intact cases. Despite its practical importance, there is scant literature on explicitly incorporating case-breaking capacity into inventory allocation models. Motivated by this gap, we study an operational case-pack allocation problem from a regional to local distribution centers with case-breaking capacity constraints. Given available stock at a regional distribution center, we optimize a concave surrogate objective function that captures product urgency, current inventory status, and expected demand at local distribution centers. We formulate the problem as an integer program and propose a column generation algorithm with efficient pricing heuristics. We develop an efficient iterative greedy algorithm and a dynamic programming-based heuristic that gradually considers larger action spaces over iterations. Computational experiments on industrial-scale instances show that the method finds numerically optimal solutions, with an average relative optimality gap under $10^{-4}$ in roughly 100 minutes, outperforming a commercial solver in both solution quality and runtime. We further analyze how the allocation plan changes under different capacity constraints and individual-unit handling requirements.

**Keywords:** case-pack allocation, e-commerce supply chains, column generation

## 1 Introduction

The growth of e-commerce has introduced high-volume and time-sensitive demand, making supply chain optimization essential. To serve nationwide customers, e-commerce companies build hierarchical logistics networks with multiple types of facilities (Wang and Minner 2025). For example, JD.com, one of the largest e-commerce companies in the world, manages a two-level network with regional and front distribution centers (Shen et al. 2025); regional distribution centers are large warehouses, while front distribution centers are located closer to customers for fast delivery, but have small capacities and get frequent replenishment from regional facilities. To enable rapid delivery, e-commerce companies proactively position inventory at

downstream facilities, often before customer orders are realized (Shen et al. 2025, Li et al. 2026). This process typically involves two related decision layers: forecasting, which determines the target inventory level at local facilities considering uncertain customer demand, and allocation, which determines how to allocate products under capacity constraints and estimated targets.

While many aspects of inventory management have been studied extensively, including forecasting (e.g., Carbonneau et al. 2008, Bandara et al. 2019), there is one critical yet underexplored factor: *case-breaking operations*. Products are often shipped in large quantities (as *case-packs*, cases or pallets) and then disaggregated into individual units at intermediate facilities (Gao et al. 2014). Figure 1 illustrates an example of a robot arm picking individual products from a case. Case-breaking operations are necessary, as downstream distribution centers have limited storage space; by allowing small shipment quantities to downstream facilities, case breaking can achieve better inventory spread.

Figure 1: A robot arm picking an individual product from a case (Berkshire Grey 2022)

Prior research has primarily focused on the design of automated case-breaking systems (also referred to as split-case picking or breakpack systems) and on evaluating their operational performance (Johnson and Meller 2002, Ketzenberg et al. 2002, Greer et al. 2020, Berkshire Grey 2022). For example, Walmart Canada, in collaboration with the simulation software MOSIMTEC, conducted a case study to understand the most attractive products to send through a breakpack system and how to schedule breaking operations in its automated system (Greer et al. 2020). While it showed that case-breaking can be beneficial in not only reducing inventory costs but also increasing profits (Ketzenberg et al. 2002), it may be impossible to break all cases upstream because of limited processing capacity. It is therefore important to carefully choose which cases to disaggregate in addition to how to distribute both individual units and intact cases.

With this motivation, we study an operational case-pack allocation problem with case-breaking capacity

constraints. The problem is formulated over a single period (i.e. decisions made on a daily basis), and intended to be solved repeatedly. Consistent with a short-term operational horizon, we assume available inventory at the regional distribution center and target inventory levels at local distribution centers are given and deterministic. Because of limited inventory and/or capacity constraints, we may not be able to meet the targets. Therefore, we optimize a surrogate objective, a concave *placement benefit function* for each product at each local distribution center. This function captures a product's urgency, current inventory status, and expected demand at each local distribution center.

We summarize our main contributions as follows:

- We introduce the case-pack allocation problem and formulate it as an integer program (IP).
- We propose a column generation (CG) algorithm to solve the problem.
- We enhance the CG algorithm with efficient pricing heuristics, including an efficient greedy iterative algorithm and a heuristic based on dynamic programming that gradually considers larger action spaces.

Our computational experiments show that our method can find near-optimal solutions with an average relative optimality gap under $10^{-4}$ in 100 minutes for large-scale instances with up to 200,000 products, outperforming a commercial IP solver in both solution quality and running time.

The remainder of the paper is organized as follows. In Section 2, we discuss relevant literature. Section 3 describes the case-pack allocation problem, and we present our solution approach in Section 4. Section 5 discusses the results of our computational study. Finally, in Section 6, we conclude the paper and discuss potential future work.

## 2 Literature Review

Our model is related to inventory allocation in e-commerce supply chains and more generally to two-echelon inventory models; we review the most relevant papers in this section. Our solution approach uses a pattern-based formulation solved by column generation. Because column generation is widely used across many optimization domains, we focus our review on pattern-based formulations for transportation and distribution problems.

The inventory allocation problem addresses how to allocate limited inventory across multiple locations to best meet demand or service targets. Several studies solve large-scale inventory allocation and reallocation problems using demand estimates as inputs. Brandimarte et al. (2024) study an inventory reallocation problem, considering both shipment from a warehouse to stores and reallocation among stores. Similarly, Naderi et al. (2020) study a transshipment problem to balance inventory across a large fashion retail network. While these papers demonstrate the importance and computational challenges of large-scale inventory allocation in retail networks, they do not consider the case-pack allocation and breaking structure, which motivates our problem.

Case-breaking and individual-unit picking has been considered through warehouse layout design. Bartholdi and Hackman (2011) study the problem of selecting products to be stored in a forward area, where individual units can be picked up more efficiently than from bulk storage area, to minimize total labor costs subject to the space limit. While such warehouse layout design can accelerate the order-picking process, maintaining dedicated forward locations for individual products may be less suitable for e-commerce distribution centers handling a large number of products with high turnover. In our model, we take the warehouse configuration and case-breaking capacity as given and study how to allocate inventory under the given capacity.

Inventory allocation has also been studied jointly with demand forecasting, demand learning, or downstream fulfillment decisions in e-commerce and multi-echelon fulfillment networks. Shen et al. (2025) propose an end-to-end algorithm that integrates demand forecasting and daily inventory allocation to minimize lost sales and transfer costs. Nambiar et al. (2021) develop an algorithm for a multi-period inventory allocation problem for seasonal goods, where demand forecasts evolve over time and allocation decisions are made to minimize lost sales. Feng et al. (2024) study a distributionally robust allocation model to improve the fulfillment rates of front distribution centers under demand uncertainty. Recently, Epstein and Ma (2025) study how to split inventory across multiple warehouses by optimizing surrogate functions that approximate downstream fulfillment decisions under uncertain customer demand. While incorporating demand uncertainty and downstream fulfillment decisions may improve supply chain efficiency, the case-pack breaking operation in our model significantly complicates the inventory allocation problem due to its combinatorial nature. Therefore, we focus on the deterministic allocation problem faced after demand estimates or downstream inventory targets have been generated by a separate forecasting or planning system.

Although our model focuses on a single-period operational allocation decision, it is related to the broader literature on two-echelon inventory systems with a single warehouse, multiple retailers, and multiple prod-

ucts. Early studies established fundamental modeling frameworks for such systems (e.g., Clark and Scarf 1960, Muckstadt and Roundy 1987, Graves 1996), and subsequent extensions have incorporated practical factors, such as stochastic lead times (e.g., Simchi-Levi and Zhao 2005); readers may refer to de Kok et al. (2018) for a review. These models typically focus on determining base-stock levels and replenishment policies over a planning horizon and therefore ignore operational constraints such as case-pack handling or transfer capacity limits.

Our solution approach is based on a pattern-based formulation solved by column generation. This approach follows the classical idea of representing combinatorial decisions by feasible patterns, as in cutting-stock formulations (Gilmore and Gomory 1961), and generating only promising patterns dynamically. In our model, a pattern represents a feasible case-pack allocation for a product from a regional distribution center to multiple local distribution centers, satisfying supply and transfer capacity limits. Closely related pattern-based formulations appear in fixed-charge transportation problems. Roberti et al. (2015) define a pattern as a feasible flow from a supply node to demand nodes satisfying supply and capacity limits and derive valid inequalities to strengthen the formulation. Mingozzi and Roberti (2018) extend this idea by defining patterns for both supply and demand nodes and solving a pattern-matching problem. A related formulation is also used by Zhao et al. (2018), who combine flow-pattern variables with a Lagrangian decomposition framework.

Our formulation differs from these studies in several ways. Fixed-charge transportation models consider a single commodity, while our problem involves multiple products with different case sizes and demand estimates. In addition, our model considers a single source, so the source-sink pattern-matching structure is not directly applicable. Finally, the pricing problem in our column-generation algorithm involves case-pack allocation decisions with a separable concave objective, rather than a linear cost. As a result, the pricing subproblem becomes a nonlinear discrete optimization problem, requiring a different solution strategy from those used in existing flow-pattern formulations.

## 3 Problem Description

We model an echelon of an e-commerce supply chain with a single origin, a set of destinations $J$, and a set of products $K$. At the origin, $N_k \in \mathbb{N}$ cases of product $k \in K$ are available, each with $Q_k \in \mathbb{N}$ units per case. We can either transfer a case intact to a single destination or break it into individual units and distribute

them across multiple destinations. The origin has a case-breaking capacity $C^S \in \mathbb{N}$, measured in time units; breaking a case for product $k \in K$ consumes $s_k > 0$ time units, and the total case-breaking time cannot exceed $C^S$. Each destination $j \in J$ has a concave placement benefit function $f_{jk}$ with integer breakpoints and $f_{jk}(0) = 0$, for each product $k \in K$. Finally, the total volume of products transferred to destination $j$ cannot exceed a lane capacity $C^L_j \in \mathbb{N}$; the lane capacity represents transportation capacity available for that destination. We denote by $v_k > 0$ the volume per unit for product $k \in K$.

### 3.1 Mathematical Formulation

We formulate the problem in (1) with variables $x_{jk}$ and $y_{jk}$ respectively denoting the number of individual units and intact cases of product $k$ transferred to destination $j$; variable $z_k$ represents the number of broken cases of product $k$:

$$\max_{x,y \in \mathbb{Z}_+^{J \times K},\ z \in \mathbb{Z}_+^K} \sum_{j,k} f_{jk}(x_{jk} + Q_k y_{jk})$$

$$\text{s.t.} \sum_k s_k z_k \le C^S \tag{1a}$$

$$\sum_k v_k (x_{jk} + Q_k y_{jk}) \le C^L_j \qquad \forall j \in J \tag{1b}$$

$$\sum_j y_{jk} + z_k \le N_k \qquad \forall k \in K \tag{1c}$$

$$\sum_j x_{jk} \le Q_k z_k \qquad \forall k \in K. \tag{1d}$$

The objective function maximizes the total placement benefits. Note that we can linearize it using a piecewise linear approximation of function $f$ with breakpoints only at certain integer values. Constraints (1a), (1b), and (1c) enforce the case-breaking, lane capacity, and supply constraints, respectively. Constraints (1d) ensure that we can transfer individual units only up to the units in broken cases.

### 3.2 Problem Complexity

We present the NP-hardness of the problem in Proposition 1.

**Proposition 1.** *The problem* (1) *is strongly NP-hard. When either* $|J| = 1$ *or* $|K| = 1$*, the problem is weakly NP-hard.*

Informally, the problem inherits the multiple knapsack problem structure even without case-breaking

components. In addition, the case-breaking constraints lead to a weak linear programming (LP) relaxation, which makes commercial IP solvers struggle. Even with a single product or a single destination, there is a polynomial-time reduction from the partition problem, which is weakly NP-hard. We provide the formal proof in Appendix A.1.

# 4 Solution Approach

To solve this strongly NP-hard problem, we adopt a CG approach by defining patterns for each product, similarly to Roberti et al. (2015) and Mingozzi and Roberti (2018). We denote by $L_k$ the set of patterns for product $k$. Pattern $\ell \in L_k$ represents a distribution of inventory for product $k$ across destinations $J$, satisfying the lane capacity and supply limits. In what follows, we formally present the master and pricing problems and the main algorithm.

