Proceedings of the 2025 Winter Simulation Conference
E. Azar, A. Djanatliev, A. Harper, C. Kogler, V. Ramamohan, A. Anagnostou, and S. J. E. Taylor, eds.

SIMULATION-BASED ONLINE RETAILER SUPPLY CHAIN INBOUND NODE ARRIVAL CAPACITY CONTROL

Michael Bloem[1], Song (Sam) Zhou[1], Kai He[1], Zhunyou (Jony) Hua[1], and Yan Xia[2]

[1]Supply Chain Optimization Technologies, Amazon, Bellevue, WA, USA
[2]Mechatronics & Sustainable Packaging, Amazon, Bellevue, WA, USA

ABSTRACT

Online retailer supply chain management involves decisions about how much and when to buy inventory, where to inbound new inventory, how to transfer inventory between warehouses, and how to fulfill customer orders. These choices must adhere to capacity constraints, such as labor plans, while minimizing impacts on customer service and profitability. This paper presents a dual decomposition framework and simulation-based cost search approach that maintains high customer service levels while better aligning buying purchase orders with inbound node arrival capacity plans. The methodology is evaluated through end-to-end discrete event simulations, which quantify the impact of candidate capacity costs on buying system behavior, as well as the downstream effects on other systems and ultimately on customer service metrics. Results demonstrate this approach can improve capacity adherence by over 60% across a five-week horizon, with less than a 4% degradation in a metric measuring the proximity of inventory to customers.

1 INTRODUCTION: ONLINE RETAILER SUPPLY CHAIN CAPACITY CONTROL

Operating the supply chain for an online retailer like Amazon or Walmart requires a variety of interconnected systems. Figure 1 is a simplified depiction of such a supply chain and the different systems involved in its operation. Buying systems determine when and how much of each product to order from vendors and to each facility, based on inputs such as demand forecasts and the current inventory state. Placement systems determine when and how to transfer inventory between facilities. For example, at a cross dock facility, placement systems determine how to distribute new inventory arrivals from vendors to downstream inventory-holding warehouses. Placement systems also send replenishment transfers from upstream storage facilities to downstream customer-facing warehouses. Fulfillment systems determine how to use inventory at customer-facing warehouses to fulfill customer purchases made on the retailer's website. For the largest online retailers, a national network can consist of millions of products, tens of thousands of vendors, dozens of cross docks, hundreds of warehouses, and hundreds of customer demand regions.

All of these buying, placement, and fulfillment decisions for millions of products must jointly adhere to capacity constraints. These include relatively static constraints, such as the physical storage capacity of warehouses, and more dynamic constraints, such as the amount of labor available to receive inventory or the number and size of trucks available to transfer inventory. Failure to adhere to the constraints leads to additional operational expenses, such as higher costs of stowing and picking items in full warehouses or reactive rerouting of trucks away from a warehouse with an excessive backlog of trailers to be received. When adhering to constraints, access to constrained capacity should be granted to various products so as to maximize profitability or minimize costs. Such potential financial outcomes may be difficult to estimate when making operational decisions, so capacity may be allocated to optimize proxy business objectives like customer service or supply chain performance metrics. These could include the fraction of customer orders that can be filled quickly and inexpensively from a nearby warehouse.

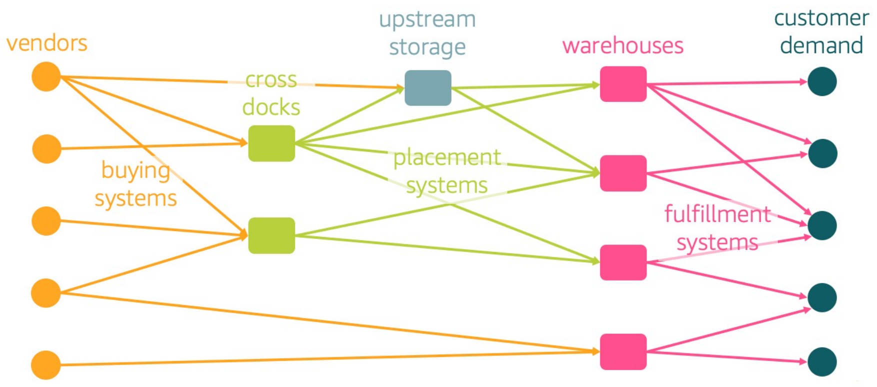


Figure 1: Overview of online retailer supply chain.

1.1 Challenges

There are five related core challenges associated with managing capacity in an online retailer supply chain.

1. Intractable cross-product problem: Simultaneously solving for capacity-adherent node-level inventory flows and storage volumes across all products is not tractable because there are millions of products, each with different inventory states and economic terms. Therefore, a practical approach must decompose the overall cross-product capacitated inventory management problem into manageable sub-problems. This restricts the set of viable solutions.
2. Sequential decision-making under uncertainty by disparate systems: Even for a single product, buying, placement, and fulfillment decisions happen at different times and with different data, binding constraints, and latency requirements. Therefore, practical approaches also decompose the overall problem into sub-problems for buying, placement, and fulfillment, all of which impact financial outcomes and change the shared inventory state over time. These are made across time and in the face of uncertainty in customer demand and lead times. Each must make some assumptions about the behavior of other systems.
3. Determining which products gain access to capacity-constrained resources: To ensure operational feasibility, the disparate decision systems must eventually determine that certain products gain access to constrained resources while others do not. This can be performed online with real-time systems, which inevitably incur some regret due a lack of foresight. Proactive mechanisms must determine how to manage and be robust to uncertainty in forecasted future outcomes.
4. Modeling and optimizing financial or performance metric objectives: Due to financial model inaccuracies, price and cost fluctuations, and limitations required to ensure optimization model tractability, most approaches cannot explicitly optimize for long-term profitability. Furthermore, the objective functions used in buying, placement, and fulfillment systems can be overlapping in time. For example, a buying decision might approximate the placement and fulfillment costs that are later explicitly optimized by other systems. Even the buying decision made in one week might consider demand in a time period that is reconsidered in the next week's buying decision. One exception would be reinforcement learning policy decision systems (Madeka, Torkkola, Eisenach, Luo, Foster, and Kakade 2022), but these have not yet been extended to a coherent (multi-agent) joint reinforcement learning approach for end-to-end inventory management. An implication is that simply adding up the objectives across many decisions for a product would double-count the same costs and cannot be trusted as an estimate of the product's overall profitability objective.
5. Constraint infeasibility: Capacity planning is typically performed by analyzing forecasts of aggregate inventory flows, while assuming that such flows are fungible across nodes. In practice, per-product buying, placement, and fulfillment decisions must adhere to a variety of constraints that prevent

