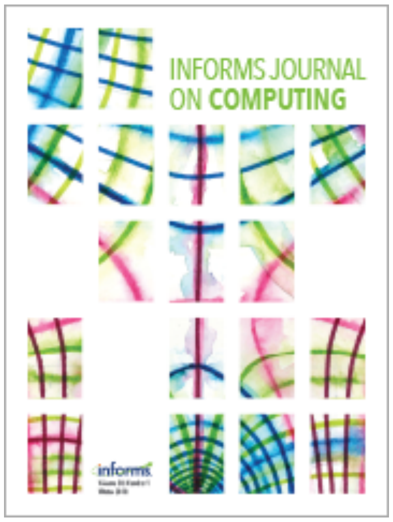


INFORMS Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Large-Scale Inventory Optimization: A Recurrent Neural Networks-Inspired Simulation Approach

Tan Wang, L. Jeff Hong

informs
https://pubsonline.informs.org/journal/ijoc

INFORMS JOURNAL ON COMPUTING
Vol. 35, No. 1, January–February 2023, pp. 196–215
ISSN 1091-9856 (print), ISSN 1526-5528 (online)

Large-Scale Inventory Optimization: A Recurrent Neural Networks–Inspired Simulation Approach

Tan Wang,[a] **L. Jeff Hong**[b,*]

[a] School of Data Science, Fudan University, Shanghai 200433, China; [b] School of Management and School of Data Science, Fudan University, Shanghai 200433, China

*Corresponding author

Contact: dwang19@fudan.edu.cn, https://orcid.org/0000-0003-2213-0933 (TW); hong_liu@fudan.edu.cn, https://orcid.org/0000-0001-7011-4001 (LJH)

Received: January 15, 2022
Revised: July 10, 2022; September 28, 2022
Accepted: October 3, 2022
Published Online in Articles in Advance: December 6, 2022

https://doi.org/10.1287/ijoc.2022.1253

Copyright: © 2022 INFORMS

Abstract. Many large-scale production networks include thousands of types of final products and tens to hundreds of thousands of types of raw materials and intermediate products. These networks face complicated inventory management decisions, which are often too complicated for inventory models and too large for simulation models. In this paper, by combining efficient computational tools of recurrent neural networks (RNNs) and the structural information of production networks, we propose an RNN-inspired simulation approach that may be thousands of times faster than the existing simulation approach and is capable of solving large-scale inventory optimization problems in a reasonable amount of time.

History: Accepted by Bruno Tuffin, Area Editor for Simulation.
Funding: This work was supported by the National Natural Science Foundation of China [Grant 72091211].

Keywords: inventory management • recurrent neural network • gradient estimation • simulation optimization

1. Introduction

Inventory is one of the most important tools for mitigating uncertainty in enterprise production and operations. Effective management of inventory can offer tremendous potential for increasing manufacturing efficiency and reducing operational cost. Therefore, inventory management has always been a core part of modern enterprise supply chain management and has long captured the interests from academics and industrial practitioners.

Nowadays, many large companies produce and offer a wide range of brands or products, for example, Coco-Cola and Unilever in fast-moving consumer goods industry, and Samsung and Xiaomi in consumer electronics industry, partly due to market differentiation and partly due to economies of scale. As a result, the bills of materials (BOM) of these companies often include tens of thousands to even hundreds of thousands of nodes, which represent raw materials, subassemblies, and final products, and show complex network topologies, which represent the production relationship of all the nodes. These complex production systems also impose tremendous challenges and opportunities to inventory management. First, there are decisions that need to be made for every node of the BOM, such as whether to keep inventory and how much to keep. Second, the complex network topology means a lot of sharing and pairing among the nodes in the production process and, to achieve the overall efficiency, one needs to take a holistic view toward all inventory decisions. Third, the overall inventory costs of these production systems are often very high, and therefore, any savings may be quite significant. For instance, in our consulting experience that motivated this study, we worked with a global manufacturer whose BOM has more than 500,000 nodes and more than 4,000,000 links (a part of the BOM is shown in Figure 1) and whose inventory at the time was worth about \$3 billion; thus, even a 1% decrease is significant. With the ongoing COVID-19 pandemic and international trade frictions, the global supply chains are facing increasingly more risks; therefore, how to manage these large-scale production and inventory systems becomes not only challenging but also extremely important.

From the perspective of research, the aforementioned problem belongs to multiechelon (or multistage) inventory optimization problem. Generally speaking, nodes in a multiechelon system represent locations or processes of a supply chain, where a final product must undergo during manufacturing, assembly, or distribution. We focus on production networks in this paper. Taking the well-studied Kodak digital camera supply chain network shown in Figure 2 (Snyder and Shen 2019) as an example, each node represents a certain

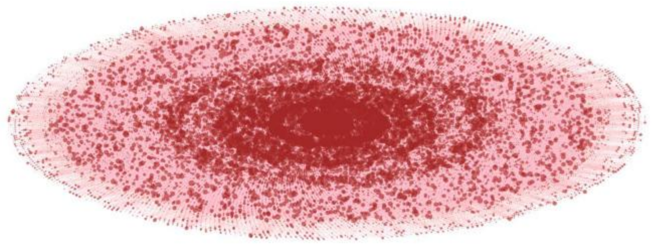


Figure 1. (Color online) BOM Network of a Large-Scale Inventory System

production process for the final product. Holding inventory at a node means holding a product that has finished production at the node, and holding no inventory at a node means pulling items from upstream nodes and producing whenever there are demands at the node.

In the existing literature, two types of models have been developed to handle the multiechelon inventory optimization problem: stochastic-service (SS) models and guaranteed-service (GS) models. The two types of models differ in replenishment mechanism between stages. SS models assume that the delivery or service time can vary based on the material availability at the supply stage, whereas GS models assume that each stage can quote a delivery or service time that can be always satisfied (Graves and Willems 2003). Regarding SS models, the existing work mainly focuses on serial systems (Clark and Scarf 1960) or assembly systems (Rosling 1989, Chen et al. 2014), which is in general not suitable for the large-scale problems considered in this paper.

GS models were first proposed by Simpson (1958). Graves and Willems (2000) develop a dynamic programming algorithm for supply chains that can be modeled as spanning trees. Lesnaia et al. (2005) show that the GS models for general acyclic networks are NP-hard problems and difficult to solve efficiently, and Humair and Willems (2011) propose two faster heuristics to solve the problems approximately. However, GS models assume that it is possible to establish meaningful deterministic upper bounds on the stochastic demands, which may be quite difficult for the large-scale problems considered in this paper and may lead to either

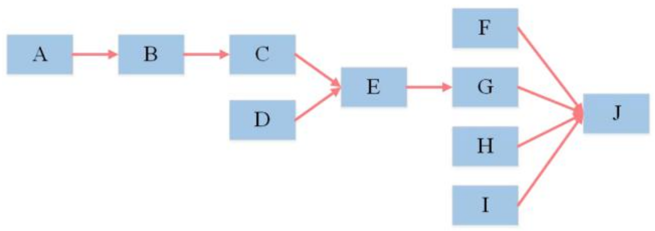


Figure 2. (Color online) Kodak Digital Camera Supply Chain Network

uncontrolled fill rates or high inventory costs. Besides, the original GS model assumes a deterministic lead time for each node in the supply chain. Humair et al. (2013) extend the model to allow random lead times but also bring a complication that the objective function is no longer concave, which complicates the optimization process. One advantage of GS models is that they tend to hold inventory only at a small number of strategic nodes of the BOM. This property is particularly appealing for large-scale problems because it drastically reduces the difficulty of managing inventories compared with the situation where all nodes hold inventory. In this paper, although we do not use GS models, we want to develop algorithms that have this property.

The simulation approach is another approach to solving inventory problems, and it allows stochastic demands and complex BOM structures. Its general framework was laid out by Glasserman and Tayur (1995). Basically, the simulation approach first builds a simulation model that simulates the evolution of the production and inventory system with randomly generated demands and given inventory policies and estimates the average cost of the system by running the simulation model for multiple replications. To minimize the inventory cost, the simulation approach computes the sample-path gradient with respect to the inventory decisions and applies the stochastic approximation (SA) algorithm (Robbins and Monro (1951), also known as stochastic gradient descent (SGD) algorithm) to optimize.

Although the simulation approach is simple conceptually, and it is capable of modeling complex production-inventory systems, it has two drawbacks that prevent it from solving the large-scale problems considered in this paper. *First and foremost, it is often too slow for large-scale problems.* For instance, for a BOM network with 5,000 nodes, we observe that the traditional simulation approach takes about an hour just to run a single simulation replication with 100 periods (Table 1) and many hours to compute a sample-path gradient. Furthermore, we show that the computational complexities of the simulation and the gradient calculation are of $O(Tn^2)$ and $O(Tn^3)$, respectively, where T is the number of periods, and n is the number of nodes in the BOM. Therefore, it is unlikely that the simulation approach may be used to solve problems with tens of thousands to hundreds of thousands of nodes. *Second, it is not clear how to use the simulation approach to find solutions that keep inventory only at a small number of nodes.* By applying the SA algorithm directly, we observe that the solution typically keeps inventory for almost all nodes, and it is very difficult to implement in practice.

To overcome the first drawback of the traditional simulation approach, we notice that the simulation of a production-inventory system is very similar to the forward pass of a recurrent neural network (RNN), which is widely used in speech recognition and language modeling, and the simulation optimization of the inventory

Table 1. Run Time of Simulation for a Single Replication

Number of nodes	Traditional algorithm	Algorithm 1 dense	Algorithm 1 sparse
1,000	1.82 min	0.25 s	0.17 s
5,000	42.70 min	1.99 s	0.59 s
10,000	169.39 min	6.24 s	1.14 s
50,000	—	125.04 s	5.45 s
100,000	—	488.41 s	10.92 s
500,000	—	/	71.18 s

policy is similar to the training of an RNN. This analogy is critical to the speedup of the simulation approach, because RNNs are sometimes bigger than the production-inventory systems that we consider and the computational tools that are used to model and to train RNNs; for example, the back-propagation (BP) algorithm for computing sample-path gradient (Werbos 1990) and the various SGD algorithms for optimization may be adapted to the simulation and optimization of the inventory decisions. Furthermore, we discover that, different from RNNs, BOM networks are typically quite sparse, which allows us to use sparse matrices to further speed up the simulation and optimization. We show that, by combining the tools of RNNs and the use of sparse matrices, we improve the computational complexities of simulation and gradient computation from $O(Tn^2)$ and $O(Tn^3)$ to $O(Tn)$ and $O(Tn)$, respectively, and reduce the computational times in the order of thousands to tens of thousands.

To overcome the second drawback of the traditional simulation approach, we propose to use L_1-regularization to force the solution of the optimization problem to be sparse, thus only allowing a small number of nodes to have nonzero base-stock levels. Although regularization is commonly used in training of RNNs to avoid overfitting (Goodfellow et al. 2016), it is used in our problems for completely different purposes. However, the similarity between the training of RNNs and the inventory optimization allows us to take full advantage of the subgradient SGD algorithm, designed for training L_1-regularized deep neural networks, to solve inventory optimization problems. Furthermore, we show that fast iterative shrinkage-thresholding (FISTA) algorithm (Beck and Teboulle 2009), which uses the Nesterov accelerated gradient method and the proximal method to achieve faster convergence, may be applied to our problem as well and achieve numerical performance that is in general slightly better than the subgradient SGD algorithm. We also show that, because of the difference between the training of RNNs and the inventory optimization, a reoptimization step may be added to our problem to further improve the performance of the inventory decisions while keeping the inventory only at a small number of nodes.