## 4.1 Master Problem

We formulate a master problem and its linear relaxation in LP (2), where $p_\ell, b_\ell, d_{j\ell}$, and $c_{j\ell}$ respectively denote the total benefit, number of broken cases, number of individual units transferred to $j$, and number of intact cases transferred to $j$ for product $k$ in pattern $\ell \in L_k$. Decision variable $w_\ell$ represents the fraction of pattern $\ell$ selected among the patterns in $L_k$ for product $k$.

$$\max_{w \geq 0} \sum_k \sum_{\ell \in L_k} p_\ell w_\ell$$

$$\text{s.t.} \sum_{\ell \in L_k} w_\ell = 1 \qquad \forall k \in K \tag{2a}$$

$$\sum_k \sum_{\ell \in L_k} v_k (d_{j\ell} + Q_k c_{j\ell}) w_\ell \leq C_j^L \qquad \forall j \in J \tag{2b}$$

$$\sum_k \sum_{\ell \in L_k} s_k b_\ell w_\ell \leq C^S. \tag{2c}$$

The objective function maximizes the total placement benefits. Constraints (2a) ensure that exactly one pattern is selected for each product. Constraints (2b) and (2c) enforce the lane and sort capacity constraints, respectively. In Proposition 2, we show that formulation (2) is a relaxation of (1) and provides a tighter dual bound, with the proof deferred to Appendix A.2. We evaluate the tightness of LP (2) in Section 5.

**Proposition 2.** *The optimal objective value of* (2) *is an upper bound for* (1)*, and its value is less than or*

*equal to that of the linear relaxation of* (1).

### 4.2 Pricing Problem

Let $\lambda_k$, $\alpha_j$, and $\beta$ denote the dual values for constraints (2a), (2b) and (2c), respectively. Given a dual solution, we formulate the pricing problem for product $k$ as follows:

$$\max_{x,y\in\mathbb{Z}_+^J,\ z\in\mathbb{Z}_+} \Bigg\{ \sum_j \Big( f_{jk}(x_{jk}+Q_k y_{jk}) - \alpha_j v_k (x_{jk}+Q_k y_{jk}) \Big) - \beta s_k z_k : \\ \sum_j y_{jk} + z_k \le N_k,\ \sum_j x_{jk} \le Q_k z_k,\ x_{jk}+Q_k y_{jk} \le C_j^L\ \forall j \in J \Bigg\}. \tag{3}$$

If the optimal objective value is greater than $\lambda_k$, then we add the corresponding pattern to the master problem. Let $g_{jk}(u) := f_{jk}(u) - \alpha_j v_k u$. By the concavity of $g$ and the nonnegativity of $\beta$, IP (3) is equivalent to

$$\max_{x,y\in\mathbb{Z}_+^J} \Bigg\{ \sum_j g_{jk}(x_{jk}+Q_k y_{jk}) - \beta s_k \left\lceil \frac{\sum_j x_{jk}}{Q_k} \right\rceil : \\ \sum_j y_{jk} + \left\lceil \frac{\sum_j x_{jk}}{Q_k} \right\rceil \le N_k,\ x_{jk} \le Q_k - 1\ \forall j \in J,\ x_{jk}+Q_k y_{jk} \le T_{jk}^*\ \forall j \in J \Bigg\}, \tag{4}$$

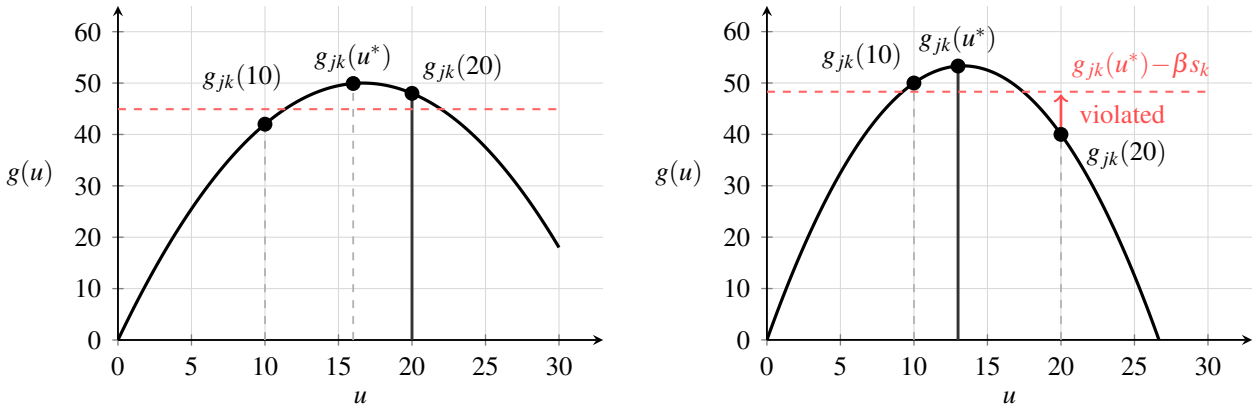
where the adjusted target $T_{jk}^*$ is defined with $u_{jk}^*$, the smallest maximizer of $g_{jk}$. Specifically,

$$T_{jk}^* := \begin{cases} Q_k \lceil u_{jk}^*/Q_k \rceil & \text{if } Q_k \lceil u_{jk}^*/Q_k \rceil \le C_j^L,\ g_{jk}(Q_k \lceil u_{jk}^*/Q_k \rceil) > g_{jk}(u_{jk}^*) - \beta s_k \\ & \quad \text{and } g_{jk}(Q_k \lceil u_{jk}^*/Q_k \rceil) > g_{jk}(Q_k \lceil u_{jk}^*/Q_k \rceil - Q_k) \\ \min\{u_{jk}^*, C_j^L\} & \text{else} \end{cases}.$$

Note that $g_{jk}$ and $T_{jk}^*$ depend on $\alpha_j$. Since $\alpha_j$ is a constant within each iteration, we omit this dependence to simplify the notation. We formally state this equivalence in Lemma 3, with the proof deferred to Appendix A.3.

**Lemma 3.** *IP* (4) *is equivalent to IP* (3).

We note the feasible region of IP (4) is substantially smaller than that of IP (3). This reduction is achieved mainly through the adjusted target, which eliminates transfer quantities that cannot be optimal based on $g_{jk}$. Figure 2 illustrates examples of adjusted targets under different shapes of $g_{jk}$. In Figure 2(a), rounding $u^*$ up to the nearest multiple of $Q_k$ loses little value and satisfies the lane capacity constraint; therefore, the target is set to be the rounded-up value. In Figure 2(b), rounding up causes the value of $g_{jk}$ to fall below the threshold $g_{jk}(u^*) - \beta s_k$, indicated by the red dashed line. Therefore, we exclude transfer quantities greater than $u^*$, i.e. $T_{jk}^* = u^*$.




(a) Case 1: $T^*_{jk} = Q_k\lceil u^*/Q_k\rceil = 20$ ($u^* = 16$)

(b) Case 2: $T^*_{jk} = \min\{u^*, C^L_j\} = 13$ ($u^* = 13$)

Figure 2: Illustration of adjusted targets $T^*_{jk}$ with different $g_{jk}$ when $Q_k = 10, C^L_j = 30$, and $\beta s_k = 5$.

#### 4.2.1 Dynamic Programming Formulation

We can solve the pricing problem by a dynamic programming (DP) algorithm in $\mathcal{O}(|J|Q_k^2N_k^2)$ time. Let $V_j(q^x, q^y)$ be the maximum value achievable from destinations $j$ to $|J|$, given that a total of $q^x$ individual units and $q^y$ intact cases have been transferred up to destinations $j-1$. The terminal condition is $V_{|J|+1}(q^x, q^y) = -\beta s_k\lceil\frac{q^x}{Q_k}\rceil$ for all $q^x, q^y \geq 0$. The optimal objective value is $v_1(0,0)$. For destination $j$, the Bellman recursion is

$$V_j(q^x, q^y) = \max_{(x,y)\in A_j(q^x,q^y)} \{V_{j+1}(q^x + x, q^y + y) + g_{jk}(x + Q_k y)\} \quad \forall (q^x, q^y), \tag{5}$$

where $A_j(q^x, q^y) = \{(x,y) : x \leq Q_k - 1, y \leq N_k - q^y - \lceil\frac{q^x+x}{Q_k}\rceil, x + Q_k y \leq T^*_{jk}, x, y \in \mathbb{Z}_+\}$. Note that the action space has size $\min\{T^*_{jk}, Q_kN_k\}$; since the sort and lane capacities are typically tight, $T^*_{jk}$ remains small in practice, and thus the actual action space is substantially smaller than its worst-case bound. We further reduce the state space using the fact that transferring $Q_k$ individual units is equivalent to transferring one intact case with an appropriate adjustment to the objective value. We formally state this property in Lemma 4, deferring the proof to Appendix A.4.

**Lemma 4.** *For any destination $j$ and state $(q^x, q^y)$, $V_j(q^x + Q_k, q^y) = V_j(q^x, q^y + 1) - \beta s_k$, and the optimal actions at states $(q^x + Q_k, q^y)$ and $(q^x, q^y + 1)$ are the same.*

By Lemma 4, we can restrict attention to states with $q^x \leq Q_k - 1$, which reduces the state space to $\mathcal{O}(Q_kN_k)$, and rewrite the Bellman equation (5) as

$$V_j(q^x, q^y) = \max_{(x,y)\in A_j(q^x,q^y)} \{V_{j+1}((q^x + x) \bmod Q_k, q^y + y + \lfloor(q^x + x)/Q_k\rfloor)$$

$$+ g_{jk}(x + Q_k y) - \beta s_k \lfloor (q^x + x)/Q_k \rfloor \} \quad \forall (q^x, q^y). \tag{6}$$

#### 4.2.2 Dynamic Programming Algorithm

We describe the DP algorithm with the reduced state space in Algorithm 1. While we state the Bellman recursion (6) in backward form, we solve the pricing problem via an equivalent forward DP, considering only reachable states and using pruning strategies. In Algorithm 1, $S_j$ represents the set of states that are reachable from the initial state $(0,0)$ through a sequence of feasible actions over the first $j-1$ destinations, and $r_j(q^x, q^y)$ denotes the maximum achievable value from the first $j$ destinations by transferring a total of $q^x$ individual units and $q^y$ intact cases. To reduce the number of reachable states, we prune states that cannot lead to an optimal solution in line 8. For *LB*, we use the objective value of a known feasible solution if one is available and use zero otherwise, since transferring no units yields a value of zero. In our main algorithm, we use the best solution found in the previous pricing iteration to compute *LB*; more details are in Section 4.3. Lemma 5 formally states that the pruning step does not eliminate any optimal solution; see Appendix A.5 for the proof.

**Algorithm 1** DP algorithm for product $k$

1: **Input**: Destinations $J$, Parameters $N_k, Q_k, \{T^*_{jk}\}_{j \in J}$, Dual value $\lambda_k, \beta$, Functions $\{g_{jk}\}_{j \in J}$, Lower bound *LB*
2: $S_1 \leftarrow \{(0,0)\}$, $S_j \leftarrow \emptyset \; \forall j \in \{2, \ldots, |J|\}$, $r_1(0,0) \leftarrow 0$
3: **for** $j = 1, \ldots, |J|$ **do**
4:  **for** each $(q^x, q^y) \in S_j$ **do**
5:  **for** each action $(x,y) \in A_j(q^x, q^y)$ **do**
6:  $(\tilde{q}^x, \tilde{q}^y) \leftarrow ((q^x + x) \bmod Q_k,\ q^y + y)$
7:  $\tilde{r} \leftarrow r_j(q^x, q^y) + g_{jk}(x + Q_k y) - \beta s_k \lfloor (q^x + x)/Q_k \rfloor$
8:  **if** $\tilde{r} < 0$ **or** $\tilde{r} + \sum_{j'=j+1}^{|J|} g_{j'k}(u^*_{j'k}) - \beta s_k \lceil \tilde{q}^x / Q_k \rceil < LB$ **then**
9:  **continue**
10:  **if** $(\tilde{q}^x, \tilde{q}^y) \notin S_{j+1}$ **or** $\tilde{r} > r_{j+1}(\tilde{q}^x, \tilde{q}^y)$ **then**
11:  $S_{j+1} \leftarrow S_{j+1} \cup \{(\tilde{q}^x, \tilde{q}^y)\}$
12:  $r_{j+1}(\tilde{q}^x, \tilde{q}^y) \leftarrow \tilde{r}$
13: $L^* \leftarrow \emptyset$
14: **for** each $(q^x, q^y) \in S_{|J|+1}$ **do**
15:  **if** $r_{|J|+1}(q^x, q^y) - \beta s_k \lceil q^x / Q_k \rceil > \lambda_k$ **then**
16:  Add backtracked solution to $L^*$
17: **return** $L^*$

**Lemma 5.** *A state $(\tilde{q}^x, \tilde{q}^y)$ with accumulated value $\tilde{r}$ generated at stage $j$ in Algorithm 1 can be pruned without loss of optimality if either $\tilde{r} + \sum_{j'=j+1}^{|J|} g_{j'k}(u^*_{j'k}) - \beta s_k \left\lceil \frac{\tilde{q}^x}{Q_k} \right\rceil < LB$ or $\tilde{r} < 0$.*

### 4.2.3 Iterative Greedy Algorithm

Under certain conditions, we can develop more efficient algorithms. We formalize these conditions as follows.