inventory from being moved arbitrarily between nodes. Some of these are physical, such as those related to the containers in which inventory arrives and is transferred. Others are self-imposed constraints designed to lead to more tractable problems or to achieve various business objectives. For example, the inbound nodes made available to buying models may be limited to encourage the buying model to order larger quantities into a smaller set of nodes, which can reduce inbound costs. Such per-product constraints can make the aggregate cross-product plans infeasible in ways that are not evident until cross-product capacity control is attempted.

1.2 Related research

Simulation-based optimization is a well-known and widely applied technique (Gosavi 2015). Capacity-constrained supply chains operating under uncertainty are challenging to model mathematically, so it is not surprising that this flexible technique has been applied to optimize parameters of decision-making systems in supply chain and inventory management settings (Jung et al. 2004; Schwartz et al. 2006; Mele et al. 2006). In this literature, simulations are used to optimize parameters of decision-making systems, but we found no examples of using simulations to find opportunity costs that ensure utilization of shared resources adheres to capacity constraints.

1.3 Contributions

In this work, we propose a tractable approach to manage inbound node arrival capacity constraints. This approach scales to the largest and most complex supply chains currently operated by online retailers: it can be applied to drive automated buying of tens of millions of products to adhere to weekly constraints at hundreds of inbound nodes. At the core of the approach is a simulation-based search for node-week capacity costs ingested by the optimization model used by buying systems, a novel framework for addressing the challenges listed above. A third and final contribution is using the simulation of customer-service metrics to quantify the impact of the capacity control mechanism, rather than relying on buying or placement system objective functions, which can overlap and involve various approximations to maintain tractability.

1.4 Document structure

The remainder of this document is structured as follows. Section 2 describes the overall inbound node capacity control framework. This section begins by reviewing the nature of the constraints in sub-section 2.2, then describes the node-level buying system in sub-section 2.3. The capacity cost search framework is introduced in sub-section 2.4.1 and detailed in sub-section 2.4.3. In between, sub-section 2.4.2 describes the simulator at the core of the capacity cost search. Section 3 describes empirical results demonstrating that the approach is tractable at scale and can improve capacity adherence by more than 60% while only degrading a customer service metric by less than 4%. Conclusions are provided in Section 4.

2 INBOUND NODE CAPACITY CONTROL VIA CAPACITY COSTS

2.1 Notation

Let $\mathscr{A}$ be the full selection of products that are sold by an online retailer. A product's inventory dynamics depend on a sequence of decentralized buying, placement, fulfillment, pricing, and other decisions and other events. Many of these events are stochastic, such as the time it takes for a vendor to fulfill an order, the time that inventory spends in an warehouse backlog, or the quantity of customer demand that is ultimately realized. Let $\mathbf{S}_t^a$ be the state of product a at time t. This state variable captures all data required to simulate the product, such as the current inventory state, inflight transfers, and forecasted demand. The ordered sequence of buying, placement, fulfillment, or other decision events that influence prodcut a's consumption of capacity is denoted by $\mathscr{D}^a$. Associated with each decision event $\mathscr{D}^a[i], i = 1, 2, \ldots, |\mathscr{D}^a|$ is a time $t^a[i]$, but these decision times and even the sequence $\mathscr{D}^a$ itself is not necessarily known in advance. For example, the times of placement decisions made at cross dock facilities are influenced by how long it takes for an order to arrive. Furthermore, decisions events may create a variable number of later events. For example, a buying decision event may result in a purchase order to one or two cross docks, which lead to one or two future placement decision events. Let $\pi(\mathbf{S}_t^a)$ denote a decision-making policy. Given the state, the policy returns a decision $\mathbf{X}_t^a$ from the corresponding set of feasible decisions $\mathscr{X}(\mathbf{S}_t^a)$. The decision could describe, for example, a set of purchase orders or inventory transfer events.

2.2 Constraints

We consider an inbound capacity controller which is refreshed weekly and covers the capacity utilizations in the next $\mathscr{T}$ weeks, where $\mathscr{T}$ is chosen so that the majority of orders have lead time less than $\mathscr{T}$ weeks. Given node-level labor plans $u_{j,t'}$, then across all buying events and for each time step t and at each inbound node j, we require

$$\mathop{\mathbf{E}}_{\xi}\left[\sum_{a\in\mathscr{A}}\sum_{t\in\mathscr{T}^{a,\text{buy}}} U_{j,t'}^{a}(\mathbf{X}_t^{a,\text{buy}})\right] = u_{j,t'} \quad \forall j \in \mathscr{J}, t' \in \mathscr{T}, \tag{1}$$

where $\mathbf{X}_t^{a,\text{buy}}$ denotes the node-level order quantity, and $U_{j,t'}^{a}$ denotes the arrival of product a into node j at time t'. In other words, we want to meet our node-level labor plans exactly, for all weeks considered. Node-level labor plans $u_{j,t'}$ are at least as large as the arrivals from orders in-flight, since the plans would be infeasible otherwise.