It is worth noting that we only adopt the underlying ideas and computational tools of RNNs to solve inventory optimization problems, and we do not train any RNNs in our algorithm. Meanwhile, different from our work, there are some recent works that train neural networks and use them directly in inventory optimization. For example, Pirhooshyaran and Snyder (2020) present a framework that uses deep neural network agents to directly determine the order-up-to levels between any adjacent pair of nodes in the supply chain, Kosasih and Brintrup (2021) investigate reinforcement-learning methods to optimize safety stocks in a linear chain of independent agents, and Oroojlooyjadid et al. (2022) propose a deep reinforcement-learning algorithm to play the famous beer game. However, those papers only consider small-scale problems, and it is not clear how to extend their methods to solve large-scale problems considered in this paper.

After overcoming the two drawbacks of the traditional simulation approach, the new simulation approach is capable of solving large-scale inventory optimization problems with tens of thousands to hundreds of thousands of nodes in a reasonable amount of time while keeping the inventory only at a small number of nodes. Furthermore, we test different methods to implement the algorithms, including the use of TensorFlow for simulation and for automatic calculation of the sample-path gradients, the use of parallel CPU processors and the use of GPUs, and obtain a wide range of insights that are useful in implementing the algorithms in different scenarios.

In summary, this paper makes the following contributions to large-scale inventory optimization and simulation modeling and optimization:

• It finds that the modeling of a complex production-inventory system is analogous to an RNN, and its simulation optimization is analogous to the training of an RNN. This analogy opens the door so that we can take advantages of efficient computational tools of deep learning to solve large-scale inventory problems. In addition, we want to point out that this analogy goes beyond production-inventory systems. It may be applied to general periodically reviewed dynamic systems and, thus, may lead to speedup of the simulation of general dynamic systems.

• It demonstrates that the BP algorithm provides the same sample-path gradient as the infinitesimal perturbation analysis (IPA; Ho et al. 1983). However, it is computationally more efficient than the IPA when the gradient is of a high dimension. It also explores the possibility of using TensorFlow or other modern computational tools to compute the sample-path gradient automatically without deriving it explicitly.

• It shows that the sparsity of the BOM network may be used to significantly reduce the computational complexities of the simulation and gradient calculation, allowing the algorithm to reduce the computation time by orders of magnitude.

• It uses L_1-regularization to keep inventory only at a small number of nodes, thus reducing the difficulty of inventory management, and proposes algorithms to efficiently solve the L_1-regularized simulation optimization problems.

The rest of this paper is organized as follows. In Section 2, we provide the problem statement and simulation-optimization formulation of the problem. In Section 3, we briefly introduce the typical structure of RNNs and make the analogy between the modeling and training of RNNs and the simulation and optimization of large-scale inventory systems. Inspired by RNNs, in Sections 4 and 5, we discuss how to use the structures of inventory systems to develop efficient simulation and optimization algorithms, followed by a comprehensive numerical study in Section 6. The paper is concluded in Section 7. A package containing all source code and data of the experiments is available at this paper's GitHub repository (https://github.com/wangruohai/LSIO-RNNISA).

2. Problem Formulation

We consider a large-scale production system with a general BOM network structure, where the demands and inventories are reviewed periodically (e.g., every day). We differentiate the items of the BOM into two categories: procurement items and production items. Procurement items are the raw materials that sit at the bottom of the BOM, and they are procured from outside. Production items are the intermediate items (a.k.a. subassemblies) or final products, and they are produced by the production system. We allow all production items to have outside demands because some intermediate items are needed for maintenance or service. In our model, the outside demands, the production time, and the procurement times may be stochastic. However, we assume that they follow known distributions (or, at least, may be generated through a simulation algorithm) that take only integer values.

We also assume that there are no capacity constraints on the production. This is a common assumption in the inventory management literature (Clark and Scarf 1960, Rosling 1989, Graves and Willems 2000). For large-scale production-inventory systems, considering capacity constraints will turn the inventory optimization problem into a complex production planning/scheduling problem (Hall and Liu 2010), which requires separate optimization tools and typically only considers deterministic/known demands. In fact, one may consider the uncapacitated inventory optimization problem studied in this paper as an add-on to the production plans to hedge the randomness in the demands.

Furthermore, if a component is used in multiple downstream items, we assume that the inventory of the component is allocated proportionally to all downstream demands so that they all have the same fill rate, when the inventory is insufficient to satisfy all demands. Notice that, for any item, the shortage is a low probability event. Then, the use of different allocation rules may not have significant impact on the performance of the system. The proportional allocation rule is simple and easy to implement, it avoids solving complex allocation optimization problems, and it maintains sample-path continuity that is crucial to the calculation of sample-path gradients.

In addition, we suppose that stationary (installation) base-stock policies are used for all nodes of the BOM, and our goal is to find the optimal base-stock levels. The base-stock policies may not be the optimal policies for our problem. However, they are widely used in practice because of its simplicity (Glasserman and Tayur 1994, Gallego and Zipkin 1999), and they are known to be optimal for serial systems (Clark and Scarf 1960) and assembly systems (Rosling 1989). If one is interested in finding the optimal policy, it becomes a reinforcement-learning problem and we refer to Oroojlooyjadid et al. (2022) and references therein. However, the reinforcement-learning algorithms proposed in the current inventory literature typically only solve small-scale problems, and more research work needs to be done to develop algorithms that are capable of solving large-scale problems. The recent development of deep reinforcement learning may be a direction to go (Arulkumaran et al. 2017).

2.1. Simulation-Based Approach

As the BOM network is complex and there are external demands for possibly all items in the BOM, the simulation algorithm also includes a lot of details. To clearly present the algorithm, we break it into the following four major steps based on the sequence of events within each period.

Step 1. Calculating the inventory positions, observing outside demands and placing the replenishment orders at the beginning of the period.

Step 2. Receiving finished orders, fulfilling outside demands, and updating the on-hand inventories and backlogs at the beginning of the period.

Step 3. Producing based on the replenishment orders and the available materials and updating the on-hand inventories at the end of the period.

Step 4. Calculating the cost at the end of the period.

Now we present the algorithm based on these four steps.

2.1.1. Step 1.

Let $IP_{t,i}$ denote the inventory position of item i at period t. It is a measure used under a base-stock policy for making ordering decisions, which equals the on-hand inventory plus the on-order quantity (i.e., the amount that is ordered but not yet received through either procurement or production) minus the backorders (i.e., the demands that have occurred but have not been fulfilled) (Snyder and Shen 2019). Let $O_{t,i}$, $D^{in}_{t,i}$ and $D^{out}_{t,i}$ denote the order quantity, the internal demand, and the outside demand of item i at period t, respectively. We show in the Appendix that the inventory position can be calculated using the following recursive formula:

$$IP_{t,i} = IP_{t-1,i} + O_{t-1,i} - D^{in}_{t-1,i} - D^{out}_{t-1,i}, \tag{1}$$

where $IP_{0,i}$ is the initial inventory position of item i at the beginning of time 0.

Once the inventory position $IP_{t,i}$ is calculated, a replenishment order $O_{t,i}$ is placed to bring $IP_{t,i} - D^{out}_{t,i} - D^{in}_{t,i}$ up to the base-stock level S_i, that is,

$$O_{t,i} = -\min\{0, IP_{t,i} - D^{out}_{t,i} - D^{in}_{t,i} - S_i\}.$$

Although the outside demands $D^{out}_{t,i}$ are clearly observed, the internal demands $D^{in}_{t,i}$ depend on the replenishment orders of downstream items. Therefore, the replenishment order of item i cannot be placed until orders of all its downstream items are placed. Let a_{ij} denote the number of units of the component i required to produce a unit of item j. Then, the internal demand of item i by its downstream item j observed at period t is $a_{ij}O_{t,j}$, and the total internal demand is $D^{in}_{t,i} = \sum_j a_{ij}O_{t,j}$. Furthermore, these calculations must start from the final products, which are the most downstream items and have no internal demands, and gradually move to the upstream items.

2.1.2. Step 2.

Let $P_{t,i}$ denote the arrived replenishment orders of item i at the beginning of period t. They may be procured from external suppliers if the item is a raw material item, or they may be produced within the production system if the item is a production item (i.e., a subassembly or a final product). Let $I_{t,i}$ and $B^{out}_{t,i}$ denote the on-hand inventory and backlogged outside demand at the *end* of period t, respectively. We assume the outside demands have higher priority and are fulfilled at the beginning of the period when the items are available. Then, the on-hand inventory at the beginning of period t after fulfilling the outside demand, denoted by $I^0_{t,i}$, is updated by

$$I^0_{t,i} = \max\{0, I_{t-1,i} + P_{t,i} - B^{out}_{t-1,i} - D^{out}_{t,i}\}, \tag{2}$$

and the unfilled part is backlogged with

$$B^{out}_{t,i} = -\min\{0, I_{t-1,i} + P_{t,i} - B^{out}_{t-1,i} - D^{out}_{t,i}\}. \tag{3}$$

Let $\ell_{t,i}$ denote the (random) lead time of item i ordered at time t. It may be the procurement lead time if the item is procured from outside or the production lead time if the item is produced within the production system. For a raw material item (say i), the procurement arrived at period $t + \ell_{t,i}$ is the order at period t, that is, $P_{t+\ell_{t,i},i} = O_{t,i}$. The production items are much more complicated, because it is determined not only by the lead time but also the availability of upstream items. We handle them in Step 3.

2.1.3. Step 3.

For a production item, orders are manufacturing commands, and they are processed immediately if the upstream components are all available. We do not assume that the BOM has a tree structure as commonly assumed in the literature (Rosling 1989, Graves and Willems 2000). Then, some components may be shared by multiple downstream items. When the inventories of these components are insufficient, the proportional allocation rule is adopted. Let $r_{t,i}$ denote the fill rate of item i at period t. Then,

$$r_{t,i} = \min\left\{\frac{I^0_{t,i}}{\sum_j a_{ij}(O_{t,j} + O^b_{t-1,j})}, 1\right\},$$

where j denotes an immediate downstream item of item i, a_{ij} denotes the number of units of the component i required to produce a unit of item j, and $O^b_{t-1,j}$ is the backlogged production orders of item j at period $t-1$ due to the unavailability of at least one of its upstream items.

Let $M_{t,i}$ denote the production quantity of item i at period t. With the allocated upstream inventories and newly assigned production orders, we can now calculate the production quantity $M_{t,i}$ and the backlogged production order $O^b_{t,i}$ as follows:

$$M_{t,i} = \min_{k, a_{k,i}\neq 0}\{r_{t,k}\}(O_{t,i} + O^b_{t-1,i}), \tag{4}$$

$$O^b_{t,i} = O_{t,i} + O^b_{t-1,i} - M_{t,i}, \tag{5}$$

where the minimum operator used in Equation (4) takes into consideration of the lowest availability of all components that are necessary in producing item i. Notice that $M_{t,i}$ will become finished goods (or arrived order) after a production lead time of $\ell_{t,i}$ periods, that is,

$$P_{t+\ell_{t,i},i} = M_{t,i}.$$

Furthermore, for an item i, the total internal demand by its downstream item j at period t is $a_{ij}(O_{t,j} + O^b_{t-1,j})$. As some production orders may be backlogged due to insufficient supply, the actual amount to be deducted from the on-hand inventory of item i is $a_{ij}M_{t,j}$. Therefore, the on-hand inventory of each item i at the end of each period t is

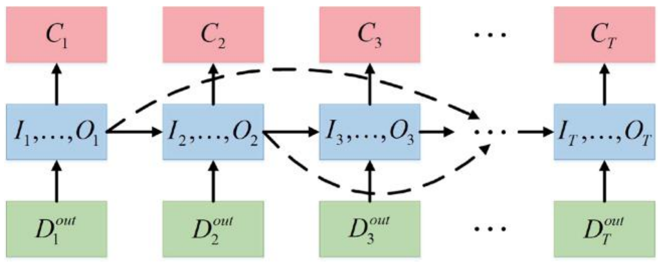
$$I_{t,i} = I^0_{t,i} - \sum_j a_{ij}M_{t,j}. \tag{6}$$

2.1.4. Step 4.

The cost of each period is composed of the inventory holding cost and the stockout penalty cost, that is,

$$C_t = \sum_i (h_i I_{t,i} + p_i B^{out}_{t,i}),$$

Figure 3. (Color online) Schematic Diagram of Inventory Simulation



where h_i and p_i denote the unit holding cost and unit stockout penalty cost of item i, respectively. We only consider the stockout penalty cost for backlogged outside demands.