**Definition 1.** *Consider a pricing problem instance for product $k$ defined with adjusted targets $\{T^*_{jk}\}_j$. We call the instance an easy instance if at most one destination $j$ has $T^*_{jk} \geq Q_k$.*

For easy instances, we develop an iterative greedy algorithm by considering two cases. First, consider the case where $T^*_{jk} < Q_k$ for all $j$. Since we cannot transfer any intact cases, we have $y_{jk} = 0$ for all $j$, and thus IP (4) reduces to

$$\max_{z \in \{0,\ldots,\min\{N_k, \lceil (\sum_j T^*_{jk})/Q_k \rceil\}\}} \left\{ h(z) := \max_{x \in \mathbb{Z}^J_+} \left\{ \sum_j g_{jk}(x_{jk}) - \beta s_k z : \sum_j x_{jk} \leq Q_k z,\ x_{jk} \leq T^*_{jk}\ \forall j \in J \right\} \right\}. \quad (7)$$

Given $z$, we can solve the inner problem optimally by greedily allocating units across destinations subject to the upper bound constraints; the greedy algorithm is optimal because $g$ is concave. In addition, since $h$ is concave (Lemma 6; see Appendix A.6 for the proof), it suffices to iterate over $z$ from 0 upward and stop as soon as $h(z)$ starts to decrease. Note that we never break more than $\lceil (\sum_j T^*_{jk})/Q_k \rceil$ cases, as additional breakage only incurs sortation penalties without providing additional benefit.

**Lemma 6.** *Function $h(z)$ is concave for any nonnegative integer $z$.*

Next, consider the case with exactly one destination $j^*$ with $T^*_{j^*k} \geq Q_k$. We can rewrite IP (4) as

$$\max_{y \in \{0,\ldots,\min\{N_k, \lfloor T^*_{j^*k}/Q_k \rfloor\}\}} \left\{ H(y) := \max_{z \in \{0,\ldots,\min\{N_k - y, \lceil (\sum_j T^*_{jk})/Q_k \rceil\}\}} h'(y,z) \right\},$$

where

$$h'(y,z) := \max_{x \in \mathbf{Z}^J_+} \left\{ \sum_{j \neq j^*} g_{jk}(x_{jk}) + g_{j^*k}(x_{j^*k} + Q_k y) - \beta s_k z : \sum_j x_{jk} \leq Q_k z,\ x_{jk} \leq T^*_{jk} - Q_k y 1_{\{j = j^*\}}\ \forall j \in J \right\}.$$

Given $y$, the inner problem reduces to the case without intact case transfers and can be written as IP (7). Furthermore, since function $H$ is concave in $y$ (Lemma 7; see Appendix A.7 for the proof), it is sufficient to iterate over $y$ from 0 upward and stop as soon as $H(y)$ begins to decrease.

**Lemma 7.** *Function $H(y)$ is concave for any nonnegative integer $y$.*

We summarize the full procedure in Algorithm 2. Since the values $T^*_{jk}$ tend to be small in tightly capacitated environments, the number of iterations over both $y$ and $z$ is also typically small in practice.

**Algorithm 2** Iterative greedy algorithm for easy instances

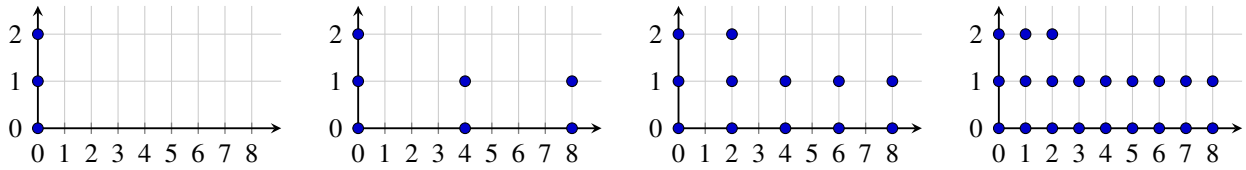
**for** $y = 0, 1, \ldots, \min\{N_k, \lfloor T^*_{j^*k}/Q_k \rfloor\}$ **do**
  **for** $z = 0, 1, \ldots, \min\{N_k - y, \lceil (\sum_j T^*_{jk})/Q_k \rceil\}$ **do**
    Compute $h'(y,z)$ by greedily allocating $Q_k z$ units, with $j^*$ pre-allocated $Q_k y$ units
    Set $H(y) = h'(y,z)$
    **if** $h'(y,z) < h'(y,z-1)$ **then**
      **break**
  **if** $H(y) < H(y-1)$ **then**
    **return** $H(y)$
**return** $H(y)$

### 4.3 Main Algorithm

In this section, we describe our algorithm (Algorithm 3). Similar to conventional CG methods, we iteratively solve the master problem and the pricing problems until no additional pattern with positive reduced cost exists. After convergence, we solve the IP version of (2) using the current set of patterns by enforcing variables $w$ to be binary. Finally, we reoptimize the distribution of individual units while fixing the case-breaking and intact-case transfer decisions; this can be efficiently solved as an LP.

To find new patterns quickly in early iterations, we propose a multiplier($\gamma$)-based DP heuristic. The heuristic defines a parameter $m := \max(1, \lfloor \gamma Q_k \rfloor)$ and restricts the transfer of individual units to multiples of $m$ in the pricing problem for each product $k$. Figure 3 illustrates how the action space in the DP heuristic changes with different values of $\gamma$. Each dot in the subplots represents a feasible action $(x, y)$. We start with parameter $\gamma = 1$, which only allows intact case transfers. As the iterations progress, we gradually reduce $\gamma$ by half, making the action space finer. Specifically, if no new patterns are found or the total reduced cost of newly generated columns is relatively small compared to the current master objective value, then we reduce $\gamma$. If $\gamma$ becomes smaller than a threshold $\theta$, then we switch to an exact algorithm; we solve IP (3) instead of running the exact DP, as this tends to be faster, especially with small $|J|$. Except for the case when $\gamma = 1$, where the DP algorithm can quickly find multiple good patterns, we use the iterative greedy algorithm (Algorithm 2) when the pricing problem is an easy instance.

In column generation, the quality of initial columns is critical for fast convergence. Algorithm 4 summarizes two greedy heuristics for generating initial patterns. The first heuristic greedily allocates units across $(j,k)$ pairs in descending order of marginal benefit *per unit*, subject to the total supply for each product ($Q_k N_k$ for product $k$) and the lane capacity for each destination ($C^L_j$ for each destination $j$). We convert the allocated units into intact cases and individual units and add one pattern per product. Note that the generated



(a) $\gamma = 1$ (b) $\gamma = 1/2$ (c) $\gamma = 1/4$ (d) $\gamma = 1/8$

Figure 3: Action space $\{(x,y) : x + Q_k y \leq T^*_{jk}\}$ with $Q_k = 9$ and $T^*_{jk} = 20$ for different values of parameter $\gamma$.

**Algorithm 3** Main algorithm

**Input**: Pricing parameters $\varepsilon$, $\theta$
DP parameter $\gamma \leftarrow 1$
Generate initial patterns $\{L_k\}_k$ using greedy heuristics (Algorithm 4)
**while true do**
  Obtain master objective value $Obj^{MP}$ and dual values $(\lambda, \alpha, \beta)$ by solving LP (2) with $\{L_k\}_k$
  **for** each product $k$ **in parallel do**
    **if** pricing problem is an easy instance **and** $\gamma < 1$ **then**
      Run the iterative greedy algorithm (Algorithm 2) and add a pattern to $L_k$ if the reduced cost $> 0$
    **else if** $\gamma \geq \theta$ **then**
      Run the DP algorithm (Algorithm 1) restricting the transfer of individual units to multiples of $\max(1, \lfloor \gamma Q_k \rfloor)$ and add generated patterns to $L_k$
    **else**
      Solve IP (3) and add a pattern to $L_k$ if the reduced cost $> 0$
  **if** $\gamma < \theta$ **and** no new patterns are found **then**
    **break**
  **else if** no new patterns are found **or** total reduced cost of newly generated patterns $\leq \varepsilon Obj^{MP}$ **then**
    $\gamma \leftarrow \gamma/2$
$(x,y,z) \leftarrow$ Solve the IP version of (2) by enforcing variables $w$ to be binary
$(x',y,z) \leftarrow$ Solve model (1) for variables $x$ after fixing variables $y$ and $z$
**return** $(x',y,z)$

patterns do not necessarily satisfy the sort capacity constraint. The second heuristic first greedily assigns intact cases in descending order of marginal benefit *per case*, while considering the total supply and lane capacity limits. It then greedily allocates units in descending order of marginal benefit per unit, subject to the residual supply and lane capacity until the sort capacity is about to be violated. As we explicitly consider the sort capacity, the set of generated patterns is feasible for IP (2). Last, we add a zero pattern representing no transfer for each product.

**Algorithm 4** Initial pattern generation

$L_k \leftarrow \emptyset$ for each product $k \in K$
**(Greedy per unit)** Greedily allocate units across $(j,k)$ pairs in descending order of marginal benefit increase per unit, subject to the supply limit $Q_k N_k$ and lane capacity $C_j^L$
Convert the allocated units into intact cases and individual units
Add one pattern to $L_k$ for each product $k \in K$
**(Two-step greedy)** Greedily assign intact cases across $(j,k)$ pairs in descending order of marginal benefit increase per case, subject to the supply limit $N_k$ and lane capacity $C_j^L$
Greedily allocate units across $(j,k)$ pairs in descending order of marginal benefit increase per unit subject to the residual supply and capacities
Add one pattern to $L_k$ for each product $k \in K$
**(No transfer)** Add a zero pattern (no transfer) to $L_k$ for each product $k \in K$
**return** $\{L_k\}_k$

# 5 Computational Study

We demonstrate the effectiveness of our algorithm through a computational study on random instances. We describe how we generate instances in Section 5.1 and present the results in Section 5.2. In Section 5.3, we analyze the impact of key parameters on the solution of our algorithm by considering modified problem settings. We performed all experiments on a high-performance computing cluster with 32 CPUs (AMD EPYC-Rome processors) and 80 GB of RAM, using Python for implementation and Gurobi 10.01 for solving mathematical models. We used a four-hour time limit and multi-threading for running Gurobi; when running the CG method, we used parallelization for solving pricing problems.