2.3 Node-level Buying

Following each product's schedule, for each product, buying systems decide and submit the number of units to procure in the current cycle. Each buying cycle aims to cover demand during a planned period called the planning horizon, a function of the vendor lead time and review periods. The model considers three major inventory supplies/pools: on-hand inventory, in-flight orders and new orders into each nodes in the network, New orders determine the inbound arrival volumes at each node in future time periods.

For each buying event $e^{a,\text{buy}}$, the buying policy $\pi^{\text{buy}}(\mathbf{S}_t^a)$ determines the quantities to purchase from available vendors to each inbound warehouse/node, based on the inventory state $\mathbf{S}_t^a$ (*e.g.*, on-hand inventory and in-flight orders). When capacity constraints are ignored, the decisions returned by $\pi^{\text{buy}}(\mathbf{S}_t^a)$ for product a at time t aims to solve the following optimization problem:

$$\begin{array}{ll} \text{minimize} & \sum_{a\in\mathscr{A}}\sum_{t\in\mathscr{T}^{a,\text{buy}}} \mathbf{E}_{\xi}\left[c_{e_t^{a,\text{buy}}}(\mathbf{S}_t^a, \mathbf{X}_t^{a,\text{buy}})\right] \\ \text{subject to} & \mathbf{X}_t^{a,\text{buy}} \in \mathscr{X}_{e^{a,\text{buy}}}(\mathbf{S}_t^a), \end{array} \tag{2}$$

where the objective function $c_{e^{a,\text{buy}}}$ includes various cost terms, such as vendor sourcing cost, inventory transfer cost, shipping cost, lost sale penalties and overage costs.

Remark 1: Sub-optimality of rolling horizon production buying systems. As discussed in subsection 1.1, this is a sequential decision-making problem under uncertainty. Ideally, we would formulate the product-level decision-making process as a dynamic programming or reinforcement learning problem. This rolling horizon approach is instead used because it is more tractable and interpretable, but it is also a heuristic that will not necessarily produce the optimal sequence of buying decisions over time (when demand is not stationary, for example).

Remark 2: Overlapping objective functions. The buying objective function $c_t^{a,\text{buy}}$ can be viewed as an approximation to the objective function in a dynamic programming formulation, but with limited modeling of the dynamics of the decision-making process and cost-to-go terms due to latency, model complexity, and input availability considerations. Furthermore, since buying planning horizons across subsequent weeks can overlap, the overall objective (3a) double-counts certain costs.

2.4 Capacity Cost Search

2.4.1 Overview and dual decomposition framework

In this section, we describe how to influence buying to achieve inbound node-week level capacity control. Our approach drives adherence to shared cross-product capacity constraints with a Lagrangian dual decomposition framework. This framework permits independent product buying decisions to become shared-constraint aware by incorporating costs on capacity consumption into their objective functions. The cost search iteratively updates the capacity/dual/opportunity costs with a second-order subgradient ascent algorithm, where the node-week arrival constraint violations are interpreted as a sub-gradient of the dual maximization problem.

As the foundation of the dual decomposition framework for capacity management, we formulate the following cross-product optimization problem over all products and all buying events:

$$\text{minimize} \sum_{a\in\mathscr{A}} \sum_{t\in\mathscr{T}^{a,\text{buy}}} \mathop{\mathbf{E}}_{\xi}\left[c_{e_t^{a,\text{buy}}}(\mathbf{S}_t^a, \mathbf{X}_t^{a,\text{buy}})\right] \tag{3a}$$

$$\text{subject to } \mathbf{X}_t^{a,\text{buy}} \in \mathscr{X}_{e_t^{a,\text{buy}}}(\mathbf{S}_t^a) \quad \forall a\in\mathscr{A}, \forall t\in\mathscr{T}^{a,\text{buy}}, \tag{3b}$$

$$\mathop{\mathbf{E}}_{\xi}\left[\sum_{a\in\mathscr{A}} \sum_{t\in\mathscr{T}^{a,\text{buy}}} U_{j,t'}^a(\mathbf{X}_t^{a,\text{buy}})\right] = u_{j,t'} \quad \forall j\in\mathscr{J}, t'\in\mathscr{T}. \tag{3c}$$

Remark 3. Sub-optimality of rolling horizon formulation. Like the production buying systems (see Remark 1), this framework poses a rolling horizon capacity control problem. Although tractable and interpretable, it will not produce the optimal sequence of constrained buying decisions. The overall dual decompostion and simulation-based cost search framework described in this document can be extended to work with dynamic programming or reinforcement learning decision system policies (Madeka et al. 2022). Furthermore, the overall cost search framework itself can also be extended into a reinforcement learning cost search selection policy (Eisenach et al. 2024).

The rolling horizon formulations and overlapping objective functions described in Remarks 1–3 preclude us from designing a mechanism to achieve global optimality, especially while limiting ourselves to minor modifications to π^{buy} that are possible under production constraints. Given that decisions for different products are largely independent, we draw inspiration from dual decomposition theory (Boyd et al. 2015), which suggests that the shared constraints be handled by adjusting the objective function in π^{buy} so that it penalizes utilization of the corresponding capacity-constrained resources. In particular, the penalty term is simply an opportunity cost (also referred to as capacity cost or dual variable) multiplied by the amount of the resource consumed. Let $\lambda_{j,t} \in \mathbf{R}$ be the capacity cost associated with arrivals to inbound node j and week t. We then seek $\lambda = \{\lambda_{j,t}\}_{j\in\mathscr{J}, t\in\mathscr{T}}$ that will satisfy equation (3c) above and equation (4) below:

$$\mathbf{X}_t^{\star a,\text{buy}} \in \mathop{\text{argmin}}_{\mathbf{X}_t^{a,\text{buy}} \in \mathscr{X}_{e_t^{a,\text{buy}}}(\mathbf{S}_t^a)} \mathop{\mathbf{E}}_{\xi}\left[c_{e_t^{a,\text{buy}}}(\mathbf{S}_t^a, \mathbf{X}_t^{a,\text{buy}}) + \sum_{j\in\mathscr{J}} \sum_{t'\in\mathscr{T}:t'\geq t} \lambda_{j,t'} U_{j,t'}^a(\mathbf{X}_t^{a,\text{buy}})\right] \quad \forall a\in\mathscr{A}, t\in\mathscr{T}^{a,\text{buy}}. \tag{4}$$

These ensure that aggregate arrivals adhere to shared capacity constraints and individual product buying decisions are optimal (relative to the modified objective), respectively. They encompass the Karush-Kuhn-Tucker (KKT) optimality conditions for problem (3a)–(3c) (Boyd and Vandenberghe 2004).