The aforementioned Steps 1–4 form an iteration (i.e., a period) of the inventory simulation algorithm, and it runs for T periods to calculate (an observation of) the cumulative cost. We illustrate the algorithm in Figure 3, and we will show that the structure of the algorithm is similar to that of an RNN.

The objective of the simulation-based approach is to find the optimal base-stock level that minimizes the expected cumulative cost over T periods, that is,

$$\min_{S} \quad \mathbb{E}\left[\sum_{t=1}^{T} C_t(S)\right]. \tag{7}$$

Here we write the cost of a period t as $C_t(S)$ just to emphasize that the cost is a random function of the base-stock levels $S = (S_1, \ldots, S_n)$.

Because the simulation model itself is very complicated and there exists no approach to deriving a closed-form expression of the expected cumulative cost, we suggest to solve Problem (7) using the SA approach, which was also adopted by Glasserman and Tayur (1995) to solve a much smaller scale inventory optimization problem. The key to use the SA approach is to calculate the sample-path gradient of $\sum_{t=1}^{T} C_t(S)$ with respect to S, which can be solved by using the IPA and will be discussed in details in Section 5.1.1.

2.2. Challenges in Large-Scale Problems

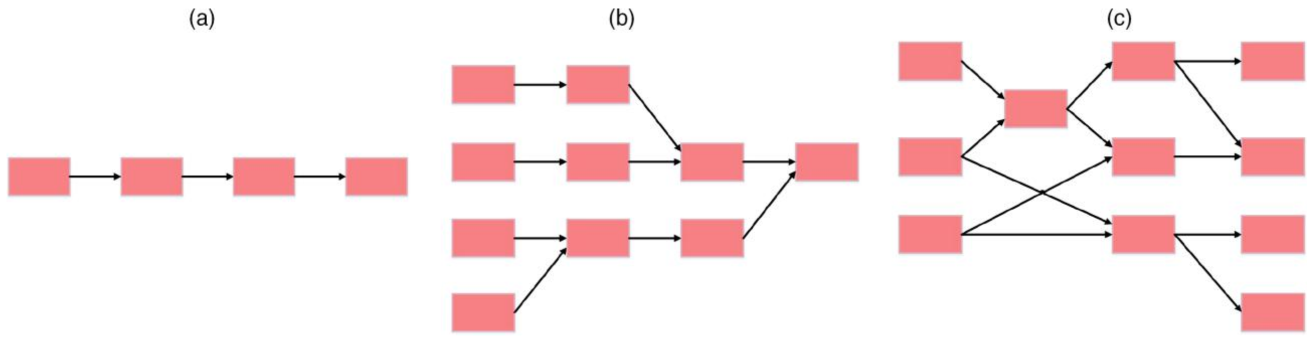
Previous studies on inventory simulation mainly focus on serial systems (Figure 4(a)) or assembly systems (Figure 4(b)), and the simulation-based approach introduced in Section 2.1 is capable of handling inventory systems with general topology (Figure 4(c)). However, this generality also brings tremendous challenges in computation, especially for large-scale problems that are considered in this paper.

In serial systems, each stage has at most one predecessor and at most one successor, and in assembly systems, each stage has at most one successor. Therefore, the computation needed to simulate and optimize these systems does not increase dramatically as the size of the problem increases. In general systems, however, the complex network structure implies that, for every node of the BOM, any other node may be its predecessors or successors. This structure significantly complicates the simulation process and results in a drastic increase of the simulation time when the problem is of large scale.

For instance, according to our testing experience, the simulation of a single replication of a system with only 5,000 nodes for 100 periods may take about an hour, and the calculation of its sample-path gradient may take more than 10 hours. Moreover, although multiple simulation replications are easy to parallel, the parallelization of a single replication of the simulation run or the parallelization of a single gradient calculation is not straight-forward and may need careful redesign of the simulation algorithm. Therefore, the first challenge of this paper is *how to reduce the computational time (including both simulation and optimization) to a manageable level for large-scale problems.*

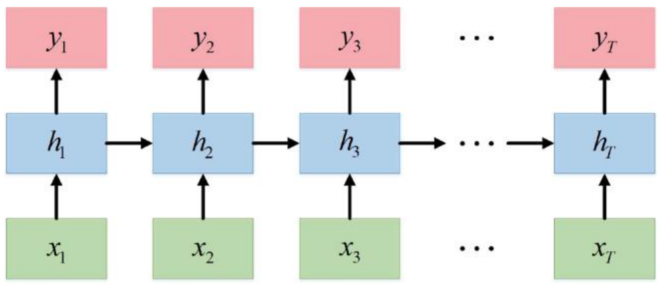
Furthermore, when the system is large and with tens of thousands or even more items, keeping inventory for all items may not be practical. However, directly applying the SA algorithms to solve Problem (7) may result in

Figure 4. (Color online) Network Topologies (Snyder and Shen 2019)



Notes. (a) Serial system. (b) Assembly system. (c) General system.

Figure 5. (Color online) Typical RNN Architecture



solutions that are nonzero for almost all items and thus very difficult to implement in practice. One may add a cardinality constraint on the number of nonzero base-stock levels, but it is very difficult to handle in the context of simulation optimization. Therefore, the second challenge of this paper is *how to find solutions that keeps inventory only at a small number of nodes.*

3. RNNs

RNN is a class of neural networks that allow previous outputs to be used as inputs while having hidden states. This enables it to exhibit temporal dynamic behaviors and to model time series data. RNN models have achieved the-state-of-the-art performance on tasks that include speech recognition (Miao et al. 2015), language modeling (Mikolov et al. 2010), and image captioning (You et al. 2016). The scale of deep RNN models can be very large; for example, the model of Miao et al. (2015) contains 8.5 million parameters.

3.1. RNN Architecture and Training

As shown in Figure 5, an RNN often has a chain-like architecture, where in each period, the middle box is a neural network that takes x_t and h_{t-1} as the inputs and outputs h_t and y_t. Although many variations exist, a common implementation of an RNN can be described by the following updating equations (Goodfellow et al. 2016):

$$h_t = f(Wh_{t-1} + Ux_t + b), \tag{8}$$

$$y_t = g(Vh_t + c), \tag{9}$$

where h_t denotes the hidden states at time $t, f(\cdot)$, $g(\cdot)$ are complex nonlinear activation functions represented by neural networks and U, V, W, b, c are the parameters of the neural networks.

Suppose that we have a data set of N samples, denoted by (x_t^n, y_t^n), $t = 1, 2, \ldots, T$ and $n = 1, 2, \ldots, N$. Given the network parameters, with an initial value of the hidden state h_1 and the outside inputs $\{x_1^n, x_2^n, \ldots, x_T^n\}$, we can use the RNN model to generate a time series of outputs, denoted by $\{\hat{y}_1^n, \hat{y}_2^n, \ldots, \hat{y}_T^n\}$. If the neural network perfectly matches the data generating process, then $\hat{y}_t^n = y_t^n$. We define a loss function $L(y_t^n, \hat{y}_t^n)$ to capture the difference between $\hat{y}_t^n$ and y_t^n. Then, to train an RNN is to find the network parameters that solves the following optimization problem:

$$\min \ \frac{1}{N}\sum_{n=1}^{N} L_T^n, \quad \text{where} \quad L_T^n = \sum_{t=1}^{T} L(y_t^n, \hat{y}_t^n). \tag{10}$$

To solve Problem (10), SGD algorithms are typically used. SGD algorithms are originated from SA algorithms, and they have been adapted to solve very high-dimensional problems (Bottou et al. 2018). To use SGD algorithms, one needs to calculate the gradient of L_T^n with respect to the network parameters. This is typically done through BP algorithm. The BP algorithm for calculating the gradient of neural networks is a celebrated result in the history of deep learning (Rumelhart et al. 1986). It is particularly efficient if the network parameters are of high dimension, and it may be implemented automatically (Baydin et al. 2018). The particular version of the BP algorithm for RNNs is called back-propagation through time (Werbos 1990), and it is capable of handling neural networks with a few million parameters (Miao et al. 2015).

3.2. Similarities Between RNN and Inventory Optimization

There are three critical similarities between the training of RNN and the simulation approach introduced in Section 2. First, both models have a complex network that evolves over time, and both have external and internal inputs in each period. In the case of RNNs, the network is a neural network, and the external and internal inputs are x_t and h_{t-1}, respectively, as illustrated in Figure 5. In the case of inventory model, the network is the production network implied by the BOM, and the external and internal inputs are D_t^{out} and $I_{t-1}, \ldots, O_{t-1}$, respectively, as illustrated in Figure 3. Because the lead times may not always be one period, the inventory model may have links that go beyond the immediate next period, as illustrated in Figure 3. Therefore, the simulation model is more complex than the RNN model in this aspect.

Second, both problems have very similar optimization formulations. The objective function of the inventory optimization problem is an expectation, which may be approximated by the sample average of N samples. Then, the problem becomes

$$\min \ \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T} C_t^n(S),$$

which has the exact same form as the RNN training problem (10). Furthermore, the decision variables of the inventory optimization problem are the base-stock levels. They can also be considered as the parameters of the production network. Therefore, the decision variables of both problems are network parameters that do not change over time.

Third, both approaches calculate sample-path gradient and optimize using gradient-based algorithms. The training of RNN uses the BP algorithm for gradient calculation and the SGD algorithm for optimization, whereas the inventory optimization problem uses the IPA algorithm for gradient calculation and the SA algorithm for optimization.

Given all these similarities between the two problems, it is natural to ask the following. "How is the training of RNN capable of solving problems with millions of decision variables?" "What can be learned to solve the large-scale inventory optimization problems?" These are the questions that we answer in next sections.

4. Simulation Model

RNN and other neural network models are typically represented in terms of tensors. The concept of a tensor is a generalization of vectors and matrices and can be easily understood as a multidimensional array (Goodfellow et al. 2016). Many computational tools, such as TensorFlow and PyTorch, have been developed for fast tensor operations, and they can easily leverage on the parallel computing capability provided by multicore CPUs and many-core GPUs to speed up the calculations. Indeed, tensors are so important that Google even named its machine learning library as "TensorFlow." Therefore, to achieve a significant speedup of the simulation model for large-scale inventory problems, a critical step that we learned from the success of RNN is the tensorization of the simulation model.

4.1. Matrix Representation of BOM

A BOM is a list of raw materials, subassemblies and their quantities, and manufacturing relations needed to manufacture the final products. Because we consider general inventory systems in this paper, the BOMs of these systems may be represented by complex directed networks. Therefore, we adopt the adjacency matrix that is commonly used in network science and graph theory to represent the BOM. The adjacency matrix of a BOM with n items is a square matrix with $n \times n$ elements, denoted by A, where its (i, j)th element a_{ij} indicates that a_{ij} units of item i are required to produce a unit of (downstream) item j. From the viewpoint of a network, it indicates that there is a directed arc from item i to item j and the weight of the arc is a_{ij} if $a_{ij} > 0$. If there is no arc, then $a_{ij} = 0$.

The adjacency matrix has $n \times n$ elements. For large-scale supply chains with 10^4 to 10^5 items, the storage of the matrices can take up a lot of memory space, and the matrix operations also need significant amount of computational power. Nevertheless, according to our observations of practical large-scale supply chains, the adjacency matrix is typically very sparse. In fact, sparsity is a common feature of large-scale complex networks in the real world (Ugander et al. 2011, Wang et al. 2012). Therefore, we use sparse matrices to represent the adjacency matrices to save memory space and to improve the computation efficiency. Specifically, the compressed sparse row format (Buluç et al. 2009) is adopted, and a SciPy.Sparse (Abbasi 2018) Python package is used in our implementation.