## 5.1 Instances

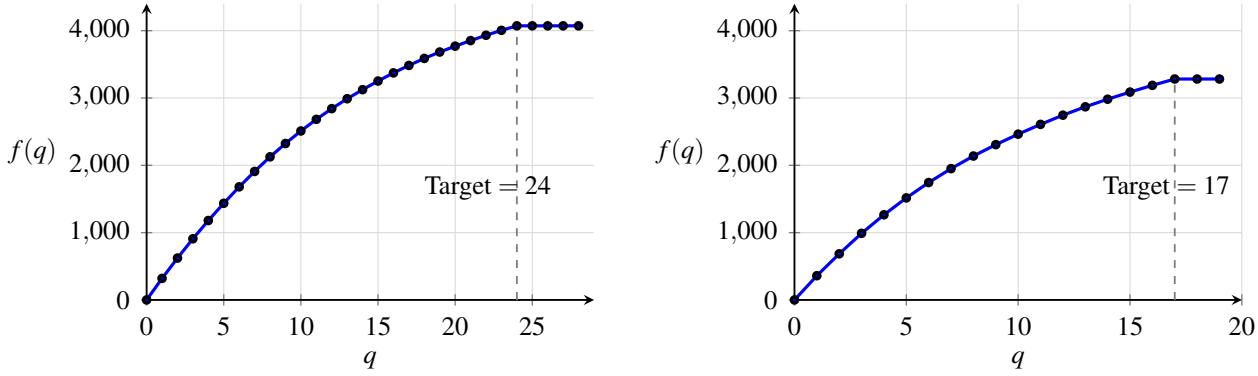
To evaluate the performance of our algorithm across a range of problem scales, we generate instances with 10 destinations and varying numbers of products $|K| \in \{5, 10, 30, 50, 70, 100, 200\} \times 10^3$. For each value of $|K|$, we construct 24 instances corresponding to all combinations of the following parameter settings:

- case size $Q_k$ for each $k$: sampled uniformly from either [2, 50] or [2, 200]
- supply $N_k$ for each $k$: sampled uniformly from either [1, 20] or [1, 50]
- supply-to-demand ratio for each $k$: 0.5, 1, or 2
- lane capacity-to-min(supply, demand) ratio for each $j$: sampled uniformly from either [1%, 5%] or [5%, 10%]

- unit product volume and unit case-breaking speed for all products

Specifically, we randomly choose the case size and supply for each product following the given distributions. We next determine the total target demand quantity corresponding to the given supply-to-demand ratio. Ratios of 0.5, 1, and 2 represent undersupply, balanced, and oversupply conditions, respectively. We then randomly distribute the total target quantity across destinations. The target quantity of each product at each destination represents the quantity at which the placement benefit function is maximized.

We next generate the placement benefit functions. For each product, we randomly sample a base value from $[200, 1200]$. For each destination, we add a random perturbation from $[-50, 50]$ to this base value to set the initial slope of the benefit function. We also randomly sample the number of breakpoints from $[20, 50]$. At each breakpoint, we decrease the slope by multiplying a random factor in $(0.90, 0.95)$ and then rounding up; after the target quantity, we set the slope to zero. Figure 4 depicts examples of the placement benefit functions for a product at different destinations. While we conduct experiments on nondecreasing concave functions, our solution approach is applicable to any concave functions.




(a) Destination 1 (initial slope = 320)

(b) Destination 2 (initial slope = 360)

Figure 4: Examples of the piecewise concave benefit functions for a product.

After generating all product data, we set the lane capacity for each destination according to the given distribution. The distribution of [1%, 5%] represents a tight-capacity condition, where each destination's lane capacity is set to a small fraction of its potential inbound flow. This scenario captures common practical settings in which lane capacities are the most binding constraints. The distribution of [5%, 10%] represents a moderate-capacity condition, where a larger fraction of demand can be served. Finally, case-breaking capacity $C^S$ is drawn uniformly from [1%, 10%] of the maximum number of breakable units.

## 5.2 Benchmarking

We compare the performance of our proposed method (Algorithm 3) and a commercial MIP solver (Gurobi). For the proposed method, we use parameters $\varepsilon = 0.1$ and $\theta = 0.5$, and a time limit of 10 minutes for the reoptimization step. We run Gurobi with a time limit of four hours. Table 1 reports the average bound quality of Gurobi and the proposed method over the 24 instances for each $|K|$. *Root LP, Gurobi bound*, and *CG bound* represent the ratios of the LP relaxation of IP (1), the bound returned by Gurobi within the four-hour time limit, and the converged CG objective, respectively, each normalized by the best bound achieved for each instance. As shown, the optimal objective value of LP (2) is on average 10.47% tighter than that of the LP relaxation of IP (1). This indicates the effectiveness of the pattern-based formulation. In addition, Gurobi struggles to improve the bound as $|K|$ increases. For the largest instance, the bound returned by Gurobi after the four-hour time limit is 5.65% worse than the optimal objective value of LP (2).

| $\|K\|$ | Root LP (%) | Gurobi bound (%) | CG bound (%) |
|---|---|---|---|
| 5,000 | 89.50 | 99.75 | 100.00 |
| 10,000 | 89.81 | 99.59 | 100.00 |
| 30,000 | 89.58 | 96.81 | 100.00 |
| 50,000 | 89.49 | 95.75 | 100.00 |
| 70,000 | 89.44 | 95.55 | 100.00 |
| 100,000 | 89.51 | 95.13 | 100.00 |
| 200,000 | 89.38 | 94.35 | 100.00 |
| Average | 89.53 | 96.71 | 100.00 |

Table 1: Average bound quality of Gurobi and the proposed method over 24 random instances for each $|K|$.

Table 2 reports the average solution quality of Gurobi and the proposed method over the 24 instances for each $|K|$. *Gurobi solution* and *Proposed method* report the average optimality gaps of the final solutions relative to the best bound for each instance. As shown, our method achieves an optimality gap within the range [0.002%, 0.004%] across all $|K|$, whereas the average optimality gap of Gurobi increases from 0.06% to 30.29% as $|K|$ increases.

Table 3 summarizes the average computational time breakdown of the proposed method for each $|K|$. We report all times in seconds. *Total* reports the average runtime of the proposed method, while the other columns report the average runtime and average ratio relative to the total runtime for each step. *Initialization*, *Master*, *Pricing*, *Final IP*, and *Reopt.* correspond to the time for generating initial patterns, solving all master problems, solving all pricing problems, solving the IP version of (2) with generated patterns, and reoptimizing only the individual unit transfers, respectively. We do not report the runtime of Gurobi as it

| $\|K\|$ | Gurobi solution (%) | Proposed method (%) |
|---|---|---|
| 5,000 | 0.06 | 0.004 |
| 10,000 | 0.45 | 0.003 |
| 30,000 | 8.93 | 0.003 |
| 50,000 | 15.39 | 0.004 |
| 70,000 | 15.10 | 0.004 |
| 100,000 | 16.98 | 0.003 |
| 200,000 | 30.29 | 0.002 |

Table 2: Average optimality gaps of the solutions obtained by Gurobi and the proposed method over 24 random instances for each $|K|$.

| $\|K\|$ | Total | Initialization | Master | Pricing | Final IP | Reopt. |
|---|---|---|---|---|---|---|
| 5,000 | 31.1 | 3.3 (10.6%) | 1.8 (5.8%) | 20.8 (66.9%) | 1.1 (3.7%) | 3.9 (12.4%) |
| 10,000 | 68.5 | 7.5 (11.0%) | 6.8 (10.0%) | 43.3 (63.3%) | 2.5 (3.6%) | 8.1 (11.7%) |
| 30,000 | 247.1 | 22.9 (9.3%) | 59.1 (23.9%) | 127.0 (51.5%) | 10.2 (4.1%) | 26.7 (10.8%) |
| 50,000 | 499.9 | 37.4 (7.5%) | 181.4 (36.3%) | 211.0 (42.3%) | 20.3 (4.0%) | 47.8 (9.6%) |
| 70,000 | 843.5 | 53.3 (6.4%) | 386.5 (45.6%) | 294.2 (35.2%) | 38.9 (4.4%) | 68.0 (8.1%) |
| 100,000 | 1562.2 | 76.2 (4.9%) | 871.2 (55.6%) | 432.5 (27.9%) | 69.1 (4.3%) | 109.2 (7.0%) |
| 200,000 | 6034.6 | 158.4 (2.6%) | 4449.6 (73.6%) | 861.4 (14.5%) | 284.6 (4.5%) | 272.8 (4.6%) |

Table 3: Average computational time breakdown for each $|K|$.

reached the four-hour time limit for all instances. As shown, our proposed method required only one hour and 40 minutes to obtain near-optimal solutions for the largest instance, while Gurobi reached the time limit for all instances. It is also notable that for the smallest instance, the proposed method found near-optimal solutions in less than a minute, while Gurobi again reached the time limit. Another important observation is that the master problem becomes the main bottleneck as $|K|$ increases. This is because patterns are defined per product; with large $|K|$, we add a large number of patterns at once to the master problem, and thus it can take a long time to solve it.

To further understand the convergence of the column generation procedure, we summarize the average algorithm statistics in Table 4. *Iterations* and *Total columns* report the average number of iterations until convergence and the average total number of columns added, respectively. *Easy instance ratio* reports the average ratio of easy instances to the total number of products. We can see that more than 70% of pricing problem instances are easy instances across all values of $|K|$, which enables us to solve pricing problems quickly. It is also notable that the column generation converges in about seven iterations, indicating that a relatively small number of pricing rounds is sufficient to generate a high-quality column set for this problem. This suggests that column generation is well suited for our problem.

| $\|K\|$ | Iterations | Total columns | Easy instance ratio |
|---|---|---|---|
| 5,000 | 6.8 | 10,571 | 71.99% |
| 10,000 | 7.1 | 20,331 | 72.92% |
| 30,000 | 6.9 | 59,216 | 72.12% |
| 50,000 | 6.9 | 95,862 | 72.62% |
| 70,000 | 6.8 | 139,088 | 72.09% |
| 100,000 | 7.1 | 184,688 | 72.89% |
| 200,000 | 6.8 | 401,240 | 71.90% |

Table 4: Average algorithm statistics for each $|K|$.

We also explored LP rounding as an alternative solution approach, as such methods are commonly used in combinatorial optimization (e.g., Lim et al. 2006); they solve a continuous relaxation and iteratively round fractional values to obtain an integer solution. Since the benefit functions have integer breakpoints and all parameters are integral, there exists an optimal solution to the linear relaxation of IP (1) with integral transfer amounts; i.e. $x_{jk} + Q_k y_{jk} \in \mathbb{Z}_+ \forall j \in J, k \in K$. Given such a solution, we can construct integer case and unit transfers as follows: $\bar{x}_{jk} = (x_{jk} + Q_k y_{jk}) \mod Q_k$, $\bar{y}_{jk} = \left\lfloor \frac{x_{jk}+Q_k y_{jk}}{Q_k} \right\rfloor$, and $\bar{z}_k = \left\lceil \frac{\sum_j \bar{x}_{jk}}{Q_k} \right\rceil$. The resulting solution may violate the case-breaking capacity; we can greedily reduce the number of broken cases, either by reducing the individual unit transfer or increasing the transfer amount to the next multiple of the case size, until the case-breaking capacity is satisfied. Although a more sophisticated rounding scheme may perform well under certain conditions, for example when the case-breaking capacity is sufficiently large, this heuristic produced optimality gaps exceeding 10% even on our smallest instances (when $|K| = 5000$). We therefore focus on the proposed CG approach, which performed significantly better in our experiments.

## 5.3 Modified Problem Settings

We analyze the impact of key parameters on the solution of our CG algorithm by considering modified problem settings.

### 5.3.1 Sensitivity to Capacity Parameters

To understand how capacity parameters affect the solution structure, we conduct experiments under four capacity configurations. For each instance described in Section 5.1, we scale either the lane capacity or sort capacity by a factor of 0.5 or 1.5.