Algorithms for a dual decomposition framework involve solving the following max-min dual optimization problem to find appropriate dual variables:

$$\max_{\lambda} \min_{\left\{\mathbf{X}_{t(a,\text{buy})}^{a,\text{buy}}\right\}} \sum_{a\in\mathscr{A}} \sum_{t\in\mathscr{T}^{a,\text{buy}}} \mathop{\mathbf{E}}_{\xi}\left[c_{e_t^{a,\text{buy}}}(\mathbf{S}_t^a, \mathbf{X}_t^{a,\text{buy}}) + \sum_{j\in\mathscr{J}} \sum_{t'\in\mathscr{T}} \lambda_{j,t'} \left(\sum_{a\in\mathscr{A}} \sum_{t\in\mathscr{T}^{a,\text{buy}}} U_{j,t'}^a(\mathbf{X}_t^{a,\text{buy}}) - u_{j,t'}\right)\right]. \tag{5}$$

We can decompose the inner minimization problem into single-product sub-problems that can be solved in parallel per-product simulations. Let $\pi^{\text{buy}}(\mathbf{S}_t^a;\lambda)$ be a modified buying policy. This policy solves the same optimization model as $\pi^{\text{buy}}(\mathbf{S}_t^a)$, but with an objective function that includes the dual capacity cost penalty:

$$\begin{array}{ll} \text{minimize} & \mathbf{E}_\xi\left[c_{e^{a,\text{buy}}}(\mathbf{S}_t^a,\mathbf{X}_t^{a,\text{buy}}) + \sum_{j\in\mathscr{J}}\sum_{t'\geq t}\lambda_{j,t'}U_{j,t'}^a(\mathbf{X}_t^{a,\text{buy}})\right] \\ \text{subject to} & \mathbf{X}_t^{a,\text{buy}} \in \mathscr{X}_{e^{a,\text{buy}}}(\mathbf{S}_t^a). \end{array} \tag{6}$$

At each iteration k of the cost search, the current costs $\lambda^{(k)}$ are used in simulated buying policy actions $\pi^{\text{buy}}(\cdot;\lambda^{(k)})$. Then simulated arrivals based on all products' decisions $\left\{\mathbf{X}_t^{a,\text{buy}}\right\}$ are aggregated to $\tilde{U}_{j,t'}(\lambda^{(k)}) = \sum_{a\in\mathscr{A}}\sum_{t\in\mathscr{T}^{a,\text{buy}}}U_{j,t'}^a(\mathbf{X}_t^{a,\text{buy}})$ and compared with the inbound capacity constraint $u_{j,t'}$ at each inbound warehouse and each week. The cost search then performs an update on $\lambda^{(k)}$ with a second-order (sub-)gradient ascent algorithm so that arrival flows are shifted from constrained node-weeks to less constrained ones. Figure 2 shows this interaction. In the following subsections, we will describe in detail a cost search algorithm that can effectively find control parameters λ that will approximately satisfy equations (3c) and (4). Before we do, we introduce the simulator that the algorithm relies on to solve the decomposed modified buying decisions (4) across all products and buying events, as well as to estimate the impact on customer service metrics.

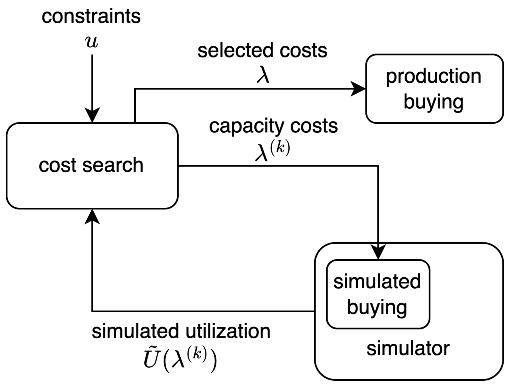


Figure 2: Capacity cost search.

2.4.2 Simulator

Multiple sequential buying, placement, and fulfillment decisions determine a product's future consumption of capacity-constrained resources over time in the supply chain network. A discrete-event simulation of these future events based on forecast input data can help ensure capacity adherence across all of these events. Let $\mathscr{A}$ be the full population of products that must adhere to the capacity constraints and let $\tilde{\mathscr{A}} \subset \mathscr{A}$ be a simulated sample of those products; associated with each sampled product is a sampling scaling factor $\gamma^a \geq 1$ used to scale its simulated statistics to produce an estimate of the aggregate population metric. Each product $a \in \tilde{\mathscr{A}}$ is simulated in τ_a trials; each trial uses different samples from a random number generator to sample from distributions that model stochastic events. Algorithm 1 informally describes such a discrete-event simulator and how it would generate the sequence of decision events $\mathscr{D}^a$ for a particular product a and trial.

To estimate the total utilization of resources across all products, we do not simulate every single product. Instead, we use a sampling methodology based on stratified sampling with Neyman allocation (SSNA) to achieve sufficiently small sampling errors while reducing the simulation computation requirements by more than 10x.