For a directed network with n nodes, its density is defined by

$$\rho = \frac{m}{n(n-1)},$$

where m is the actual number of edges (Albert and Barabási 2002). If the network is fully connected, the actual number of edges is $n(n-1)$. Therefore, the density is a measure of the network sparsity. Given the density ρ, the average (in) degree of the network, denoted by $\langle k \rangle$, is

$$\langle k \rangle = \frac{m}{n} = (n-1)\rho \approx n\rho.$$

In the context of a BOM network, the average (in) degree indicates the average number of upstream components needed to assemble a downstream product, which typically does not grow with the scale of the network. Therefore, it is reasonable to assume that the $\langle k \rangle$ is a constant, that is,

$$n\rho \sim \text{constant}. \tag{11}$$

This relationship is important for our understanding of the impact of the sparsity on the performance of the simulation algorithm and the inventory optimization algorithms.

4.2. Tensorization of the Simulation Model

In the simulation model presented in Section 2.1, except for the adjacency matrix, all other variables at time t may be represented by vectors with n elements, for example, the on-hand inventory vector $I_t = (I_{t,1}, \ldots, I_{t,n})$, the order quantity vector $O_t = (O_{t,1}, \ldots, O_{t,n})$, and so on. With these vectors and the adjacency matrix, the equations introduced in Section 2.1 can be rewritten in the form of vector and matrix operations. Indeed, most of these transformations are straightforward; here, we only present a few operations related to the adjacency matrix.

- **Replenishment Orders.** The internal demand of an item depends on the orders of its downstream items. It may be calculated by $D_t^{in} = O_t \times A'$, where A' denotes the transpose of A. Therefore, the update of inventory positions (Equation (1)) may be written as

$$IP_t = IP_{t-1} + O_{t-1} - O_{t-1} \times A' - D_{t-1}^{out}.$$

Furthermore, the replenishment order of an item depends on the internal demands that depend on the replenishment orders of the downstream items. Therefore, in Section 2.1, we proposed an iterative algorithm that starts from the most downstream items (i.e., final products) to calculate the replenishment orders of all items. The number of iterations depend on the number of layers of the BOM network. Suppose that there are n_L layers. Then, the longest distance between two nodes (items) is $n_L - 1$. We know that for a directed network with an adjacency matrix denoted by A, $(A^{\gamma})_{ij} > 0$ if and only if it takes γ steps from node i to node j (Wang et al. 2012). Here, A^{γ} denotes the γth power of matrix A. Thus, the number of layers can be calculated by

$$n_L = \sup\left\{ \gamma \in N, \sum_{i,j} 1_{\{(A^{\gamma})_{ij} > 0\}} > 0 \right\} + 1,$$

where $1_{\{\cdot\}}$ is an indicator function.

- **Fill Rates.** The fill rate vector of the items can be calculated by

$$r_t = \min\{I_t^0 / [(O_t + O_{t-1}^b) \times A'], 1\},$$

where the operators "/" and $\min(\cdot)$ denote the element-wise division and minimization, respectively.

Let $C_t^{sum} = \sum_{u=1}^{t} C_u$ denote the cumulative cost up to period t and let $I^{initial}$ denote the given initial on-hand inventory. The tensor representation of the simulation model is presented in Algorithm 1. The outside demand $D_{t,i}^{out}$ and the lead time $\ell_{t,i}$ of each item at each period in the algorithm may be randomly generated with given distributions. Index set of the nonzero elements for each column i of the adjacency matrix can be obtained by functions of the sparse matrix and is denoted by Ψ_i in Algorithm 1. The notation $\|\cdot\|_{-\infty}$ denotes the negative infinity norm, which means the minimum of the absolute values of all components of a vector.

In Algorithm 1, the state variables are represented by vectors (i.e., one-dimensional tensors), the BOM network is represented by the adjacency matrix (i.e., a two-dimensional tensor), and most of the computations are represented in vectors and matrix operations. This is known as tensorization, which enables us to use efficient computational tools available in software packages and significantly lowers the barrier for parallelization (Harris et al. 2020). However, there are also operations in the algorithm that cannot be easily or efficiently tensorized. For instance, as the lead times of different items are different, operations involving lead times correspond to state variables of different time periods. Thus, they cannot be efficiently tensorized and are presented as scalar operations (lines 11 and 15).

In general, whether an operation may be efficiently tensorized depends on the model assumptions, and it requires analysts' effort when developing the algorithm. In Section 2, we make some assumptions in our model. If these assumptions do not hold or if one wants to incorporate other operational details, further analysts' effort may be needed to build a tensorized algorithm.

Algorithm 1 (Tensorized Simulation Algorithm)

```
1  **Initialization:** $IP_0 = I_0 = I^{initial}, O_0 = 0, D_0^{out} = 0, B_0^{out} = 0,$
   $O_0^b = 0, C_0^{sum} = 0;$
2  **for** $t = 1$ *to* $T$ **do**
3      $IP_t \leftarrow IP_{t-1} + O_{t-1} - O_{t-1} \times A' - D_{t-1}^{out};$
4      $O_t \leftarrow -\min\{0, IP_t - D_t^{out} - S\};$
5      **for** $\varsigma = 1$ *to* $n_L - 1$ **do**
6          $IP_t^{temp} \leftarrow IP_t - D_t^{out} - O_t \times A';$
7          $O_t \leftarrow -\min\{0, IP_t^{temp} - S\};$
8      $I_t^0 \leftarrow \max\{0, I_{t-1} + P_t - B_{t-1}^{out} - D_t^{out}\};$
9      $B_t^{out} \leftarrow -\min\{0, I_{t-1} + P_t - B_{t-1}^{out} - D_t^{out}\};$
10     **for** *each raw material item i* **do**
11       $P_{t+\ell_{t,i},i} \leftarrow O_{t,i}$
12     $r_t \leftarrow \min\left\{\frac{I_t^0}{(O_t + O_{t-1}^b) \times A'}, 1\right\};$
13     **for** *each final product or subassembly item i* **do**
14         $M_{t,i} = \|r_{t,\Psi_i}\|_{-\infty};$
15         $P_{t+\ell_{t,i},i} \leftarrow M_{t,i};$
16     $O_t^b \leftarrow O_t + O_{t-1}^b - M_t;$
17     $I_t \leftarrow I_t^0 - M_t \times A';$
18     $C_t \leftarrow I_t \times h' + B_t^{out} \times p';$
19     $C_t^{sum} \leftarrow C_{t-1}^{sum} + C_t;$
```

4.3. Complexity Analysis

Computational complexity is often used to understand the efficiency of an algorithm and to compare the efficiency of different algorithms. In this paper, we also adopt this tool to compare different algorithms. However, we want to emphasize that computational complexity does not completely explain the differences of the run times of different algorithms. For instance, based on our experience, tensorization may reduce the run time of our simulation algorithm by orders of magnitude, but it does not reduce the algorithm's complexity. Nevertheless, it provides a good measure to compare algorithms when all of them are properly tensorized.

The computational complexity of the addition/multiplication of two scalars is defined as O(1). Then, the computational complexity of vector (with n entries) addition is $O(n)$, and that of the multiplication of a $1 \times n$ vector and a $n \times n$ matrix is $O(n^2)$. Therefore, in Algorithm 1, the computational complexities of lines 11, 15, 19 are O(1), those of lines 4, 7, 8, 9, 14, 16, 18 are $O(n)$, and those of lines 3, 6, 12, 17 are $O(n^2)$, respectively. Furthermore, in each period, lines 11, 14, and 15 are executed $O(n)$ times and all other parts are executed once. Given that there are totally T periods in the simulation, the overall computational complexity of a simulation replication is $O(Tn^2)$.

If the matrix is sparse with density ρ, the computational complexity of multiplication of a $1 \times n$ vector and a $n \times n$ sparse matrix reduces to $O(\rho n^2)$. In our problem, the adjacency matrix is typically sparse. Then, using sparse matrix techniques in the simulation algorithm not only reduces the memory consumption, but also improves the computing efficiency. Therefore, the overall computational complexity reduces to $O(T\rho n^2)$. If $n\rho \sim$ constant as we showed in Equation (11), the overall computational complexity is $O(Tn)$. We summarize these results in the following theorem.

Theorem 1. *For a general inventory system with n items, the overall computational complexity of a single replication of Algorithm 1 with T periods is* $O(Tn^2)$. *Using the sparse matrix techniques to handle the adjacency matrix, the complexity is* $O(T\rho n^2)$. *Furthermore, if Equation* (11) *holds, the complexity is* $O(Tn)$.

Remark 1. Although Theorem 1 is easy to derive, it has important implications. First, it showed that the use of sparse matrix techniques in the simulation algorithm can reduce the computational complexity from $O(Tn^2)$ to $O(Tn)$. This is a significant reduction, especially when handling large-scale inventory systems, where n is in the orders of 10^4–10^5. Second, it provides a benchmark result to understand the performance of the gradient computation algorithms that are discussed in next section.

5. Simulation Optimization

Once we have the simulation model, our next step is to develop a gradient-based simulation optimization algorithm to solve the inventory optimization problem (7). To develop such an algorithm, we need to find an efficient algorithm to estimate the gradient of the expected total inventory cost $\mathbb{E}[\sum_{t=1}^{T} C_t(S)]$ with respect to the base-stock levels S.

5.1. Gradient Computation

The simplest method for gradient estimation may be finite difference approximations. It is easy to understand and implement. However, it produces biased estimators and the computation time is prohibitively long, because at least $n+1$ simulation runs are necessary to compute just one observation of the gradient. Single-run unbiased gradient estimation methods have been proposed in the simulation literature, including the IPA method and the likelihood ratio method. Between these two methods, it is reported that when both are applicable, the IPA method typically produces gradient estimators that have smaller variances than the likelihood ratio method does (Glasserman 2003).

5.1.1. IPA.

IPA is one of the most important gradient estimation methods in the field of stochastic simulation (Ho et al. 1983). Based on Glasserman (2003), we can estimate the derivative of $\mathbb{E}[J(\theta)]$ using $\mathbb{E}\left[\frac{d}{d\theta}J(\theta)\right]$, where $\frac{d}{d\theta}J(\theta)$ is known as the IPA estimator (or pathwise gradient estimator), if following conditions are satisfied:

- Function $J(\theta)$ is differentiable with a probability of one, and
- Function $J(\theta)$ is stochastically Lipschitz.

In our simulation model, the mathematical operations are typically addition, multiplication, min(), or max(), which satisfy the previous conditions. Besides, a detailed validation for using the IPA method in gradient estimation for multiechelon production-inventory systems can be seen in Glasserman and Tayur (1995).

Let $I_t^{temp} = I_{t-1} + P_t - B_{t-1}^{out} - D_t^{out}$, $I_t^{need} = (O_t + O_{t-1}^b) \times A'$ and $\beta_{t,i} = \min_{k, a_{k,i} \neq 0}\{r_{t,k}\}$, and let E denote the identity matrix. Corresponding to each step in the simulation model, the gradient of the total sample cost with respect to the base-stock levels can be calculated using the procedure shown in Algorithm 2.