Figure 5 shows the percentage change from the baseline across four metrics, intact-case fraction, product coverage, average destinations per product, and destination assortment share, for each of the four capacity

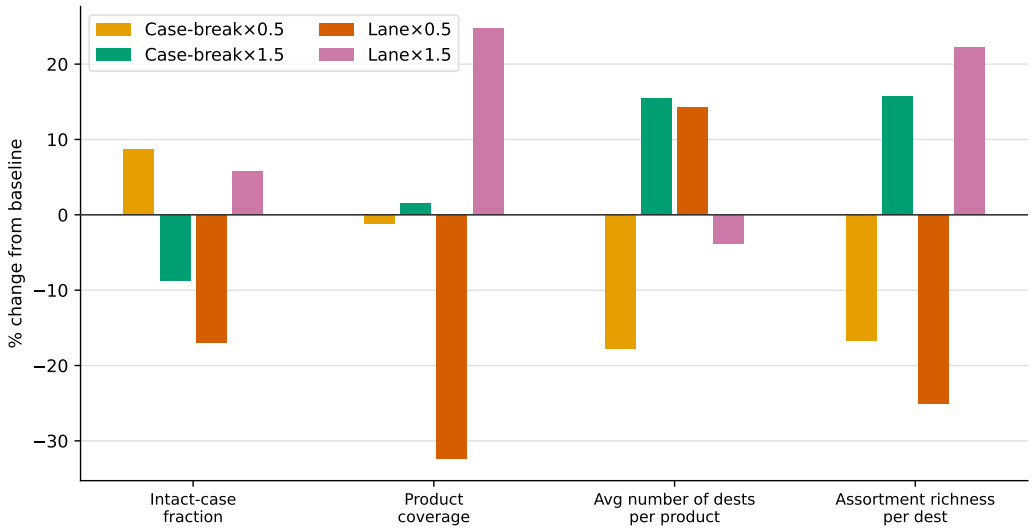



Figure 5: Percentage change from baseline across four metrics for each capacity configuration.

configurations. *Intact-case fraction* is the proportion of shipped units that are transferred as intact cases. *Product coverage* is the percentage of products with positive transfer units. *Avg number of dests per product* measures how many destinations on average each product is sent to. Finally, *Assortment richness per dest* represents the average fraction of all products received by each destination. For each metric, the four bars, from left to right, correspond to case-breaking capacity multiplied by 0.5, case-breaking capacity multiplied by 1.5, lane capacity multiplied by 0.5, and lane capacity multiplied by 1.5.

As shown, tightening sort capacity increases the intact-case fraction by about 9%; due to limited sort capacity, the algorithm favors transferring intact cases over breaking them. However, this leads to a narrow distribution across destinations. Both the average number of destinations per product and the destination assortment share decrease by around 18-19%. Increasing sort capacity produces the opposite effect.

Lane capacity has a different impact. Tightening lane capacity reduces product coverage by roughly 32% and destination assortment share by 25%. Under limited lane capacities, the algorithm prioritizes the most valuable products and works as a product-selection mechanism. Notably, tighter lane capacity also leads to more case breaking, as the algorithm uses individual units to better utilize constrained lanes. Increasing lane capacity again reverses these effects. In contrast to lane capacity, increasing sort capacity significantly increases the average number of destinations per product substantially (by 15%), while increasing lane capacity leaves it nearly unchanged. This suggests that sort capacity primarily enables broader distribution of individual units across destinations.

We also examine how total excess units, which are the units transferred beyond target quantities, change across different sort capacity configurations. Table 5 reports the percentage change from baseline under different sort capacity configurations. As expected, decreasing sort capacity increases excess units; the algorithm favors transferring intact cases over breaking them and intact cases cannot be precisely adjusted to match target quantities. Conversely, increasing sort capacity reduces excess units by enabling more precise allocation through case breaking.

| $\|K\|$ | Sort$\times$0.5 | Sort$\times$1.5 |
|---|---|---|
| 5,000 | 45.78 | -36.93 |
| 10,000 | 56.01 | -37.09 |
| 30,000 | 47.94 | -30.35 |
| 50,000 | 48.14 | -32.90 |
| 70,000 | 41.95 | -33.28 |
| 100,000 | 49.41 | -34.83 |
| 200,000 | 41.56 | -30.27 |

Table 5: Percentage change from baseline in total excess units.

Overall, these results suggest that sort capacity primarily determines how products are transferred, while lane capacity governs what gets shipped and to where. A local distribution center seeking to increase the variety of products it receives should consider increasing the lane capacity rather than sort capacity. In contrast, if reducing excess inventory at local distribution centers is the primary objective, increasing sort capacity is likely to be more effective.

### 5.3.2 Eliminating Residual Units in Broken Cases

Recall that in our model, not all units in broken cases need to be transferred. Although a regional distribution center typically has more storage space than local distribution centers, leaving too many individual units from broken cases at the regional distribution center can create operational challenges. These residual units may require additional handling, tracking, and storage, and they may complicate subsequent picking and replenishment operations. Therefore, in settings where such residual unit handling is costly or operationally challenging, it may be preferable to require all individual units from broken cases to be transferred.

We examine whether the current model results in a substantial number of residual units. Figure 6 summarizes the average number of non-transferred individual units in broken cases for each $|K|$ and case size configuration; *small* and *large* represent the case-size sampling distributions $[2,50]$ and $[2,200]$, respectively. For each $|K|$, the two bars, from left to right, correspond to the *small* and *large* case-size classes. As shown

in the figure, the number of non-transferred individual units in broken cases increases as $|K|$ increases and the case size becomes larger. Specifically, in the largest instances with large case sizes, around 600 individual units are left at the regional distribution center on average. This suggests that handling residual units can be an issue in some settings.

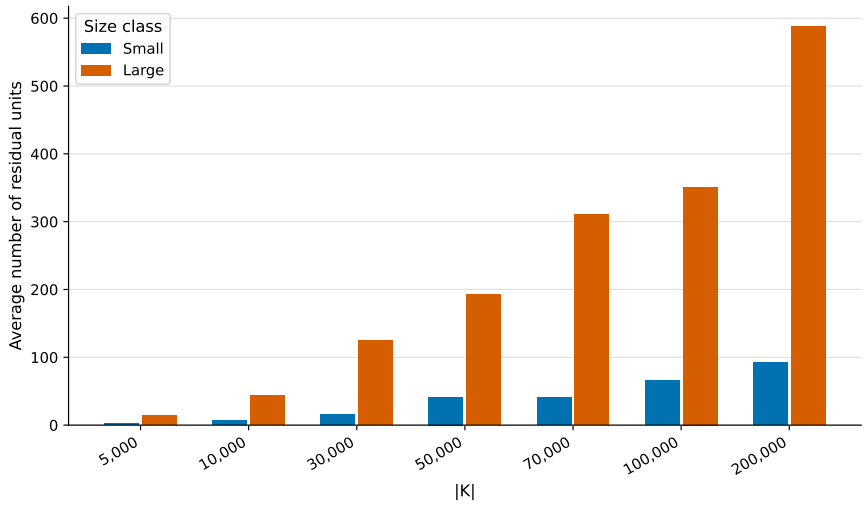



Figure 6: Average number of non-transferred individual units in broken cases.

Motivated by this observation, we consider a variant in which all individual units from broken cases must be transferred. To impose this requirement, we can replace the inequality sign with an equality sign in constraints (1d). In addition, we can apply our solution approach (Algorithm 3) with a slight modification; we need to modify the terminal condition of the DP algorithm to ensure that we transfer all units in broken cases. While the iterative greedy algorithm no longer guarantees optimality for easy instances under the modified problem setting, we can still use the multiplier-based DP heuristic, which helps preserve computational efficiency.

We test our algorithm under this additional requirement and compare the resulting objective values and solution times with those under the original setting. On average, the objective value decreases by 0.008% and the solution time increases by 49%. One possible explanation for the negligible difference in the objective value is the relatively large number of products compared to the tight capacity constraints. If the no-residual requirement makes transferring a product less beneficial, the algorithm can transfer another product instead. Another possible reason is the shape of our placement benefit functions. In our experiments, the benefit functions become flat after target quantities. If the model penalized excess units, we might observe a larger decrease in the objective value. However, as we investigate in the next section, this turns out not to be the

case.

### 5.3.3 Penalty for Excess Units

While our CG approach is applicable to general concave benefit functions, its solution quality and runtime might depend on their shapes. To evaluate the performance of our algorithm on other relevant benefit functions, we model penalties for excess units above target quantities and conduct experiments on the largest instances (when $|K| = 200,000$). Specifically, we set the penalty for the first excess unit by multiplying the initial base value by $-0.5$ and adding a random perturbation. We randomly sample five breakpoints and decrease the slope by multiplying a random factor in $(1.05, 1.15)$ at each breakpoint. Note that the shape of each benefit function up to the target quantity remains the same as in Section 5.1.

As expected, the total number of excess units decreases by 82.5% compared to the case without penalties, while the total transfer quantity remains the same. In addition, the optimality gap remains very small at 0.003% while the solution time increases from 101 to 244 minutes. Although this increase in solution time is substantial, the performance dominance of our CG algorithm over Gurobi is even more significant; Gurobi obtains an average optimality gap of 37.75% within the four-hour time limit. This demonstrates that our CG approach finds numerically optimal solutions in reasonable time when excess units are penalized. Lastly, we compare the solutions with and without the no-residual requirement as discussed in Section 5.3.2; even with penalties for excess units, the difference in the objective value remains negligible.

## 6 Conclusions

In this paper, we studied a case-pack allocation problem that explicitly incorporates case-breaking capacity constraints in e-commerce distribution systems. We proposed a pattern-based formulation and a column generation algorithm with efficient pricing heuristics, including dynamic programming-based methods and an iterative greedy algorithm, to efficiently solve large-scale instances.

Our computational results demonstrate that the proposed approach achieves near-optimal solutions with very small optimality gaps and significantly outperforms a commercial solver in both solution quality and scalability. For the largest instance with 200,000 products and 10 destinations, our algorithm finds near-optimal solutions in 1 hour and 40 minutes, whereas Gurobi returns solutions with an optimality gap of 30% after 4 hours. Through experiments under modified problem settings, we further demonstrate that our CG

algorithm is effective across various benefit functions and capacity conditions, as well as under an additional requirement that all units from broken cases must be transferred.

Our computational experiments primarily focus on settings with tight sort and lane capacities. While such conditions are common in e-commerce supply chain applications, our algorithm may also be effective under more relaxed capacity conditions. For example, in grocery applications with a large number of destinations and frequent replenishment through recurring truck schedules, lane capacities may be less restrictive as replenishment quantities can be distributed across multiple delivery opportunities. In addition, perishable products could have benefit functions with large penalties for excess units. Another interesting direction is to extend the model by integrating downstream fulfillment decisions explicitly and incorporating online decision-making to capture the continuous nature of operations at large warehouses.

# References


K. Bandara, P. Shi, C. Bergmeir, H. Hewamalage, Q. Tran, and B. Seaman. Sales Demand Forecast in E-commerce Using a Long Short-Term Memory Neural Network Methodology. In T. Gedeon, K. W. Wong, and M. Lee, editors, *Neural Information Processing*, pages 462–474, Cham, 2019. Springer International Publishing. ISBN 978-3-030-36718-3.

J. J. Bartholdi and S. T. Hackman. Warehouse & distribution science: release 0.92. *Atlanta, GA, The Supply Chain and Logistics Institute, School of Industrial and Systems Engineering, Georgia Institute of Technology*, 2011.

Berkshire Grey. Break pack warehouse. `https://www.berkshiregrey.com/learn/break-pack-warehouse`, 2022. Accessed: 2025-12-26.

P. Brandimarte, G. Craparotta, and E. Marocco. Inventory reallocation in a fashion retail network: A matheuristic approach. 317(2):603–615, 2024. ISSN 0377-2217. doi: 10.1016/j.ejor.2024.04.016. URL `https://www.sciencedirect.com/science/article/pii/S0377221724002972`.

R. Carbonneau, K. Laframboise, and R. Vahidov. Application of machine learning techniques for supply chain demand forecasting. 184(3):1140–1154, 2008. ISSN 0377-2217. doi: 10.1016/j.ejor.2006.12.004. URL `https://www.sciencedirect.com/science/article/pii/S0377221706012057`.

A. J. Clark and H. Scarf. Optimal Policies for a Multi-Echelon Inventory Problem. *Management Science*, 6(4):475–490, July 1960. ISSN 0025-1909. doi: 10.1287/mnsc.6.4.475. URL `https://pubsonline.informs.org/doi/abs/10.1287/mnsc.6.4.475`. Publisher: INFORMS.

T. de Kok, C. Grob, M. Laumanns, S. Minner, J. Rambau, and K. Schade. A typology and literature review on

stochastic multi-echelon inventory models. 269(3):955–983, 2018. ISSN 0377-2217. doi: 10.1016/j.ejor.2018.02.047. URL https://www.sciencedirect.com/science/article/pii/S0377221718301802.