Algorithm 1: Discrete-event simulation trial for one product a.

```
Data: initial $\mathbf{S}_t^a$, random number generator seed
1  initialize empty sequence of decision events $\mathscr{D}^a$
2  initialize random number generator (RNG) with seed
3  initial time-ordered event queue $\Delta^a$, sampling from distributions with RNG as needed
4  $i \leftarrow 0$
5  while ($\Delta^a$ is not empty) and ($t <$ simulation end time) do
6      Pop next future event $e$ off of $\Delta^a$
7      $t \leftarrow$ time of event
8      if $e$ is not a decision event then
9          simulate $e$, sampling from distributions with RNG as needed
10     else
11         $i{+}{+}$
12         $t^a[i] \leftarrow t$
13         $\mathscr{D}^a[i] \leftarrow e$
14         $\mathbf{X}_t^a \leftarrow \pi_e(\mathbf{S}_t^a)$     // $\pi_e$ is the decision-making system corresponding to $e$
15     end
16     compute $\mathbf{S}_{t+}^a$ based on $\mathbf{S}_t^a$ and simulation of $e$     // $\mathbf{S}_{t+}^a$ is state after event at $t$
17     update simulation statistics based on $t$, $\mathbf{S}_t^a$, $\mathbf{X}_t^a$, and $\mathbf{S}_{t+}^a$
18     push future events generated by simulation of $e$ onto $\Delta^a$
19     $\mathbf{S}_t^a \leftarrow \mathbf{S}_{t+}^a$     // Update current state to state after event $e$
20 end
```

Buying quality metrics: Various simulated metrics measure how well we accomplish Buying systems' goal of ordering the right inventory at the right time to the right place to provide faster delivery speeds and lower fulfillment costs. These metrics are calculated in Algorithm 1 step 17 based on simulated customer orders and our supply availability at customer-facing outbound nodes. In particular, we compute "preferred node availability" to capture the fraction of customer orders for which the ordered item was available in a pre-specified "preferred" fulfillment node mapped to the corresponding customer location. In production, the metric is defined using actual customer order data and a real-time product supply picture archived at the time of customer order. In simulation, we currently approximate this with the 60th percentile of the weekly product demand forecast and the weekly average simulated end-of-day inventory.

2.4.3 Sequential Quadratic Programming Cost Search

The sub-sections that follow will leverage the following notation. The superscript (k) will be used to specify the value of a quantity at iteration $k \geq 0$. The sub-script h will be used to index a (j,t') node-week tuple for which we have a constraint. Furthermore, let H be the number of node-week constraints. Let $\tilde{U}_h(\lambda^{(k)}) = \sum_{a\in\mathscr{A}} \sum_{t\in\mathscr{T}^{a,\text{buy}}} U_h^a(\mathbf{X}_t^{a,\text{buy}}(\lambda^{(k)}))$ represent the consumption of the resource constrained by u_h, observed in the simulation when the opportunity cost vector is $\lambda^{(k)}$. Let $\tilde{U}(\lambda^{(k)})$ and u be corresponding H-dimensional vectors. Then let $V^{(k)} = \tilde{U}(\lambda^{(k)}) - u$ be the H-dimensional vector of capacity violations observed in simulation using $\lambda^{(k)}$. Let $\delta_\lambda^{(k)}$ be the cost step to find the next candidate cost vector $\lambda^{(k+1)}$ from the current cost vector: $\lambda^{(k+1)} = \lambda^{(k)} + \delta_\lambda^{(k)}$ Let $\delta_V^{(k)}$ be the change in violation from iteration k to $k+1$: $\delta_V^{(k)} = V^{(k+1)} - V^{(k)}$.

Sub-gradient of the dual problem: The dual problem is generally non-differentiable, so we rely on a subgradient at each iteration to design the search algorithm. A subgradient for the dual problem (5) is readily available at the end of the simulation run in iteration k: the simulated constraint violation vector $V^{(k)}$

when the opportunity costs in the simulation are $\lambda^{(k)}$. This subgradient is intuitive: it suggests increasing the cost when the violation is positive and vice versa.

Second-order matrix estimate: Buying decisions to order inventory into candidate nodes are zero-sum because the buying model responding to capacity costs is designed to determine inbound node order quantities, not to find the aggregate order quantity. Furthermore, increasing the capacity cost of buying into one node-week, even while holding all other costs constant, will impact allocations to other node-weeks because they become relatively more attractive under this cost increase. To allow cost search to capture and respond to such cross-constraint relationships, we maintain and iteratively update an estimate of a second-order Hessian-like matrix $\mathbf{H}^{(k)}$ using the opportunity cost vectors and corresponding subgradient vectors from earlier cost search iterations. More precisely, the $h\hat{h}$th element of the second-order matrix represents the marginal change in subgradient component h (*i.e.*, change in violation of constraint h) per unit change in cost component $\hat{h}$, at the current cost vector: $\mathbf{H}_{h\hat{h}} = \frac{\Delta V_h}{\Delta \lambda_{\hat{h}}}$. We initialize a diagonal Hessian estimate with the results of an initial simulation with small uniform costs and another simulation after a heuristic initial cost step. We iteratively update the estimate of the Hessian using BFGS updates (Johnson 2019), which are based on the cost and violation vectors from that iteration's simulation and the previous iteration's simulation:

$$\mathbf{H}^{(k+1)} = \mathbf{H}^{(k)} + \frac{\delta_V^{(k)\top}\delta_V^{(k)}}{\delta_V^{(k)\top}\delta_\lambda^{(k)}} - \frac{\mathbf{H}^{(k)}\delta_\lambda^{(k)}\delta_\lambda^{(k)\top}\mathbf{H}^{(k)}}{\delta_\lambda^{(k)}\mathbf{H}^{(k)}\delta_\lambda^{(k)}}. \tag{7}$$