Algorithm 2 (Procedure for Gradient Computation Using IPA)

```
1  Initialization: $\frac{\partial IP_0}{\partial S} = \frac{\partial I_0}{\partial S} = 0, \frac{\partial O_0}{\partial S} = 0, \frac{\partial B_0^{out}}{\partial S} = 0, \frac{\partial O_0^b}{\partial S} = 0, \frac{\partial C_0^{sum}}{\partial S} = 0;$
2  for $t = 1$ to $T$ do
3      $\frac{\partial IP_t}{\partial S} \leftarrow \frac{\partial IP_{t-1}}{\partial S} + \frac{\partial O_{t-1}}{\partial S} - A \times \frac{\partial O_{t-1}}{\partial S};$
4      $\frac{\partial O_t}{\partial S} \leftarrow diag\left(-1_{\{(IP_t - D_t^{out} - S) < 0\}}\right) \times \left(\frac{\partial IP_t}{\partial S} - E\right);$
5      for $\varsigma = 1$ to $n_L - 1$ do
6          $\frac{\partial IP_t^{temp}}{\partial S} \leftarrow \frac{\partial IP_t}{\partial S} - A \times \frac{\partial O_t}{\partial S};$
7          $\frac{\partial O_t}{\partial S} \leftarrow diag\left(-1_{\{(IP_t^{temp} - S) < 0\}}\right) \times \left(\frac{\partial IP_t^{temp}}{\partial S} - E\right);$
8      $\frac{\partial I_t^{temp}}{\partial S} \leftarrow \frac{\partial I_{t-1}}{\partial S} + \frac{\partial P_t}{\partial S} - \frac{\partial B_{t-1}^{out}}{\partial S};$
9      $\frac{\partial I_t^0}{\partial S} \leftarrow diag(1_{\{I_t^{temp} > 0\}}) \times \frac{\partial I_t^{temp}}{\partial S};$
10     $\frac{\partial B_t^{out}}{\partial S} \leftarrow diag(-1_{\{I_t^{temp} \leq 0\}}) \times \frac{\partial I_t^{temp}}{\partial S};$
11     for each raw material item $i$ do
12         $\frac{\partial P_{t+\ell_{t,i},i}}{\partial S} \leftarrow \frac{\partial O_{t,i}}{\partial S}$
13     $\frac{\partial I_t^{need}}{\partial S} \leftarrow A \times \left(\frac{\partial O_t}{\partial S} + \frac{\partial O_{t-1}^b}{\partial S}\right);$
14     $\frac{\partial r_t}{\partial S} \leftarrow diag\left(1_{\left\{\frac{I_t^0}{I_t^{need}} < 1\right\}}\right) \times diag\left(\frac{1}{I_t^{need}}\right) \times \left(\frac{\partial I_t^0}{\partial S} - diag\left(\frac{I_t^0}{I_t^{need}}\right) \times \frac{\partial I_t^{need}}{\partial S}\right);$
15     for each final product or subassembly item $i$ do
16         $\frac{\partial \beta_{t,i}}{\partial S} \leftarrow (1_{\{r_{t,k} = \beta_{t,i}, a_{k,i} \neq 0\}})_{1 \times n} \times \frac{\partial r_t}{\partial S};$
17         $\frac{\partial M_{t,i}}{\partial S} \leftarrow (O_{t,i} + O_{t-1,i}^b)\frac{\partial \beta_{t,i}}{\partial S} + \beta_{t,i}\left(\frac{\partial O_{t,i}}{\partial S} + \frac{\partial O_{t-1,i}^b}{\partial S}\right);$
18         $\frac{\partial P_{t+\ell_{t,i},i}}{\partial S} \leftarrow \frac{\partial M_{t,i}}{\partial S};$
19     $\frac{\partial I_t}{\partial S} \leftarrow \frac{\partial I_t^0}{\partial S} - A \times \frac{\partial M_t}{\partial S};$
20     $\frac{\partial O_t^b}{\partial S} \leftarrow \frac{\partial O_t}{\partial S} + \frac{\partial O_{t-1}^b}{\partial S} - \frac{\partial M_t}{\partial S};$
21     $\frac{\partial C_t}{\partial S} \leftarrow h \times \frac{\partial I_t}{\partial S} + p \times \frac{\partial B_t^{out}}{\partial S};$
22     $\frac{\partial C_t^{sum}}{\partial S} \leftarrow \frac{\partial C_{t-1}^{sum}}{\partial S} + \frac{\partial C_t}{\partial S}$
```

It can be seen that most operations in Algorithm 2 are the operations of the Jacobian matrices. Given that the computational complexities of the addition and multiplication of two scalars are both $O(1)$, the computational complexities of the addition and multiplication of two $n \times n$ matrices are $O(n^2)$ and $O(n^3)$, respectively. Therefore, for the steps in the procedure, the computation complexities of lines 12, 16–18, and 22 are all $\mathrm{O}(n)$, those of lines 4, 7–10, 14, 20, and 21 are all $\mathrm{O}(n^2)$, and those of lines 3, 6, 13, and 19 are all $\mathrm{O}(n^3)$. Furthermore, lines 12 and 16–18 are all executed $\mathrm{O}(n)$ times and other lines are all executed $\mathrm{O}(1)$ times in each period. Hence, the overall computational complexity of calculating the IPA estimator over T periods is $\mathrm{O}(Tn^3)$. Considering the sparsity of the BOM network and Equation (11), we have the following theorem on the computational complexity of Algorithm 2.

Theorem 2. *For a general inventory system with n items, the computational complexity of Algorithm 2 for a simulation with T periods is* $\mathrm{O}(Tn^3)$. *Suppose that the sparse matrix techniques are used to handle the adjacency matrix, the computational complexity is* $\mathrm{O}(T\rho n^3)$. *Furthermore, if Equation (11) holds, the computational complexity may be reduced to* $\mathrm{O}(Tn^2)$.

Remark 2. Recall that the computational complexity of the simulation algorithm is approximately $\mathrm{O}(Tn)$ when Equation (11) holds, the computational complexity of calculating the IPA estimator is actually of the same order as the finite difference estimation, which requires running the simulation model at least $n+1$ times. This result seems somehow counterintuitive, because we typically think the IPA estimator, which requires only a single replication, is computationally more efficient than the finite-difference estimators, which require $O(n)$ replications. The reason is that, although the IPA calculates the sample path gradient in just one simulation, its calculation requires repeatedly handling of the Jacobian matrices, which requires far more computing time than the vector computations in the simulation algorithm.

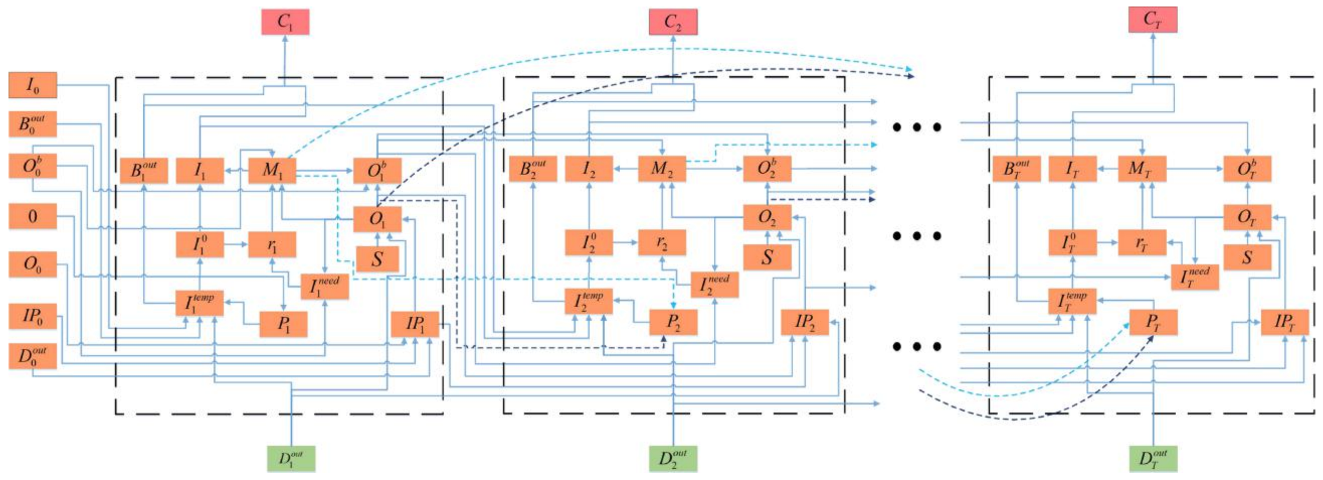
Numerical results also show that for large-scale inventory systems, gradient estimation using IPA is rather time-consuming. To meet the needs of further simulation optimization, more efficient gradient estimation methods need to be considered.

5.1.2. BP. As we have shown in Section 3, the training of large-scale RNNs also need to compute the sample-path gradient of the sample total loss with respect to the weights of the neural networks, which is typically done by the BP algorithms. The BP algorithm for training neural networks was proposed by Rumelhart et al. (1986), and it is one of the corner stones of deep learning. The idea of the BP algorithm is to compute the gradient using a reverse mode, after finishing the calculation of the function value. It is different from the IPA algorithm, which uses a forward mode to calculate the gradient alongside the calculation of the function value. It has been shown that, when the input is of a multidimensional vector (such as the weight vector in RNN training or the base-stock level vector in inventory optimization), the BP algorithm is typically computationally more efficient than the forward mode algorithms (Giles and Glasserman 2006).

In deep learning frameworks like TensorFlow, PyTorch, and so on, BP algorithms are typically implemented through computational graphs. A computational graph is a directed graph for expressing and evaluating mathematical expressions of an algorithm. We construct the computational graph of Algorithm 1 and present it in Figure 6. It shows how the simulation algorithm is executed, and it serves as a map for us to develop the BP algorithm for the inventory model.

The BP algorithm calculates the gradient after finishing the calculation of the function value. In our inventory simulation model, it requires recording certain gradient-related information in the process of simulation before entering in the backward stage. We summarize this part in Algorithm 3.

Figure 6. (Color online) Computational Graph



Algorithm 3 (Procedure for Gradient Computation Using BP Part 1)

```
1  for $t = 1$ to $T$ do
2      push $\frac{\partial O_t}{\partial IP_t} = diag\left(-1_{\{(IP_t - D_t^{out} - S)<0\}}\right)$ into Stack1;
3      for $\varsigma = 1$ to $n_L - 1$ do
4          push $\frac{\partial O_t}{\partial IP_t^{temp}} = diag\left(-1_{\{(IP_t^{temp} - S)<0\}}\right)$ into Stack1
5          record $\frac{\partial I_t^0}{\partial I_t^{temp}} = diag\left(1_{\{I_t^{temp}>0\}}\right)$ and $\frac{\partial B_t^{out}}{\partial I_t^{temp}} = diag\left(-1_{\{I_t^{temp}\le 0\}}\right)$;
6          for each item $i$ which has procurement order in period $t$ do
7              record $i$ in List1 of period $t + \ell_{t,i}$;
8              record $\ell_{t,i}$ in List2 of period $t + \ell_{t,i}$;
9          record $\frac{\partial r_t}{\partial I_t^0} = diag\left(1_{\left\{\frac{I_t^0}{I_t^{need}}<1\right\}}\right) \times diag\left(\frac{1}{I_t^{need}}\right)$ and
           $\frac{\partial r_t}{\partial I_t^{need}} = diag\left(1_{\left\{\frac{I_t^0}{I_t^{need}}<1\right\}}\right) \times diag\left(\frac{1}{I_t^{need}}\right) \times diag\left(\frac{I_t^0}{I_t^{need}}\right)$;
10         for each item $i$ whose started production is nonzero in period $t$ do
11             record $i$ in List3 of period $t + \ell_{t,i}$;
12             record $\ell_{t,i}$ in List4 of period $t + \ell_{t,i}$;
13             record $\beta_{t,i}$ and $\frac{\partial M_{t,i}}{\partial r_t} = (O_{t,i} + O_{t-1,i}^b)(1_{\{r_{t,k}=\beta_{t,i}, a_{k,i}\neq 0\}})_{1\times n}$;
```

After recording gradient-related information during the simulation process, the gradient of the sample total cost with respect to the base-stock levels may be computed form the last period T backward to period 1. The procedure is summarized in Algorithm 4.