B. Epstein and W. Ma. Optimizing Inventory Placement for a Downstream Online Matching Problem, May 2025. URL http://arxiv.org/abs/2403.04598. arXiv:2403.04598 [cs].

H. Feng, M. Feng, Q. Wang, Q. Jin, Y. Zhang, L. Cao, and X. Hao. Improving front distribution center fulfillment rates: A distributionally robust approach, 2024. URL https://papers.ssrn.com/abstract=5054555.

L. Gao, D. J. Thomas, and M. B. Freimer. Optimal Inventory Control with Retail Pre-Packs. *Production and Operations Management*, 23(10):1761–1778, Oct. 2014. ISSN 1059-1478. doi: 10.1111/poms.12189. URL https://doi.org/10.1111/poms.12189. Publisher: SAGE Publications.

P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem. *Operations research*, 9(6):849–859, 1961.

S. C. Graves. A Multiechelon Inventory Model with Fixed Replenishment Intervals. *Management Science*, 42(1): 1–18, Jan. 1996. ISSN 0025-1909. doi: 10.1287/mnsc.42.1.1. URL https://pubsonline.informs.org/doi/abs/10.1287/mnsc.42.1.1. Publisher: INFORMS.

A. B. Greer, S. Ponsford, and S. Martin. Simulating an automated breakpack system in a walmart distribution center. In K.-H. Bae, B. Feng, S.-C. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, editors, *Proceedings of the 2020 Winter Simulation Conference*. Institute for Operations Research and the Management Sciences, 2020. URL https://informs-sim.org/wsc20papers/054.pdf.

M. E. Johnson and R. D. Meller. Performance analysis of split-case sorting systems. *Manufacturing & Service Operations Management*, 4(4):258–274, 2002.

M. Ketzenberg, R. Metters, and V. Vargas. Quantifying the benefits of breaking bulk in retail operations. *International Journal of Production Economics*, 80(3):249–263, Dec. 2002. ISSN 0925-5273. doi: 10.1016/S0925-5273(02)00258-X. URL https://www.sciencedirect.com/science/article/pii/S092552730200258X.

X. Li, Y. Zheng, Z. Zhou, and Z. Zheng. Demand prediction, predictive shipping, and product allocation for large-scale e-commerce, 2026. URL https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3277125.

A. Lim, F. Wang, and Z. Xu. A transportation problem with minimum quantity commitment. *Transportation science*, 40(1):117–129, 2006.

A. Mingozzi and R. Roberti. An exact algorithm for the fixed charge transportation problem based on matching source and sink patterns. *Transportation Science*, 52(2):229–238, 2018.

J. A. Muckstadt and R. O. Roundy. Multi-item, one-warehouse, multi-retailer distribution systems. *Management Science*, 33(12):1613–1621, 1987.

S. Naderi, K. Kilic, and A. Dasci. A deterministic model for the transshipment problem of a fast fashion retailer

under capacity constraints. 227:107687, 2020. ISSN 0925-5273. doi: 10.1016/j.ijpe.2020.107687. URL https://www.sciencedirect.com/science/article/pii/S0925527320300797.

M. Nambiar, D. Simchi-Levi, and H. Wang. Dynamic inventory allocation with demand learning for seasonal goods. 30(3):750–765, 2021. ISSN 1937-5956. doi: 10.1111/poms.13315. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/poms.13315. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/poms.13315.

R. Roberti, E. Bartolini, and A. Mingozzi. The fixed charge transportation problem: An exact algorithm based on a new integer programming formulation. *Management Science*, 61(6):1275–1291, 2015.

Z.-J. M. Shen, S. Sun, Y. Qi, H. Hu, N. Kang, J. Zhang, X. Wang, and X. Lin. JD.com improves fulfillment efficiency with data-driven integrated assortment planning and inventory allocation. 55(5):386–398, 2025. ISSN 2644-0865. doi: 10.1287/inte.2025.0245. URL https://research.ebsco.com/plink/6dbb8b86-af0a-3ea5-a960-7a4559c2d99b.

D. Simchi-Levi and Y. Zhao. Safety stock positioning in supply chains with stochastic lead times. 7(4):295–318, 2005. ISSN 1523-4614. doi: 10.1287/msom.1050.0087. URL https://research.ebsco.com/plink/cb095784-1e2a-33b6-99f9-a9880b163f9a.

Y. Wang and S. Minner. Data-driven multi-location inventory placement in digital commerce. *Computers & Industrial Engineering*, 200:110842, Feb. 2025. ISSN 0360-8352. doi: 10.1016/j.cie.2024.110842. URL https://www.sciencedirect.com/science/article/pii/S0360835224009641.

Y. Zhao, T. Larsson, E. Rönnberg, and P. M. Pardalos. The fixed charge transportation problem: a strong formulation based on lagrangian decomposition and column generation. 72(3):517–538, 2018. ISSN 0925-5001, 1573-2916. doi: 10.1007/s10898-018-0661-y. URL http://link.springer.com/10.1007/s10898-018-0661-y.

# A Proofs

## A.1 Proof of Proposition 1

*Proof.* We show that there exists a polynomial-time reduction from the multiple knapsack problem to the problem in Section 3.1. Consider a multiple knapsack problem instance with item set $K = [n]$ and knapsack set $J = [m]$, where where $t_k$ and $l_k$ denote respectively the weight and value of item $k \in K$ and $W_j$ denote the capacity of knapsack $j \in J$. We construct a case-pack allocation problem instance as follows: $Q_k = t_k, N_k = 1, s_k = t_k, v_k = 1$ for $k \in K$; $C_j^L = W_j$ for $j \in J$; $f_{jk}(u) = \frac{l_k}{t_k} u$ for $j \in J, k \in K$, and $C^S = 0$. The construction is polynomial in the size of the multiple knapsack instance.

Since $C^S = 0$, constraints (1a) implies $z_k = 0$ for all $k = K$, which further implies $x_{jk} = 0$ for all $j \in$

$J, k \in K$ by constraints (1d). Consequently, only variables $y_{jk}$ remain. Constraints (1c) and (1b) reduce to $\sum_k y_{jk} \leq 1 \;\; j \in J$ and $\sum_k t_k y_{jk} \leq W_j \;\; j \in J$, respectively. Note that they coincide with multiple knapsack constraints. Last, the objective function reduces to $\sum_{j,k} f_{jk}(t_k y_{jk}) = \sum_{j,k} l_k y_{jk}$, which is identical to the objective of the multiple knapsack problem. In conclusion, both problems have identical constraints and objective function, which implies the equivalence of their optimal objective functions.

We next show that there exists a polynomial-time reduction from the partition problem to the problem in Section 3.1 when $|K| = 1$. In this case, the problem can be simplified as

$$\max_{x,y \in \mathbb{Z}_+^J} \Big\{ \sum_{j \in J} f_j(x_j + Q y_j) : \sum_j x_j \leq QB, \sum_j y_j \leq N - B, x_j + Q y_j \leq C_j^L \; \forall j \in J \Big\}.$$

Here, we suppress the index $k$ to simplify notation and define $B := \left\lfloor \frac{C^S}{Q_k} \right\rfloor$. Consider a partition problem instance with positive integers $\{a_j\}_{j \in J}$ with $\sum_{j \in J} a_j = 2T$. Without loss of generality, we assume $0 < a_j < T$ for all $j \in J$. Given such an instance, we construct an instance of our problem as follows: set $Q = T, B = 1, N - B = |J|$ and $C_j^L = T \; \forall j \in J$. For each $j$, define

$$f_j(u) = \begin{cases} 2u, & 0 \leq u \leq a_j, \\ 2a_j - \dfrac{a_j}{T - a_j}(u - a_j), & a_j \leq u \leq T, \\ a_j - 2T(u - T), & u \geq T. \end{cases}$$

Note that the break points of $f_j$ are $a_j$ and $T$, which are integers, and $f_j$ is concave. Moreover, we have $f_j(a_j) = 2a_j$ and $f_j(T) = a_j$. The construction is polynomial in the size of the partition problem instance.

We now show that there exists a subset $S \subseteq J$ satisfying $\sum_{j \in S} a_j = T$ if and only if the constructed optimization instance has objective value at least $3T$. First, suppose there exists $S \subseteq J$ with $\sum_{j \in S} a_j = T$. Define

$$\begin{cases} y_j = 1, x_j = 0 & \text{for } j \in S \\ y_j = 0, x_j = a_j & \text{for } j \notin S. \end{cases}$$

Then, $(x, y)$ satisfies all constraints as $\sum_j y_j = |S| \leq |J| = N - B$, $\sum_j x_j = \sum_j a_j = T = QB$, and $x_j + Q y_j \leq T$ for all $j$. Its objective value is $\sum_{j \in S} f_j(T) + \sum_{j \notin S} f_j(a_j) = \sum_{j \in S} q_j + \sum_{j \notin S} 2a_j = 3T$.

Conversely, suppose the constructed optimization instance has a feasible solution with objective value at least $3T$. Let $(x, y)$ denote such solution. Due to the lane capacity limit, we have $y_j \in \{0, 1\}$ and $x_j = 0$ if $y_j = 1$ for all $j$. In addition, we can show $0 \leq x_j \leq a_j$ because $f_j$ is nonincreasing on $[a_j, \infty)$. Let

$S = \{j : y_j = 1\}$. For each $j \in S$, we have $x_j = 0$ and $y_j = 1$, so $f_j(x_j + Qy_j) = f_j(T) = a_j$. For every $j \notin S$, we have $0 \le x_j \le a_j$ and $y_j = 0$, so $f_j(x_j + Qy_j) = f_j(x_j) = 2x_j$. Therefore, the total objective value is

$$\sum_{j \in S} a_j + \sum_{j \notin S} 2x_j \le \sum_{j \in S} a_j + 2\min\{T, 2T - \sum_{j \in S} a_j\},$$

where the inequality follows from

$$\sum_{j \notin S} x_j \le \sum_{j \in J} x_j \le QB = T,$$

and

$$\sum_{j \notin S} x_j \le \sum_{j \notin S} a_j = 2T - \sum_{j \in S} a_j.$$

If $\sum_{j \in S} a_j < T$, then the objective value is $\sum_{j \in S} a_j + 2T < 3T$. Else, then the objective value is $4T - \sum_{j \in S} a_j$. Since the objective value is at least $3T$, it must be that $\sum_{j \in S} a_j = T$. Therefore, the partition instance has $S \subseteq J$ satisfying $\sum_{j \in S} a_j = T$.

Therefore the decision version of the optimization problem is NP-hard. Since the reduction is from the partition problem, the hardness obtained is weak NP-hardness. We can show the weak NP-hardness of the problem when $|J| = 1$ using a similar reduction from the partition problem. □

## A.2 Proof of Proposition 2

*Proof.* Let LP (1) denote the linear relaxation of IP (1). We show that any feasible solution to LP (2) can be mapped to a feasible solution of LP (1) with at least equal objective value. Given a feasible solution $w^*$ to LP (2), we construct a solution $(x^*, y^*, z^*)$ as follows: $x^*_{jk} = \sum_{\ell \in L_k} d_{j\ell} w^*_\ell, y^*_{jk} = \sum_{\ell \in L_k} c_{j\ell} w^*_\ell$, and $z^*_k = \sum_{\ell \in L_k} b_\ell w^*_\ell$. Constraints (2b) imply $\sum_k x^*_{jk} + Q_k y^*_{jk} \le C^L_j \ \forall j \in J$. Similarly, constraint (2c) implies $\sum_k Q_k z^*_k \le C^S$. Since each pattern $\ell \in L_k$ represents a feasible allocation for product $k$, we have $\sum_j c_{j\ell} + b_\ell \le N_k$, which implies $\sum_j y^*_{jk} + z^*_k = \sum_{\ell \in L_k} \Big(\sum_j c_{j\ell} + b_\ell\Big) w_\ell \le \sum_{\ell \in L_k} N_k w_\ell = N_k$. In addition, we have $\sum_j d_{j\ell} \le Q_k b_\ell$, which implies $\sum_j x^*_{jk} = \sum_j \sum_{\ell \in L_k} d_{j\ell} w^*_\ell = \sum_{\ell \in L_k} w^*_\ell \sum_j d_{j\ell} \le \sum_{\ell \in L_k} w^*_\ell Q_k b_\ell = Q_k z^*_k$. Thus, $(x^*, y^*, z^*)$ is feasible to LP (1).