Step size constraints: Sequential quadratic programming (SQP) and related Newton step algorithms typically select step sizes smaller than those suggested by the solution of the QP solved at each iteration because the QP is only a *local* second-order approximation of the optimization objective function. Our setting provides additional reasons for us to suspect that the QP might be a poor local approximation. We have no closed-form expression for the Hessian-like matrix at the current cost vector, so we must rely on an estimate based on data from earlier iterations. While the gradient at a given point is unique, there can be non-unique subgradients of the generally non-differentiable dual objective function at a given cost vector. Finally, given how production systems violate assumptions of the dual decomposition formulation, we do not expect the violation vector to necessarily always have the properties of a subgradient vector. Unfortunately, given that every evaluation of a candidate cost vector involves an expensive simulation and the lack of a global objective function, we cannot perform the trust region or line search methods that are typically employed to determine step sizes. We therefore impose multiple other heuristic step size constraints, drawing inspiration from step size selection approaches developed for stochastic gradient descent. The first step size constraint is an exponentially-decayed initial maximum step size. The next step size constraint is an adaptive "bisection" step size constraint designed to prevent steps from going beyond recently-explored costs that led to a violation with a different sign (*e.g.*, led to under-utilization of the constraint when the latest iteration violated the constraint). In particular, the constraint restricts cost steps on an element-by-element basis (*i.e.*, per cost vector component λ_h) based on $\lambda_h^{(k)}$ and $V_h^{(k)}$ in the last few iterations. Cost steps are restricted to be smaller than one half the distance to a recent cost with a corresponding violation whose sign differs from the sign of the violation corresponding to the current cost. To simplify notation, we will at iteration k denote the most binding of these constraints on positive cost steps as $\bar{\delta}_{\lambda,h}^{(k)}$ and denote the most binding of these constraints on negative cost steps as $\underline{\delta}_{\lambda,h}^{(k)}$.

Cutting plane constraints: We observed empirically that cost search was slowed when the QP sometimes suggested cost steps with a sign that is opposed to the corresponding element of the current subgradient (*e.g.*, increasing the cost for a constraint that is already not violated). This behavior typically is caused by an over-reaction by the Hessian estimate to large changes in constraint violations across iterations. Therefore, we forbid cost steps with a sign that contradicts that of the corresponding element of the latest sub-gradient. This idea is related to cutting-plane methods, but we aggressively generate a cut along each dimension based on the sign of the corresponding element of the current sub-gradient (violation) vector. We call these *element-wise subgradient consistency* constraints.

SQP for cost steps: We now present the QP solved at iteration k to determine the cost step $\delta_\lambda^{(k)}$ that will specify the next candidate cost vector $\lambda^{(k+1)}$:

$$\text{maximize } \frac{1}{2}\delta_\lambda^\top \mathbf{H}^{(k)}\delta_\lambda + \delta_\lambda^\top V^{(k)} \tag{8a}$$

$$\text{subject to } \underline{\delta}_{\lambda,h}^{(k)} \le \delta_{\lambda,h} \le \bar{\delta}_{\lambda,h}^{(k)}, \quad h = 1,\dots,H \tag{8b}$$

$$\delta_{\lambda,h} \ge 0 \text{ if } V_h^{(k)} \ge 0 \text{ else } \delta_{\lambda,h} < 0, \quad h = 1,\dots,H. \tag{8c}$$

The objective (8a) is a second-order local approximation of the dual objective based on the latest subgradient vector $V^{(k)}$ and Hessian-like matrix estimate $\mathbf{H}^{(k)}$. The dual decomposition framework relies on the simulated decisions at each iteration to accomplish the minimization in each decomposed primal problem in the overall dual objective (5). Constraints (8b) summarize the step size constraints. Constraints (8c) enforce element-wise subgradient consistency.

The overall simulation-based capacity cost search algorithm is summarized in Algorithm 2.

Algorithm 2: Simulation-based cost search.

```
Data: constraints $u$, SSNA sample $\tilde{\mathscr{A}}$ and scaling factor per product $\gamma^a \ge 1$, starting costs $\lambda^0$,
      max permitted iterations $K$
1  initialize costs $\lambda_h^{(0)} \leftarrow \lambda^0$ for all $h = 1,\dots,H$
2  $k \leftarrow 0$
3  while $(k < K)$ and (not achieved stopping criteria) do
4      parallel for $a \in \tilde{\mathscr{A}}$ do
5          simulate product $a$ using Algorithm (1) with $\lambda^{(k)}$
6          persist simulated utilization $u_h^a$ for all $h = 1,\dots,H$
7      end
8      $\tilde{U}_h(\lambda^{(k)}) \leftarrow \sum_{a\in\tilde{\mathscr{A}}} \gamma^a u_h^a$ for all $h = 1,\dots,H$
9      $V^{(k)} \leftarrow \tilde{U}(\lambda^{(k)}) - u$
10     Update $\mathbf{H}^{(k-1)}$ to $\mathbf{H}^{(k)}$ with equation (7)
11     Solve QP (8a)–(8c) to find $\delta_\lambda$
12     $\lambda^{(k+1)} \leftarrow \lambda^{(k)} + \delta_\lambda$
13     $k{+}{+}$
14 end
```

3 EMPIRICAL RESULTS

To illustrate the performance of the proposed simulation-based cost search methodology, we describe simulated performance on an example real-world scenario. This cost search scenario is for the supply chain network supporting certain products sold in the Amazon.com online retail business in the continental United States. To avoid divulging proprietary details of Amazon's operations, we do not report or plot absolute values of relevant metrics. The cost search was conducted during the week of 2025-02-09 to drive adherence to inbound node capacity constraints for the next five weeks. The scenario included more than one hundred inbound warehouses and therefore more than five hundred unique warehouse-week constraints. This cost search continued for 30 iterations, which is the maximum number of iterations that could be completed before the final costs were required to perform production buying. The costs from the 27th iteration were promoted to production because they achieved the best simulated capacity adherence.