Algorithm 4 (Procedure for Gradient Computation Using BP Part 2)

```
1  Initialization: $\frac{\partial C_T^{sum}}{\partial I_{T+1}^{temp}} = 0$, $\frac{\partial C_T^{sum}}{\partial IP_T} = 0$, $\frac{\partial C_T^{sum}}{\partial S} = 0$, $\frac{\partial C_T^{sum}}{\partial O_T^b} = 0$,
   $\frac{\partial C_T^{sum}}{\partial r_t} = 0$, $\frac{\partial C_T^{sum}}{\partial O_t} = 0$, $\frac{\partial C_T^{sum}}{\partial M_t} = 0 (t = 1, \ldots, T)$;
2  for $t = T$ to $1$ do
3      $\frac{\partial C_T^{sum}}{\partial I_t} \leftarrow h + \frac{\partial C_T^{sum}}{\partial I_{t+1}^{temp}}$, $\frac{\partial C_T^{sum}}{\partial B_t^{out}} \leftarrow p - \frac{\partial C_T^{sum}}{\partial I_{t+1}^{temp}}$;
4      $\frac{\partial C_T^{sum}}{\partial M_t} \leftarrow \frac{\partial C_T^{sum}}{\partial M_t} - \frac{\partial C_T^{sum}}{\partial I_t} \times A - \frac{\partial C_T^{sum}}{\partial O_t^b}$;
5      $\frac{\partial C_T^{sum}}{\partial O_t^{temp}} \leftarrow \frac{\partial C_T^{sum}}{\partial O_t^b} + \frac{\partial C_T^{sum}}{\partial M_t} \times diag\left((\beta_{t,i})_{1\times n}\right)$;
6      for each item $i$ whose started production is nonzero in period $t$ do
7          $\frac{\partial C_T^{sum}}{\partial r_t} \leftarrow \frac{\partial C_T^{sum}}{\partial r_t} + \frac{\partial C_T^{sum}}{\partial M_{t,i}} \frac{\partial M_{t,i}}{\partial r_t}$;
8      $\frac{\partial C_T^{sum}}{\partial I_t^0} \leftarrow \frac{\partial C_T^{sum}}{\partial I_t} + \frac{\partial C_T^{sum}}{\partial r_t} \times \frac{\partial r_t}{\partial I_t^0}$;
9      $\frac{\partial C_T^{sum}}{\partial I_t^{need}} \leftarrow \frac{\partial C_T^{sum}}{\partial r_t} \times \frac{\partial r_t}{\partial I_t^{need}}$;
10     $\frac{\partial C_T^{sum}}{\partial O_t^{temp}} \leftarrow \frac{\partial C_T^{sum}}{\partial O_t^{temp}} + \frac{\partial C_T^{sum}}{\partial I_t^{need}} \times A$;
11     $\frac{\partial C_T^{sum}}{\partial I_t^{temp}} \leftarrow \frac{\partial C_T^{sum}}{\partial I_t^0} \times \frac{\partial I_t^0}{\partial I_t^{temp}} + \frac{\partial C_T^{sum}}{\partial B_t^{out}} \times \frac{\partial B_t^{out}}{\partial I_t^{temp}}$;
12     $\frac{\partial C_T^{sum}}{\partial O_t} \leftarrow \frac{\partial C_T^{sum}}{\partial O_t} + \frac{\partial C_T^{sum}}{\partial O_t^{temp}} + \frac{\partial C_T^{sum}}{\partial IP_t} \times (E - A)$;
13     for $\varsigma = 1$ to $n_L - 1$ do
14         pop $\frac{\partial O_t}{\partial IP_t^{temp}}$ from Stack1;
15         $\frac{\partial C_T^{sum}}{\partial IP_t^{temp}} \leftarrow \frac{\partial C_T^{sum}}{\partial O_t} \times \frac{\partial O_t}{\partial IP_t^{temp}}$, $\frac{\partial C_T^{sum}}{\partial S} \leftarrow \frac{\partial C_T^{sum}}{\partial S} - \frac{\partial C_T^{sum}}{\partial IP_t^{temp}}$;
16         $\frac{\partial C_T^{sum}}{\partial IP_t} \leftarrow \frac{\partial C_T^{sum}}{\partial IP_t} + \frac{\partial C_T^{sum}}{\partial IP_t^{temp}}$, $\frac{\partial C_T^{sum}}{\partial O_t} \leftarrow - \frac{\partial C_T^{sum}}{\partial IP_t^{temp}} \times A$;
17     pop $\frac{\partial O_t}{\partial IP_t}$ from Stack1;
18     $\frac{\partial C_T^{sum}}{\partial S} \leftarrow \frac{\partial C_T^{sum}}{\partial S} - \frac{\partial C_T^{sum}}{\partial O_t} \times \frac{\partial O_t}{\partial IP_t}$;
19     $\frac{\partial C_T^{sum}}{\partial IP_{t-1}} \leftarrow \frac{\partial C_T^{sum}}{\partial IP_t} + \frac{\partial C_T^{sum}}{\partial O_t} \times \frac{\partial O_t}{\partial IP_t}$;
20     $\frac{\partial C_T^{sum}}{\partial O_{t-1}^b} \leftarrow \frac{\partial C_T^{sum}}{\partial O_t^{temp}}$, $\frac{\partial C_T^{sum}}{\partial P_t} \leftarrow \frac{\partial C_T^{sum}}{\partial I_t^{temp}}$;
21     for each item $i$ in List1 do
22         get the recorded lead time $\ell_{u,i}$ in List2 $(u = t - \ell_{u,i})$;
23         $\frac{\partial C_T^{sum}}{\partial O_{t-\ell_{u,i},i}} \leftarrow \frac{\partial C_T^{sum}}{\partial P_{t,i}}$;
24     for each item $i$ in List3 do
25         get the recorded lead time $\ell_{u,i}$ in List4 $(u = t - \ell_{u,i})$;
26         $\frac{\partial C_T^{sum}}{\partial M_{t-\ell_{u,i},i}} \leftarrow \frac{\partial C_T^{sum}}{\partial P_{t,i}}$;
```

Different from the IPA algorithm (Algorithm 2), which is mainly composed of operations of Jacobian matrices, the BP algorithm (Algorithms 3 and 4) is mainly composed of vector operations. In Algorithm 3, the computational complexities of lines 2, 4, 5, and 9 are $O(n)$, those of lines 7–8 and 11–13 are $O(1)$, and they are executed $O(n)$ times in each period. Therefore, the computational complexity of Algorithm 3 is $O(Tn)$. In Algorithm 4, the computational complexities of lines 22–23 and 25–26 are all $O(1)$, those of lines 3, 5, 7, and 20 are all $O(n)$, and those of lines 4, 8–12, 15–16, and 18–19 are all $O(n^2)$. Because 7, 22–23, and 25–26 are executed $O(n)$ times, whereas others are executed $O(1)$ times in each time period, the computational complexity of Algorithm 4 is therefore $O(Tn^2)$. Then, we have the following theorem on the computational complexity of the BP algorithm.

Theorem 3. *For a general inventory system with n items, the computational complexity of the BP algorithm (i.e., Algorithms 3 and 4) for a simulation with T periods is* $O(Tn^2)$. *Suppose that the sparse matrix techniques are used to handle the adjacency matrix, the computational complexity is* $O(T\rho n^2)$. *Furthermore, if Equation (11) holds, the computational complexity may be reduced to* $O(Tn)$.

With the BP algorithm, the computational complexity of gradient calculation is now the same as that of the simulation algorithm, and it is of an order n faster than the IPA algorithm. In the numerical results reported in Section 6, we find that the BP algorithm and IPA algorithm produce the same gradient estimates (as expected), but the BP algorithm is significantly faster especially when n is large.

5.2. Optimization Algorithm

As described in Section 2, the objective of the inventory optimization problem is to find the optimal base-stock levels S that minimize the expected cumulative cost over T periods, that is,

$$\min_S \quad \mathbb{E}\left[\sum_{t=1}^{T} C_t(S)\right].$$

To solve this, we can compute the sample-path gradient with respect to the base-stock levels and apply SGD algorithms. However, by applying SGD algorithms directly, we observe that the solution typically keeps inventory at almost all nodes, and it is very difficult to implement in practice. Instead, we want to find solutions that keep inventory only at a small fraction of the nodes. To solve the problem, we add the L_1 norm of the decision variables along with a tuning parameter $\lambda > 0$, and solve the following problem:

$$\min_S \quad \mathbb{E}\left[\sum_{t=1}^{T} C_t(S)\right] + \lambda\|S\|_1. \tag{12}$$

The L_1 regularization is also known as Lasso in the statistics literature. It is known in the machine learning literature that, with an appropriate tuning parameter λ, adding the L_1 regularization can force many of the decision variables to zero and keep only those very important ones nonzero, thus introducing sparsity to the optimal solution (Tibshirani 1996).

From another perspective, for large-scale inventory optimization problems, the number of samples used in the optimization process is not very large, typically less than number of decision variables (i.e., the base-stock levels). Thus, it is an over-parameterized model (Soltanolkotabi et al. 2018), and the bias of the sample optimal value cannot be ignored. To reduce the bias in the optimal solution, regularization methods are also typically used (Bühlmann and Van De Geer 2011). Therefore, the L_1 regularization introduced in Problem (12) not only keeps the inventory only at a small fraction of the nodes but also helps reduce the bias.

The L_1 norm term in Problem (12) is not smooth. One way to solve the optimization problem is to use the stochastic subgradient descent (SSGD) algorithm (Shor 2012). Let $F(S) = \mathbb{E}[f(S)] = \mathbb{E}[\sum_{t=1}^{T} C_t(S)]$ and $G(S) = \lambda\|S\|_1$. The iteration of the SSGD algorithm takes the form

$$S_{k+1} = S_k - t_k\big(\nabla f(S_k) + \xi_k\big),$$

where t_k is the step size, and ξ_k denotes the subgradient of G at S_k. A main drawback of this method is its lack of capability in exploiting the problem structure of the L_1 norm, thus having poor convergence properties (Xiao 2009).

In contrast, there exist other optimization methods that exploit the problem structures and are better suited for this problem. The objective function of Problem (12) is the sum of a smooth function and a simple convex nonsmooth function. The proximal gradient method, sometimes called ISTA (iterative shrinkage-thresholding algorithm), can solve this type of problems, where the objective is a sum of a differentiable term and a nondifferentiable convex function, with a convergence rate of $O(1/k)$ (Parikh and Boyd 2014). By using Nesterov acceleration method (Nesterov 1983), FISTA (fast ISTA) promotes the convergence rate to $O(1/k^2)$ (Beck and Teboulle 2009). FISTA is currently a popular algorithm in the field of machine learning.

For the optimization problem with objective

$$\min F(x) + G(x),$$

where F is a smooth cost function, and G is a possibly nonsmooth regularization term, the basic iteration of FISTA is

$$\begin{aligned} x_{k+1} &= prox_{t_{k+1}G}\big(y_k - t_{k+1}\nabla F(y_k)\big) \\ y_{k+1} &= x_{k+1} + \frac{k}{k+r}(x_{k+1} - x_k), \end{aligned}$$

where $r \geq 3$, $\{t_k\}$ is a sequence of positive and nonincreasing step sizes, and $prox_{t_{k+1}G}(\cdot)$ is the proximal operator. If $G = \lambda\|x\|_1$,

$$prox_{t_{k+1}G}(x)_i = sign(x_i)\max\big(0, (|x_i| - t_{k+1}\lambda)\big).$$

When the function F is represented as an expectation $F(x) = \mathbb{E}_\xi[f(\xi, x)]$, as in our problem, we need to use a stochastic version of FISTA (Atchadé et al. 2017, Salim and Hachem 2019). The true gradient $\nabla F(y_k)$ is replaced by a sample-path gradient $\nabla f(y_k)$ in each iteration. When used with a constant step size, the stochastic FISTA achieves a convergence rate of $O(1/\sqrt{k})$, and with decreasing step sizes it achieves a rate of $O\big(\log(k)/\sqrt{k}\big)$. Moreover, the stochastic FISTA is close to its $O(1/k^2)$ deterministic behavior in the first several iterations (Salim and Hachem 2019).