It remains to compare the objective values. By the definition of $p_\ell := \sum_j f_{jk}(d_{j\ell} + Q_k c_{j\ell})$ for pattern $\ell \in L_k$, we have

$$\sum_{j,k} f_{jk}(x^*_{jk} + Q_k y^*_{jk}) = \sum_{j,k} f_{jk}\Big(\sum_{\ell \in L_k} (d_{j\ell} + Q_k c_{j\ell}) w^*_\ell\Big) \ge \sum_{j,k} \sum_{\ell \in L_k} w^*_\ell f_{jk}(d_{j\ell} + Q_k c_{j\ell})$$

$$= \sum_k \sum_{\ell \in L_k} w^*_\ell \sum_j f_{jk}(d_{j\ell} + Q_k c_{j\ell}) = \sum_k \sum_{\ell \in L_k} w^*_\ell p_l,$$

where the inequality follows from the concavity of $f_{jk}$ and $\sum_{\ell \in L_k} w^*_\ell = 1$. □

## A.3 Proof of Lemma 3

*Proof.* We will show that there exists an optimal solution $(x,y,z)$ of IP (3) satisfying the following:

1. $z_k = \left\lceil \frac{\sum_j x_{jk}}{Q_k} \right\rceil$
2. $x_{jk} \leq Q_k - 1$
3. $x_{jk} + Q_k y_{jk} \leq T^*_{jk}$ .

Consider an arbitrary optimal solution $(x,y,z)$ to IP (3). For the sake of contradiction, suppose $(x,y,z)$ violates $z_k = \left\lceil \frac{\sum_j x_{jk}}{Q_k} \right\rceil$ for some $k$. Due to the flow balance constraint, this implies $z_k > \left\lceil \frac{\sum_j x_{jk}}{Q_k} \right\rceil$. Reducing $z_k$ to $\left\lceil \frac{\sum_j x_{jk}}{Q_k} \right\rceil$ makes the solution remain feasible and does not decrease the objective value. Therefore, we can always convert an optimal solution to satisfy $z_k = \left\lceil \frac{\sum_j x_{jk}}{Q_k} \right\rceil$.

We next suppose there exists $j,k$ such that $x_{jk} > Q_k - 1$. We can construct a feasible solution $(x',y',z)$ as follows: $x'_{jk} = x_{jk} \mod Q_k, y'_{jk} = y_{jk} + (x_{jk} - x'_{jk})/Q_k$, and all other elements remain the same. Since the total transferred unit for product $k$ to destination $j$ does not change, the objective value also remains unchanged.

Last, we prove that $(x,y,z)$ satisfies $x_{jk} + Q_k y_{jk} \leq T^*_{jk}$. Recall that $T^*_{jk}$ is either $Q_k \lceil u^*_{jk}/Q_k \rceil$ or $\min\{u^*_{jk}, C^L_j\}$. It is trivial that $x_{jk} + Q_k y_{jk} \leq C^L_j$ by the lane capacity constraint, so it is sufficient to consider when $T^*_{jk} = Q_k \lceil u^*_{jk}/Q_k \rceil$ or $u^*_{jk}$. We first show that $x_{jk} + Q_k y_{jk} \leq Q_k \lceil u^*_{jk}/Q_k \rceil$. For the sake of contradiction, suppose there exists $(x,y,z)$ with $x_{jk} + Q_k y_{jk} > Q_k \lceil u^*_{jk}/Q_k \rceil$ for some $j,k$. Since $g_{jk}$ is concave, $u^*_{jk}$ is the smallest maximizer of function $g_{jk}$, and $Q_k \lceil u^*_{jk}/Q_k \rceil \geq u^*_{jk}$, we have $g_{jk}(Q_k \lceil u^*_{jk}/Q_k \rceil) \geq g_{jk}(x_{jk} + Q_k y_{jk})$. Therefore, reducing $x_{jk}$ and/or $y_{jk}$ to satisfy $x_{jk} + Q_k y_{jk} = Q_k \lceil u^*_{jk}/Q_k \rceil$ only makes the objective value better, as it only reduces the sortation penalty ($\beta s_k \left\lceil \frac{\sum_j x_{jk}}{Q_k} \right\rceil$) but increases the adjusted benefit for destination $j$ ($g_{jk}(x_{jk} + Q_k y_{jk})$).

It remains to consider the case when $T^*_{jk} = u^*_{jk} < Q_k \lceil u^*_{jk}/Q_k \rceil$. For the sake of contradiction, suppose there exists $(x,y,z)$ with $x_{jk} + Q_k y_{jk} > u^*_{jk}$. As we have shown above, it is enough to consider the case when $x_{jk} + Q_k y_{jk} \leq Q_k \lceil u^*_{jk}/Q_k \rceil$. If $x_{jk} + Q_k y_{jk} < Q_k \lceil u^*_{jk}/Q_k \rceil$, then reducing $x_{jk}$ to satisfy $x_{jk} + Q_k y_{jk} = u^*_{jk}$ only makes the objective value better; we can prove this by using a similar argument in the previous paragraph. We now suppose $x_{jk} + Q_k y_{jk} = Q_k \lceil u^*_{jk}/Q_k \rceil$; i.e. $x_{jk} = 0, y_{jk} = \lceil u^*_{jk}/Q_k \rceil$. We will show that we can construct a solution with a better or equal objective value by using the definition of $T^*_{jk}$; the fact $T^*_{jk} = u^*_{jk}$ implies at least one of the following conditions is not met: (1) $Q_k \lceil u^*_{jk}/Q_k \rceil \leq C^L_j$, (2) $g_{jk}(Q_k \lceil u^*_{jk}/Q_k \rceil) > g_{jk}(u^*_{jk}) -$

$\beta s_k$ and (3) $g_{jk}(Q_k\lceil u^*_{jk}/Q_k\rceil) > g_{jk}(Q_k\lceil u^*_{jk}/Q_k\rceil - Q_k)$. Suppose condition (2) is not met. We construct a modified solution $(x',y',z')$ with $x'_{jk}+Q_k y'_{jk} = u^*_{jk}$ and all other $x,y$ values unchanged. Note that $z' := \left\lceil \frac{\sum_j x'_{jk}}{Q_k} \right\rceil$ is either $z$ or $z+1$ because $0 \le x'_{jk} - x_{jk} \le Q_k - 1$. We then get

$$\begin{aligned} obj(x',y',z') - obj(x,y,z) &= g_{jk}(u^*_{jk}) - g_{jk}(Q_k\lceil u^*_{jk}/Q_k\rceil) + \beta s_k(z'-z) \\ &\ge g_{jk}(u^*_{jk}) - g_{jk}(Q_k\lceil u^*_{jk}/Q_k\rceil) - \beta s_k \ge 0, \end{aligned}$$

where $obj(\cdot)$ refers to the objective value of a given solution and the last inequality is from the violation of condition (2). Finally, suppose condition (3) is not met. We construct a modified solution $(x'',y'',z'')$ with $x''_{jk}+Q_k y''_{jk} = Q_k\lceil u^*_{jk}/Q_k\rceil - Q_k$ and all other $x,y$ values unchanged. In this case, $z'' = z$ because $x''_{jk} = x_{jk} = 0$. We then get

$$obj(x'',y'',z') - obj(x,y,z) = g_{jk}(Q_k\lceil u^*_{jk}/Q_k\rceil - Q_k) - g_{jk}(Q_k\lceil u^*_{jk}/Q_k\rceil) \ge 0,$$

where the inequality is from the violation of condition (3). Therefore, we can find a solution with a better or equal objective value satisfying $x_{jk} + Q_k y_{jk} \le u^*_{jk}$. □

### A.4 Proof of Lemma 4

*Proof.* We prove the lemma by induction on $j$ from backward. For the base case when $j = |J|+1$, consider an arbitrary state $(q^x, q^y)$. We have

$$V_{|J|+1}(q^x+Q_k, q^y) = -\beta s_k \left\lceil \frac{q^x+Q_k}{Q_k} \right\rceil = -\beta s_k \left\lceil \frac{q^x}{q_K} \right\rceil - \beta s_k = V_{|J|+1}(q^x, q^y+1) - \beta s_k.$$

Suppose the induction hypothesis holds true for destinations $j = n+1,\dots,|J+1|$ for some $n \le |J|$. We will prove the induction hypothesis for $j=n$ and arbitrary $q^x, q^y$. Notice that action spaces at $(q^x+Q_k, q^y)$ and $(q^x, q^y+1)$ are equivalent; i.e., $A_n(q^x+Q_k, q^y) = A_n(q^x, q^y+1)$. By using this equivalence and the Bellman recursion, we get

$$\begin{aligned} v_n(q^x+Q_k, q^y) &= \max_{(x,y)\in A_n(q^x+Q_k,q^y)} \left\{ V_{n+1}(q^x+Q_k+x, q^y+y) + g_{nk}(x+Q_k y) \right\} \\ &= \max_{(x,y)\in A_n(q^x,q^y+1)} \left\{ V_{n+1}(q^x+Q_k+x, q^y+y) + g_{nk}(x+Q_k y) \right\} \\ &= \max_{(x,y)\in A_n(q^x,q^y+1)} \left\{ V_{n+1}(q^x+x, q^y+1+y) - \beta s_k + g_{nk}(x+Q_k y) \right\} \\ &= \max_{(x,y)\in A_n(q^x,q^y+1)} \left\{ V_{n+1}(q^x+x, q^y+1+y) + g_{nk}(x+Q_k y) \right\} - \beta s_k = v_n(q^x, q^y+1) - \beta s_k, \end{aligned}$$

where the third equality follows from the induction hypothesis. □

### A.5 Proof of Lemma 5

*Proof.* We first prove the pruning condition based on *LB*. Since $u^*_{j'k}$ maximizes $g_{j'k}$ for each $j'$, the maximum adjusted benefit we can achieve from stages $j+1$ to $|J|$ is at most $\sum_{j'=j+1}^{|J|} g_{j'k}(u^*_{j'k})$. Furthermore, the total sort penalty is at least $\beta s_k \lceil \frac{\tilde{q}^x}{Q_k} \rceil$, which is the terminal sort penalty when ignoring the individual units transferred from stages $j+1$ to $|J|$. Therefore, $\tilde{r} + \sum_{j'=j+1}^{|J|} g_{j'k}(u^*_{j'k}) - \beta s_k \lceil \frac{\tilde{q}^x}{Q_k} \rceil$ is an upper bound on the objective value we can obtain starting from state $(\tilde{q}^x, \tilde{q}^y)$ with accumulated value $\tilde{r}$.