3.1 Capacity adherence

Figure 3 shows the change in capacity adherence across cost search iterations. Capacity adherence is measured with constraint-weighted mean absolute percent deviation of utilization from the constraint (wMAPD); lower values represent superior adherence to constraints. The figure also plots a lower bound on the achievable wMAPD, which is computed by considering how inbound node restrictions prevent arbitrary reallocations of weekly order quantities between inbound nodes (see topic 5 in sub-section 1.1; lower bound computation details are outside the scope of this paper). The simulated wMAPD does not decrease monotonically across iterations, but it cumulatively decreases 63% from about 5x the lower bound to about double the lower bound.

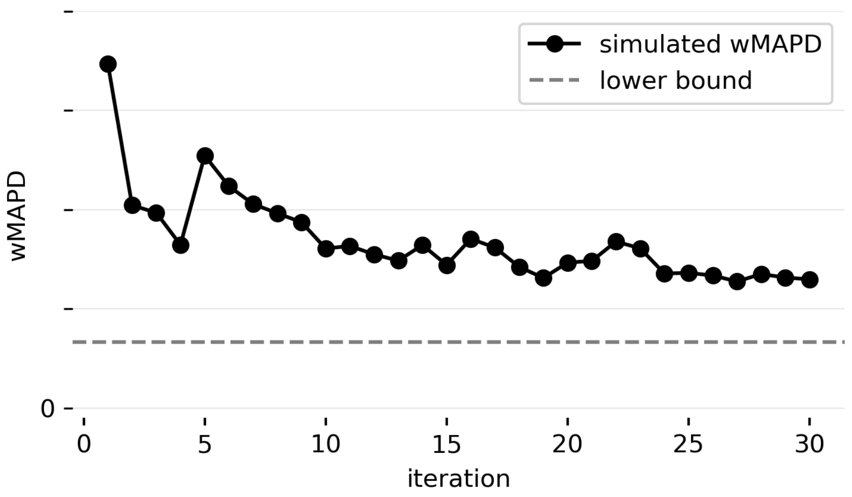
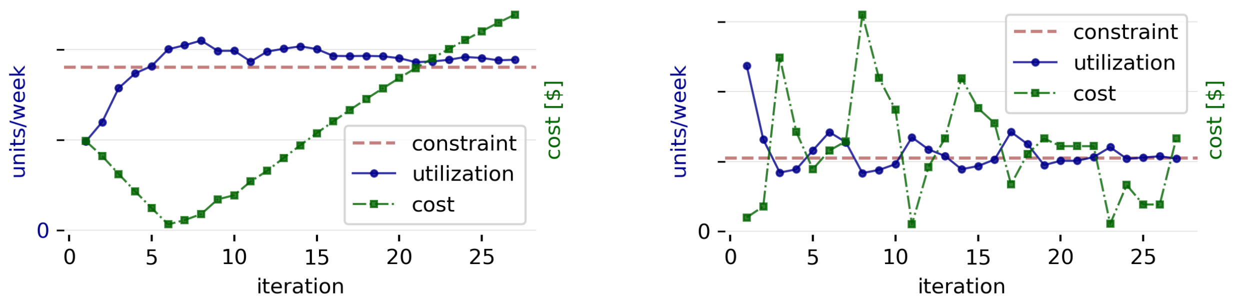


Figure 3: Capacity adherence across iterations.

Figure 4a and 4b shows the changes in capacity cost and inbound units utilization across iterations for two example node-weeks. Unconstrained utilization of the first example warehouse-week is below the constraint, causing the cost to decrease. This decrease and changes in costs at alternative warehouse-weeks eventually incentivizes the buying model to order more units into this warehouse-week. By the 6th iteration, the utilization exceeds the constraint and the cost begins increasing. The cost continues to increase for the remainder of the cost search, gradually reducing the utilization closer to the constraint. Unconstrained utilization of the second warehouse-week was more than double the constraint, leading to a rapid cost increase that drove the utilization below the constraint by the third iteration. The utilization fluctuated above and below the constraint four more times in the remaining iterations as the cost corresponding to the constraint and other costs evolved. Subsequent utilization fluctuations diminish in size relative to the constraint, and the utilization in the 27th iteration is within just a few percent of the constraint. Such utilization fluctuations are not surprising because the buying model is a linear program (LP). A small cost change across iterations can incentivize the LP to move relatively large order quantities between inbound warehouses for individual products. This non-smoothness is diminished by the large number of products being ordered, but its presence persists even in aggregate weekly arrivals, making cost search more challenging.

Finally, Figure 5 depicts the unconstrained and constrained utilization at a sample of 20 warehouses in the third simulated week. The constrained utilization is from the simulation in the 27th iteration with the costs that were promoted to production buying. Unconstrained utilization exceeds the constraint at roughly half of the warehouse-weeks, and vice versa. Constrained utilization is closer to the constraint at each warehouse-week, with utilization decreases at initially over-utilized warehouse-weeks, and vice versa.



(a) First example warehouse-week.

(b) Second example warehouse-week.

Figure 4: Utilization and costs across iterations.

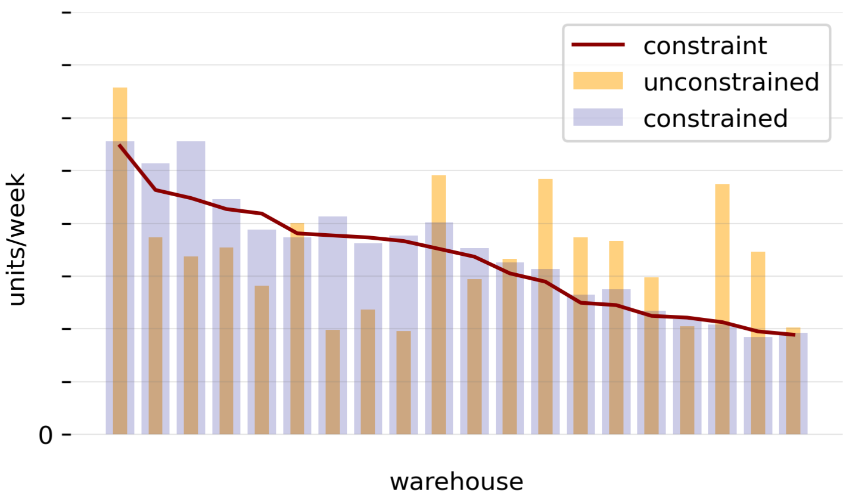


Figure 5: Unconstrained and constrained simulated utilization relative to constraints, for a subset of warehouses and at a particular simulated week.