The inventory optimization problems are different from machine learning problems such as linear regression or training of neural networks. In machine learning problems, there are often inherent sparsity in the parameters, that is, only a small number of explanatory variables or parameters are relevant. Therefore, asymptotically, under appropriate conditions, L_1-regularization not only recovers the true nonzero parameters but also delivers the optimal objective value for the original problem without regularization (Wainwright 2009). In inventory optimization problems, however, there is often no reason to believe that the optimal solution to the original problem (i.e., Problem (7)) is sparse, and the sparsity is an additional requirement that we want to bring to the solution. Therefore, we suggest adding a reoptimization step to further improve the performance of the inventory decisions once the inventory locations are determined by Problem (12). Then, we have the following two-stage procedure.

Stage 1. Solve the optimization problem with the L_1 norm regularizer to select the inventory locations:

$$S_1^* = \operatorname{argmin}\left\{ \mathbb{E}\left[\sum_{t=1}^{T} C_t(S)\right] + \lambda \|S\|_1 \right\}.$$

Stage 2. Based on the result from the previous step, fix certain elements of S to be zero and solve the modified problem:

$$\min \mathbb{E}\left[\sum_{t=1}^{T} C_t(\tilde{S})\right] \quad \text{where } \tilde{S} = S \odot 1_{\{S_1^*>0\}},$$

where $\odot$ denotes element-wise product.

In Stage 1, our goal is to select a small fraction of the nodes to keep inventory. Parameter λ controls the number of nonzero inventory levels and needs to be tuned for specific problems. In Stage 2, we only consider the selected inventory locations and solve a much smaller-scale smooth inventory optimization problems to determine their base-stock levels. We use the stochastic FISTA in Stage 1 and the SGD in Stage 2.

Solving the regularized problem in two stages is not new. In the field of high-dimensional regression, Meinshausen (2007) proposed a two-stage procedure, termed the relaxed Lasso for high-dimensional data where the number of predictor variables p is much larger than the number of observations n. Their theoretical and numerical results demonstrate that the two-stage procedure performs better than the regular Lasso estimator for high-dimensional data.

6. Numerical Experiments

In this section, we first test the performances of the simulation and gradient computation algorithms on inventory systems of different scales. Then, we conduct a series of experiments to understand the behaviors of the optimization algorithm and to compare it with the GS model of Graves and Willems (2000) on small- to medium-scale problems where the GS model applies. In this section, all the computer programs are coded in Python and all the experiments are run on a computer with two Intel Xeon Gold 6248R CPUs (each with 24 cores) and 256 GB RAM. Deterministic lead times are used in all experiments, except the ones in Section 6.3.3. A package containing all source code and data of the experiments is available at this paper's GitHub repository (https://github.com/wangruohai/LSIO-RNNISA). Details that are not described in this section can be found in the package.

6.1. Simulation and Gradient Computation

To test the performance of our algorithms, we build test inventory problems based on the problem characteristics that we observe from our consulting experience. In the test problems, we use BOM networks that have general topology (Figure 4(c)) and set the average degrees $\langle k \rangle$ to be 10. Demands are normally distributed, lead times are deterministic, and time horizons are 100 periods. Other details of the settings may be found in the paper's GitHub repository.

6.1.1. Performance of the Simulation Algorithms.

We test the traditional simulation algorithm, the tensorized simulation algorithms (i.e., Algorithm 1) with dense matrices or sparse matrices for inventory systems with different number of nodes, ranging from medium-scale system with 1,000 nodes to very large-scale system with 500,000 nodes, and report the simulation run times of different algorithms, averaging over 10 independent replications in Table 1.

There are several interesting findings from the results. First, the computational complexities of the traditional algorithm and the tensorized algorithm with dense matrices are approximately $O(n^2)$, whereas that of the tensorized algorithm with sparse matrices is approximately $O(n)$, which are consistent with Theorem 1. Second, although tensorization does not improve the computational complexity, it reduces the computational time significantly. Third, the use of sparse matrices reduces significantly both the memory requirements and the run times, further allowing the algorithm to handle very large-scale problems.

6.1.2. Performance of the Gradient Computation Algorithms.

We use the same test problems to test the two gradient computation algorithms proposed in this paper: the IPA algorithm (Algorithm 2) and the BP Algorithm (Algorithms 3 and 4). Both algorithms are tensorized, and both may use dense or sparse adjacency matrices. The run times of different algorithms are reported in Table 2.

From the results we see that the computational complexities of the IPA algorithms, with dense and sparse adjacency matrices, and the BP algorithms, with dense and sparse adjacency matrices, are approximately $O(n^3)$, $O(n^2)$, $O(n^2)$ and $O(n)$, respectively, which are consistent with Theorems 2 and 3. Furthermore, it is clear that the BP algorithms significantly outperform the IPA algorithms in general, and the BP algorithm with sparse adjacency matrices is capable of handling very large-scale problems.

Table 2. Run Time of Gradient Computation for a Single Replication

Number of nodes	IPA dense	IPA sparse	BP dense	BP sparse
1,000	0.64 min	0.38 min	1.23 s	0.73 s
5,000	20.05 min	4.30 min	6.88 s	2.84 s
10,000	139.94 min	18.65 min	18.18 s	5.32 s
50,000	—	1,264.22 min	268.85 s	25.45 s
100,000	—	—	958.17 s	51.13 s
500,000	/	—	/	278.62 s

Table 3. Run Time of Simulation Using TensorFlow for a Single Replication

Number of nodes	TensorFlow CPU dense	TensorFlow GPU dense	TensorFlow CPU sparse	TensorFlow GPU sparse
1,000	1.93 s	2.58 s	1.91 s	2.64 s
5,000	15.58 s	11.33 s	9.77 s	11.25 s
10,000	42.24 s	22.94 s	19.70 s	22.09 s
50,000	739.20 s	130.28 s	100.00 s	113.01 s
100,000	2,923.14 s	/	199.75 s	226.32 s
500,000	/	/	949.22 s	1125.61 s

6.1.3. Simulation and Gradient Computation Using TensorFlow. One major impediment to the use of our algorithms in general inventory optimization is the construction of the computational graph and the derivation of the sample-path gradient. Small changes of the logic of the simulation process may result in quite different computational graphs and thus very different gradient estimators. Therefore, the use of these algorithms may require a significant amount of analyst's effort, which may be very expensive in practice. This motivates us to program the tensorized simulation algorithm using the machine-learning tool TensorFlow, which may construct the computational graph and compute the sample-path gradient automatically. Moreover, the use of TensorFlow also makes it possible to use GPUs for matrix operations used in the algorithms, which may provide further speedups.

We implement the tensorized simulation algorithm using TensorFlow and also use its automatic BP tool to compute the sample-path gradient. We test both algorithms with dense or sparse adjacency matrices. We further test the algorithms using an additional NVIDIA GeForce RTX 3090 GPU with 24 GB of memory. The results are reported in Tables 3 and 4.

There are several interesting findings from these results. First, when the inventory problems are of medium to large scales (e.g., 1,000 to 50,000 nodes), the TensorFlow implementations of the simulation algorithm and the automatic BP algorithm, both with sparse adjacency matrices, provide acceptable run times, making TensorFlow a very promising tool for inventory optimization. Second, it appears that the use of GPU does not always accelerate the computation. It is because, compared with CPU, GPU requires additional data communication between the main memory and the GPU memory and is not efficient for less compute-intensive tasks (such as scenarios with a small number of nodes or when sparse matrix is adopted; Teodoro et al. 2009). Third, compared with Tables 1 and 2, it is clear that our original method with sparse adjacency matrices is significantly faster than the TensorFlow implementations, making it an ideal tool to solve large- to very large-scale problems, although it requires a significant amount of analyst's effort to construct the computational graph and to derive the BP gradient.

6.1.4. Performance of Multiple Replications. To solve the inventory optimization problems, the objective value and the gradient need to be evaluated based on multiple replications (i.e., a mini-batch) in each epoch (i.e., iteration of the optimization algorithms). Therefore, it is important to understand the run times of the algorithms when multiple replications are run in parallel. We test the algorithms with a mini-batch of 10 and report the results in Tables 5 and 6.

We use the multiprocessing parallel computing technique, and each replication is executed on a separate CPU core. As the communication and data transmission between the parent and children processes takes some time, the run time of multiple replications is higher than that of a single replication. However, as can be seen in the tables, the run time of a mini-batch of 10 is much lower than 10 times that of a single replication, suggesting that we should take advantage of the multicore capability in the optimization process.

6.2. Performance of the Optimization Algorithms

Remember that in the Stage 1 of the optimization procedure, we solve a L_1 regularized stochastic optimization problem (12). Both the SSGD and FISTA algorithms

Table 4. Run Time of Gradient Computation Using TensorFlow for a Single Replication

Number of nodes	TensorFlow CPU dense	TensorFlow GPU dense	TensorFlow CPU sparse	TensorFlow GPU sparse
1,000	11.51 s	15.15 s	12.43 s	13.31 s
5,000	70.83 s	69.80 s	60.63 s	61.76 s
10,000	186.65 s	137.78 s	140.51 s	124.10 s
50,000	2,139.76 s	/	863.60 s	663.36 s
100,000	8,703.72 s	/	2,265.24 s	
500,000	/	/	21,629.01 s	

Table 5. Run Time of Simulation for Multiple Replications

Number of nodes	Single replication	Mini-batch of 10
1,000	0.17 s	0.63 s
5,000	0.59 s	1.45 s
10,000	1.14 s	2.54 s
50,000	5.45 s	11.34 s
100,000	10.92 s	21.99 s
500,000	71.18 s	126.27 s

Table 6. Run Time of BP for Multiple Replications

Number of nodes	Single replication	Mini-batch of 10
1,000	0.73 s	1.27 s
5,000	2.84 s	3.95 s
10,000	5.32 s	7.75 s
50,000	25.45 s	33.36 s
100,000	51.13 s	66.58 s
500,000	278.62 s	374.37 s

may be used. We first compare the performance of these two algorithms on the inventory optimization problem with different scales. The subgradient of the L_1 norm ($\|S\|_1$) is needed in the implementation of the SSGD algorithm, and we use the sign function (sign(S)). To provide a fair comparison, for both algorithms, we solve the same test problems used in Section 6.1, we apply the same mini-batch of 10 replications to evaluate the cost and gradient in each epoch, and we use the same regularization parameter (λ) and the step size (t_k). Furthermore, because the two algorithms may stop at different epochs, we report the results based on fixed numbers of epochs. More details of the experiment settings are available at this paper's GitHub repository.