We next justify the pruning step when $\tilde{r} < 0$. Consider the smallest $j$ such that there exist state $(q^x, q^y) \in S_j$, and action $(x,y) \in A_j(q^x,q^y)$ resulting in the next state $(\tilde{q}^x, \tilde{q}^y)$ with accumulated value $\tilde{r} < 0$. Let $r \geq 0$ denote the optimal value of state $(q^x, q^y)$. For the sake of contradiction, suppose taking action $(x,y)$ at state $(q^x, q^y)$ in stage $j$ is optimal. We can construct a new solution by taking action $(0,0)$ at state $(q^x, q^y)$ in stage $j$ and keep all other actions the same from the optimal solution. The objective value of the new solution is the objective value of the optimal solution + $r - \tilde{r}$. Since $r \geq 0 > \tilde{r}$, the objective value of the new solution is better, which contradicts to the optimality of the optimal solution. □

### A.6 Proof of Lemma 6

*Proof.* Recall that $h(z) = -\beta s_k z + \max_{x \in \mathbf{Z}^J_+} \left\{ \sum_j g_{jk}(x_{jk}) : \sum_j x_{jk} \leq Q_k z, x_{jk} \leq T^*_{jk} \ \forall j \in J \right\}$. We can relax the integrality constraints of the maximization problem (the second term) because the constraint matrix is totally unimodular and $g_{jk}$ has integer breakpoints for all $j$:

$$\begin{aligned}\phi(z) &:= \max_{x \in \mathbf{Z}^J_+} \left\{ \sum_j g_{jk}(x_{jk}) : \sum_j x_{jk} \leq Q_k z, x_{jk} \leq T^*_{jk} \ \forall j \in J \right\} \\ &\equiv \max_{x \geq 0} \left\{ \sum_j g_{jk}(x_{jk}) : \sum_j x_{jk} \leq Q_k z, x_{jk} \leq T^*_{jk} \ \forall j \in J \right\}.\end{aligned}$$

Since $-\beta s_k z$ is concave in $z$, it is enough to show the concavity of $\phi(z)$. Consider any nonnegative integers $z^1, z^2$ and $\theta \in [0,1]$. Let $x^1$ and $x^2$ be the optimizers of $\phi(z^1)$ and $\phi(z^2)$, respectively. Define $\bar{x} := \theta x^1 + (1-\theta)x^2$ and $\bar{z} := \theta z^1 + (1-\theta)z^2$. We can show that $\bar{x}$ is feasible to the maximization problem at $\bar{z}$:

$$\begin{aligned}&\sum_j \bar{x}_{jk} = \sum_j \left( \theta x^1_{jk} + (1-\theta)x^2_{jk} \right) = \theta \sum_j x^1_{jk} + (1-\theta) \sum_j x^2_{jk} \leq \theta Q_k z^1 + (1-\theta) Q_k z^2 = Q_k \bar{z}, \\ &\bar{x}_{jk} = \theta x^1_{jk} + (1-\theta)x^2_{jk} \leq \theta T^*_{jk} + (1-\theta) T^*_{jk} = T^*_{jk} \quad \forall j \in J.\end{aligned}$$

By the concavity of $g_{jk}$, we have

$$\sum_j g_{jk}(\bar{x}_{jk}) = \sum_j g_{jk}\Big(\theta x^1_{jk} + (1-\theta)x^2_{jk}\Big) \geq \theta \sum_j g_{jk}(x^1_{jk}) + (1-\theta)\sum_j g_{jk}(x^2_{jk}),$$

which implies

$$\phi(\bar{z}) \geq \sum_j g_{jk}(\bar{x}_{jk}) \geq \theta\phi(z^1) + (1-\theta)\phi(z^2).$$

□

## A.7 Proof of Lemma 7

*Proof.* To simplify notation, we suppress index $k$ and write $Q = Q_k, N = N_k, s = s_k, T_j = T^*_{jk}$ and $g_j = g_{jk}$. Define $\bar{z}(y) := N - y$. We aim to prove $H(y+1) + H(y-1) \leq 2H(y)$ for any integer $y \geq 1$, where $H(y) = \max_{z \leq \bar{z}(y)} h'(y,z)$, $h'(y,z) = \max_{x \in X(y,z)} \{\sum_{j \neq j^*} g_j(x_j) + g_{j^*}(x_{j^*} + Qy) - \beta sz\}$, and $X(y,z) := \{x \in \mathbb{Z}^J_+ : \sum_j x_j \leq Qz, x_j + Qy1_{\{j=j^*\}} \leq T_j \ \forall j \in J\}$. Define $\phi(y,z) := h'(y,z) + \beta sz$.

We first show that $\phi(y,z)$ satisfies

$$\phi(y+1,z^+) + \phi(y-1,z^-) \leq \begin{cases} \phi(y,z^+ + 1) + \phi(y,z^- - 1) & \text{if } z^+ < z^- \\ \phi(y,z^+) + \phi(y,z^-) & \text{otherwise} \end{cases},$$

for any nonnegative integers $z^+, z^-$, and $y$ with $1 \leq y \leq \min\{N, \lfloor T_{j^*}/Q \rfloor\} - 1$. Because the $-\beta sz$ terms are identical on both sides, the same inequalities also hold for the function $h'$.

Recall that the objective in $\phi(y,z)$ is $\sum_{j \neq j^*} g_j(x_j) + g_{j^*}(x_{j^*} + Qy)$. Since $g_j$ is concave with integer breakpoints, the discrete marginal benefits $\Delta_j(t) := g_j(t) - g_j(t-1)$ are nonincreasing in $t$. Therefore, an optimal solution is obtained by choosing at most $Qz$ marginal items with the largest weights among

$$\{(j^*,t) : t = Qy+1, \ldots, T_{j^*}\} \quad \text{and} \quad \bigcup_{j \neq j^*} \{(j,t) : t = 1, \ldots, T_j\}.$$

The objective value of $\phi(y,z)$ is the sum of the weights of the chosen marginal items and the fixed marginal benefits for destination $j^*$, namely $\Delta_{j^*}(1), \ldots, \Delta_{j^*}(Qy)$.

We next compare the fixed marginal benefits for $y+1$ and $y-1$. Relative to $y$, moving from $y-1$ to $y$ fixes one additional block of $Q$ marginal units of destination $j^*$, which is

$$B^- := \{(j^*,t) : t = Q(y-1)+1, \ldots, Qy\},$$

while moving from $y$ to $y+1$ fixes the next block

$$B^+ := \{(j^*,t) : t = Qy+1,\dots,Q(y+1)\}.$$

By the concavity of $g_{j^*}$, every item in $B^-$ has weight at least as large as every item in $B^+$. In particular,

$$\Delta_{j^*}(Q(y-1)+r) \geq \Delta_{j^*}(Qy+r), \qquad r = 1,\dots,Q.$$

Thus, fixing the earlier block $B^-$ is always at least as valuable as fixing the later block $B^+$.

Let

$$B := \{(j^*,t) : t = Q(y+1)+1,\dots,T_{j^*}\} \cup \bigcup_{j\neq j^*} \{(j,t) : t = 1,\dots,T_j\}.$$

For a set $S$ of marginal items, let

$$w(S) := \sum_{(j,t)\in S} \Delta_j(t).$$

For a set $A$ of marginal items, define

$$G_A(m) := \max\{w(S) : S \subseteq A,\ |S| \leq Qm\}.$$

For convenience in the cardinality arguments below, we allow distinct dummy items of zero weight to be added to any feasible selection. Thus, an optimal selection attaining $G_A(m)$ may always be represented as containing exactly $Qm$ items, without changing its weight.

Let $S^+$ attain $G_B(z^+)$ and $S^-$ attain $G_{B^-\cup B^+\cup B}(z^-)$, with zero-weight dummy items added if necessary so that $|S^+| = Qz^+$ and $|S^-| = Qz^-$. Because every item in $B^-$ has weight at least as large as every item in $B^+$, we may choose $S^-$ such that it contains either all of $B^-$ or no item of $B^+$. Using the marginal representation,

$$\phi(y+1,z^+) + \phi(y-1,z^-) = 2g_{j^*}(Qy) + w(B^+) - w(B^-) + w(S^+) + w(S^-).$$

We now consider the following two cases.

**Case 1:** $z^+ < z^-$. First, suppose $B^- \subseteq S^-$. Define $U := S^- \setminus B^-$. Then, $S^+ \cup B^+$ contains $Q(z^+ + 1)$ elements of $B^+ \cup B$, while $U$ contains $Q(z^- - 1)$ elements of $B^+ \cup B$. Therefore,

$$\begin{aligned} G_{B^+\cup B}(z^+ + 1) + G_{B^+\cup B}(z^- - 1) &\geq w(S^+) + w(B^+) + w(U) \\ &= w(S^+) + w(S^-) + w(B^+) - w(B^-). \end{aligned}$$

Next, suppose $B^- \not\subseteq S^-$. Let $V := S^- \cap B^-$ and $V' := S^- \setminus V$. Since our choice of $S^-$ implies that $S^-$ contains

no element of $B^+$, we have $V' \subseteq B$. Since $z^- \geq z^+ + 1$, we have

$$|V' \setminus S^+| \geq Qz^- - |V| - Qz^+ \geq Q - |V| > 0.$$

Choose $Q - |V|$ elements $D \subseteq V' \setminus S^+$ and $|V|$ elements $B' \subseteq B^+$. Then,

$$U^+ := S^+ \cup D \cup B', \quad U^- := V' \setminus D$$

are feasible selections of $Q(z^+ + 1)$ and $Q(z^- - 1)$ elements, respectively, from $B^+ \cup B$. Because every item in $B^-$ has weight at least as large as every item in $B^+$ and both $B^- \setminus V$ and $B^+ \setminus B'$ contain $Q - |V|$ elements, we have

$$w(B^- \setminus V) \geq w(B^+ \setminus B'),$$

which is equivalent to

$$w(B') \geq w(V) + w(B^+) - w(B^-).$$

Combining these inequalities, we obtain

$$\begin{aligned} G_{B^+ \cup B}(z^+ + 1) + G_{B^+ \cup B}(z^- - 1) &\geq w(U^+) + w(U^-) = w(S^+) + w(V') + w(B') \\ &\geq w(S^+) + w(V') + w(V) + w(B^+) - w(B^-) \\ &= w(S^+) + w(S^-) + w(B^+) - w(B^-). \end{aligned}$$

Finally,

$$\begin{aligned} \phi(y+1, z^+) + \phi(y-1, z^-) &= 2g_{j^*}(Qy) + w(B^+) - w(B^-) + w(S^+) + w(S^-) \\ &\leq 2g_{j^*}(Qy) + G_{B^+ \cup B}(z^+ + 1) + G_{B^+ \cup B}(z^- - 1) \\ &= \phi(y, z^+ + 1) + \phi(y, z^- - 1). \end{aligned}$$

**Case 2:** $z^+ \geq z^-$. If $B^- \subseteq S^-$, then both $S^+$ and $(S^- \setminus B^-) \cup B^+$ are feasible selections for $G_{B^+ \cup B}(z^+)$ and $G_{B^+ \cup B}(z^-)$, respectively, and we have

$$G_{B^+ \cup B}(z^+) + G_{B^+ \cup B}(z^-) \geq w(S^+) + w(S^-) + w(B^+) - w(B^-).$$

If $B^- \not\subseteq S^-$, again let $V := S^- \cap B^-$ and $V' := S^- \setminus V$. Choose any $|V|$ elements $B' \subseteq B^+$. Then, $V' \cup B'$ is feasible for $G_{B^+ \cup B}(z^-)$, and the same dominance argument gives

$$w(B') \geq w(V) + w(B^+) - w(B^-).$$

Consequently,

$$\begin{aligned} G_{B^+\cup B}(z^+) + G_{B^+\cup B}(z^-) &\geq w(S^+) + w(V') + w(B') \\ &\geq w(S^+) + w(S^-) + w(B^+) - w(B^-). \end{aligned}$$

Therefore, we get

$$\begin{aligned} \phi(y+1,z^+) + \phi(y-1,z^-) &= 2g_{j^*}(Qy) + w(B^+) - w(B^-) + w(S^+) + w(S^-) \\ &\leq 2g_{j^*}(Qy) + G_{B^+\cup B}(z^+) + G_{B^+\cup B}(z^-) \\ &= \phi(y,z^+) + \phi(y,z^-). \end{aligned}$$

Finally, we show the concavity of $H$. Let $z^+ \in \arg\max_{z\leq\bar{z}(y+1)} h'(y+1,z)$ and $z^- \in \arg\max_{z\leq\bar{z}(y-1)} h'(y-1,z)$. First, suppose $z^+ < z^-$. We have

$$H(y+1) + H(y-1) = h'(y+1,z^+) + h'(y-1,z^-) \leq h'(y,z^+ + 1) + h'(y,z^- - 1) \leq H(y) + H(y),$$

where the last inequality follows from $0 \leq z^+ + 1 \leq \bar{z}(y)$ and $0 \leq z^- - 1 \leq \bar{z}(y)$. Next, suppose $z^+ \geq z^-$. Since $\bar{z}(y+1) \leq \bar{z}(y)$, we have $z^- \leq z^+ \leq \bar{z}(y)$. Therefore,

$$H(y+1) + H(y-1) = h'(y+1,z^+) + h'(y-1,z^-) \leq h'(y,z^+) + h'(y,z^-) \leq H(y) + H(y).$$

□