3.2 Buying quality

Our business objective is to improve inbound warehouse-week capacity adherence with minimal degradation to buying quality (*e.g.*, supply chain costs and customer service metrics). To quantify the impact on buying quality, we compare the preferred node availability metric with and without the capacity costs. For weeks 1–4, the promoted capacity costs lead to a 3.74% reduction in the simulated preferred node availability metric. To put a 3.74% reduction in context, it is smaller than the median month over month absolute percent change in this metric in 2024 (4.4%).

4 CONCLUSIONS

In this paper, we propose a simulation-based capacity cost search approach for improving adherence to inbound node arrival capacity constraints in an online retailer supply chain. The proposed approach is based upon a dual decomposition framework, ensuring that individual product buying decisions remain independent. In each iteration, parallel discrete-event simulations of individual products provide an estimate of aggregate capacity utilization. Simulated constraint violations are used as a sub-gradient in a second-order gradient ascent algorithm to find updated capacity costs. The approach is tractable and can scale to scenarios with as many products and warehouses as in the largest online retailer supply chain networks. We demonstrate in a real-world scenario from the Amazon.com US supply chain that the approach can improve capacity adherence by 63% while degrading a buying quality metric by less than 4%.

ACKNOWLEDGMENTS

We appreciate our colleague Yunan Liu's thoughtful suggestions for the structure of this document and his thorough feedback on earlier drafts.

REFERENCES

Boyd, S., and L. Vandenberghe. 2004. *Convex Optimization*. Cambridge, UK: Cambridge University Press.

Boyd, S., L. Xiao, A. Mutapcic, and J. Mattingley. 2015. "Notes on Decomposition Methods". Notes for Stanford University EE364B. https://web.stanford.edu/class/ee364b/lectures/decomposition_notes.pdf, accessed 27th August, 2025.

Eisenach, C., U. Ghai, D. Madeka, K. Torkkola, D. Foster, and S. Kakade. 2024. "Neural Coordination and Capacity Control for Inventory Management". *arXiv preprint*. *arXiv:2410.02817*.

Gosavi, A. 2015. *Simulation-Based Optimization*. New York: Springer.

Johnson, S. G. 2019. "Quasi-Newton Optimization: Origin of the BFGS update". Notes for MIT 18.335. https://ocw.mit.edu/courses/18-335j-introduction-to-numerical-methods-spring-2019/1b52b607c223977b35ba1d826e4d8df1_MIT18_335JS19_lec30.pdf, accessed 27th August, 2025.

Jung, J. Y., G. Glau, J. F. Pekny, G. V. Reklaitis, and D. Eversdyk. 2004. "A Simulation Based Optimization Approach to Supply Chain Management Under Demand Uncertainty". *Computers & Chemical Engineering* 28:2087–2106.

Madeka, D., K. Torkkola, C. Eisenach, A. Luo, D. P. Foster, and S. M. Kakade. 2022. "Deep Inventory Management". *arXiv preprint*. *arXiv:2210.03137*.

Mele, F. D., G. Guillén, A. Espuña, and L. Puigjaner. 2006. "A Simulation-Based Optimization Framework for Parameter Optimization of Supply-Chain Networks". *Industrial and Engineering Chemistry Research* 45:3133–3148.

Schwartz, J. D., W. Wang, and D. E. Rivera. 2006. "Simulation-Based Optimization of Process Control Policies for Inventory Management in Supply Chains". *Automatica* 42:1311–1320.

AUTHOR BIOGRAPHIES

MICHAEL BLOEM a Senior Applied Scientist in Supply Chain Optimization Technologies at Amazon. In this role, deploys mathematical optimization, simulation, and machine learning to solve cross-functional problems at the intersections of buying, placement, and capacity. He received his B.S.E. degree with majors in electrical and computer engineering and economics from Calvin College in 2004 and his M.S. degree in electrical and computer engineering from the University of Illinois at Urbana-Champaign in 2007. In 2015, Michael received his PhD in operations research from the Department of Management Science and Engineering at Stanford University.

SONG (SAM) ZHOU is an Applied Scientist in Supply Chain Optimization Technologies at Amazon. He specializes in optimization algorithms and their applications to controlling inbound decisions in the supply chain. He received his B.S. in Mathematics from Tsinghua University, China and PhD in Operations Research from Cornell University, US.

KAI HE is a Senior Applied Scientist in Supply Chain Optimization Technologies at Amazon. In this role, his expertise is to drive better automatic inventory procurement decisions for retail business. His work is closely related to inbound capacity management, vendor and customer experiences improvement and supply chain cost reduction. He holds PhD in industrial engineering from the Department of Industrial Engineering at University of Pittsburgh.

ZHUNYOU (JONY) HUA is a Senior Business Intelligence Engineer in Supply Chain Optimization Technologies at Amazon. In this role, his work focuses on inventory placement and capacity analytics related initiatives, such as analytics strategy, data engineering and predictive modeling. He received his B.S. degree with majors in mathematics and economics from Nanyang Technological University, Singapore in 2013 and his M.Eng. in operations research from Cornell University in 2015.

YAN XIA is a Senior Manager, Applied Science in Mechatronics & Sustainable Packaging at Amazon. Yan's team drives the science behind mission-critical automation and packaging solutions that impact billions of customer shipments annually across Amazon's worldwide marketplaces. He received his B.S. degree in Math in 2010 and his M.S. and Ph.D. degree in Industrial Engineering from the University at Buffalo (SUNY) in 2015.