The results are shown in Figure 7. The four rows represent four different problem scales, that is, 10,000,

Figure 7. (Color online) Optimization Curve with Fixed Epochs

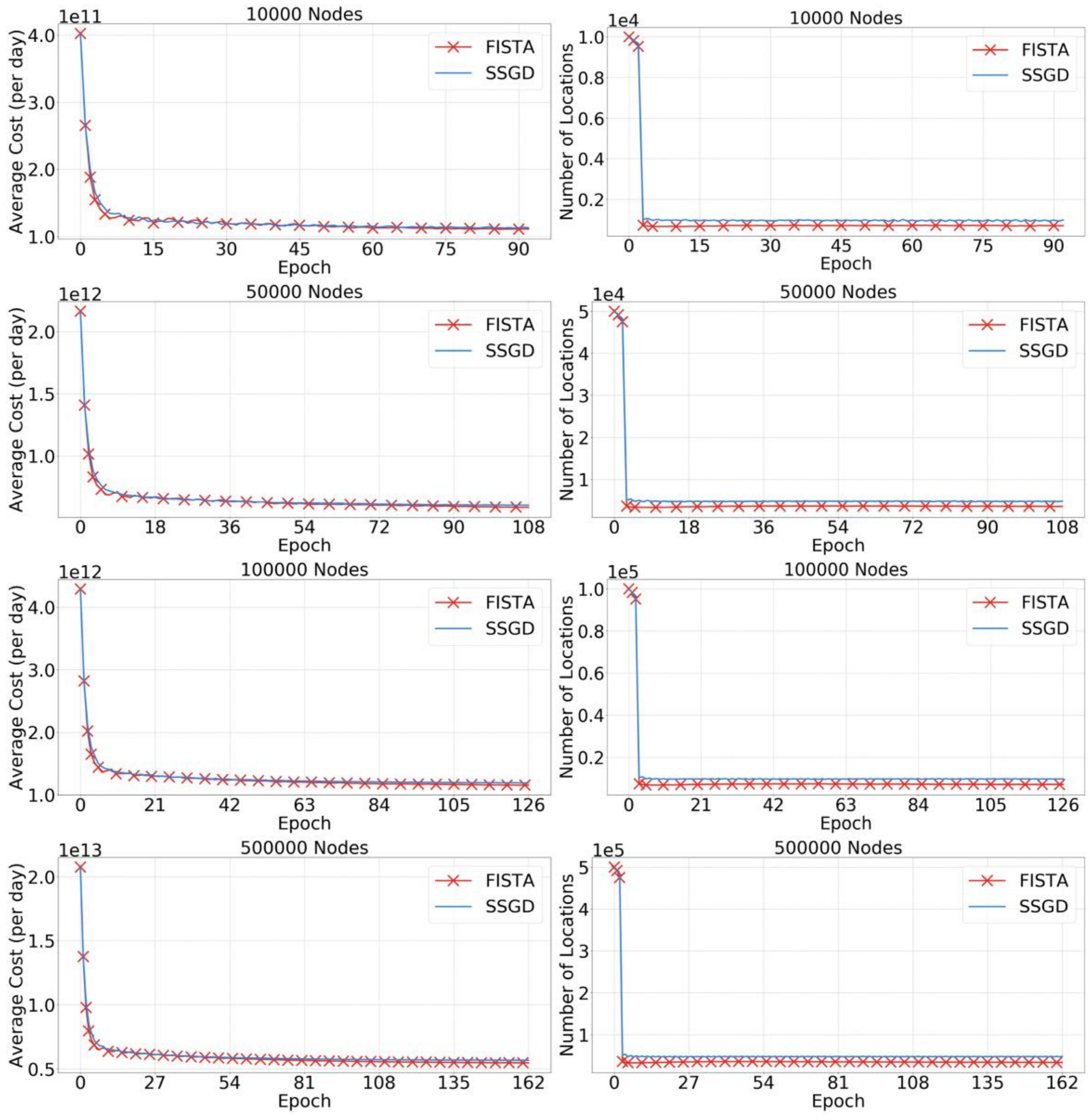


Table 7. Cost Improvement in Stage 2

Number of nodes	10,000	50,000	100,000	500,000
Cost improvement	3.17%	1.41%	1.23%	1.17%

50,000, 100,000, and 500,000 nodes, and the two columns show the objective values (i.e., the average total cost) and the sparsity of the solutions (i.e., the numbers of nonzero base-stock levels). From these plots, we see that, in terms of the objective values, both algorithms have similar performance and the FISTA algorithm finds slightly better solutions. In terms of the sparsity of the solutions, the FISTA algorithm tends to find solutions that are sparser. Therefore, we conclude that in the Stage 1, the FISTA algorithm is a preferred one among the two.

The solution of Stage 1 provides the information not only on where to keep inventory but also on the base-stock levels. However, in our optimization procedure, we only keep the location information and suggest to use Stage 2 optimization to optimize the base-stock levels. To test the advantage of a second stage, we conduct additional simulation experiments to estimate the objective values of the solutions of Stage 1 and Stage 2 and report the results in Table 7. From the results it is clear that the Stage 2 solution is clearly better than the Stage 1 solution for all four problems, validating the use of the Stage 2 to further improve the quality of the solution from Stage 1.

In Table 8, we report the computational times of a typical run of the two-stage optimization procedure, where the FISTA is used in Stage 1 and the SGD is used in Stage 2, and both algorithms are run until the termination conditions are satisfied. From the table, we see that a large-scale inventory optimization problem with 50,000 nodes may be solved in about two hours, and a very large-scale problem with 500,000 nodes may be solved in about 1.5 days. Because inventory decisions are not real-time decisions and they are updated infrequently (e.g., once every six months), these run times are in general acceptable.

6.3. Comparison with the GS Model

The simulation optimization algorithm appears to work well in Section 6.2. However, without a benchmark algorithm to compare with, it is difficult to know the actual performance of the algorithm. To partly solve this problem, we consider the GS model of Graves and Willems (2000). The GS model requires the BOM to be a spanning tree with bounded demands and deterministic lead times, which are difficult to satisfy for complex production systems. Nevertheless, it provides an optimization algorithm that can be used as a benchmark. In this section, we consider three different types of test problems. The first one satisfies all requirements of the GS model, the second one has a more complex network structure, and the third one has random lead times. We compare our algorithm with the algorithm of Graves and Willems (2000). From these three types of test problems, we conclude that when the assumptions of the GS model are satisfied, both algorithms produce solutions with similar objectives and solution sparsity. However, when the assumptions are not satisfied, our algorithm performs significantly better than the GS algorithm, as expected.

Table 8. Run Time of the Two-Stage Optimization Procedure

Number of nodes	10,000	50,000	100,000	500,000
Stage 1	18.12 min	101.40 min	3.87 hr	23.00 hr
Stage 2	14.31 min	32.94 min	1.69 hr	12.02 hr

Throughout this section we assume that the outside demands of all items are independent and normally distributed with different means and variances. To implement the GS model, we need an upper bound for each demand, and we set it as an upper quantile of the normal distribution as suggested by Graves and Willems (2000). Moreover, the GS model quotes a committed service time to each external demand. We revise the logic of our simulation model slightly to adopt the same logic to ensure a more fair comparison.

6.3.1. Spanning Trees.

We start with a simple example of Kodak digital camera supply chain pictured in Figure 2. The lead times and holding costs are listed in Table 9.

We apply both algorithms, and the optimal base-stock levels found by both algorithms are shown in Table 10. As can be seen from the table, both algorithms produce solutions with similar costs and sparsity, with the GS algorithm performing slightly better than our algorithm.

To understand the performances of both algorithms for larger-scale problems, we built some larger inventory systems with spanning tree topology. To do that, we first convert the general BOM networks used in test problems of Section 6.1 into undirected networks. Then, we use the minimum spanning tree algorithm (Graham and Hell 1985) to create spanning trees out of the undirected networks. Finally, the spanning trees are reverted to directed networks with directions of corresponding edges in the original general BOM networks.

Table 9. Lead Time Information and Holding Cost Coefficients of Kodak Supply Chain

	A	B	C	D	E	F	G	H	I	J
l_i	2	3	2	4	2	6	3	4	3	2
h_i	1	3	4	6	12	20	13	8	4	50

Table 10. Optimization Results for Kodak Supply Chain Network

	Base-stock level										
Algorithm	A	B	C	D	E	F	G	H	I	J	Cost
GS	109.70	0	202.00	156.37	0	0	0	0	0	0	1.62×10^5
Our	0	37.08	100.59	0	102.18	0	43.28	0	0	0	1.67×10^5

Table 11. Optimization Performance on Larger Spanning Trees

	GS model			Our algorithm		
Number of nodes	Cost	Location	Time	Cost	Location	Time
10,000	3.72×10^{11}	2,637	1.92 hr	3.94×10^{11}	2,570	1.18 hr
50,000	1.94×10^{12}	13,215	97.70 hr	1.87×10^{12}	14,973	4.62 hr

Table 12. Optimization Results for Modified Kodak Supply Chain Network

	Base-Stock Level										
Algorithm	A	B	C	D	E	F	G	H	I	J	Cost
GS	109.70	0	202.00	156.37	0	0	0	0	0	0	2.30×10^5
Our	63.66	60.63	57.66	0	57.85	0	43.52	0	0	0	1.89×10^5

The results of both algorithms are reported in Table 11. Again, we see that both algorithms produce solutions with similar costs and sparsity. Furthermore, we notice that, when the number of nodes is large, for example, 50,000, our algorithm is significantly faster.

6.3.2. General Network. The GS model proposed by Graves and Willems (2000) is developed for supply chains that can be modeled as spanning trees. For general systems that may include undirected cycles, this model is no longer applicable. Next, we modify the Kodak network and incorporate an undirected cycle ("D-E-G-J-D" in Figure 8) to explore the performance of our method and the GS model for application in general networks.

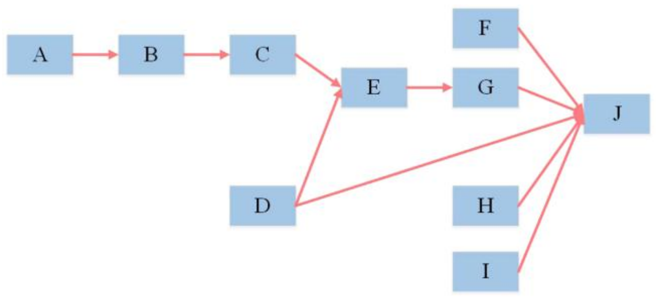
To apply the GS model to the network in Figure 8, a simple idea is to cut off an edge and regard it as a spanning tree. Here, we cut off the arc between D and J so that the optimal base-stock level is the same as that of Table 10. We also obtain the optimal base-stock level by our algorithm, and the results are shown in Table 12. From the results we can see that our algorithm finds the solution with two more nonzero base-stock levels and better cost, compared with the solution of the GS model.

6.3.3. Random Lead Times. The GS model (Graves and Willems 2000) assumes a deterministic lead time for each node in the supply chain. Nevertheless, this is typically not true in practice. For instance, procurement lead times are often quite volatile. We consider a case that the procurement lead times of the raw material nodes in Figure 2 are random, that is, $\ell_{t,i} = \max\left(1, round\left(N(\mu_{\ell_i}, 0.05\mu_{\ell_i})\right)\right)$. For the GS model, we use μ_{ℓ_i} as the (deterministic) lead time. The optimal solutions found by both algorithms are evaluated based on the simulation model, and the total cost of 100 periods for the GS model and our algorithm are 2.14×10^6 and 1.04×10^6, respectively. It is clear that our algorithm performs significantly better than the GS model in this instance.

7. Concluding Remarks

Inspired by the similarities between inventory optimization and RNNs, in this paper, we develop a new

Figure 8. (Color online) Modified Kodak Supply Chain Network



framework for inventory simulation and inventory optimization for large-scale inventory optimization problems on complex production networks. The framework uses modern computational tools to reduce the computational time drastically so that large-scale problems with tens to hundreds of thousands of nodes may be solved in a reasonable amount of time.

There are a few directions that one may further extend our work. First, many practical inventory problems have constraints, for example, capacity constraints or fill-rate constraints. It is interesting to consider how to incorporate these constraints in the framework. Second, the similarity between RNN and dynamic simulation holds not only for inventory simulation but also for general periodically reviewed dynamic systems. It would be interesting to further study how the framework developed in this paper may be extended to simulation and optimization of general dynamic systems.

Acknowledgments

The authors thank Bruno Tuffin, the associate editor, and three referees for helpful comments that improved the presentation of the paper.

Appendix. Derivation of the Inventory Position (Equation (1))

By definition, the inventory position equals the on-hand inventory plus the on-order quantity minus the backorders (demand that have occurred but have not been satisfied), that is,

$$IP_{t,i} = I_{t-1,i} + OO_{t,i} - (B^{out}_{t-1,i} + B^{in}_{t-1,i}), \tag{A.1}$$

where $OO_{t,i}$ denotes the on-order quantity of item i at period t, and $B^{out}_{t-1,i}$ and $B^{in}_{t-1,i}$ denote the backlogged outside and internal demand of item i at period $t-1$, respectively. The on-order quantity refers to the amount that is ordered but not yet received through either procurement or production, that is,

$$OO_{t,i} = \sum_{u=1}^{t-1}(O_{u,i} - P_{u,i}). \tag{A.2}$$

The backlogged internal demand can be calculated by

$$B^{in}_{t,i} = \sum_j a_{ij} O^b_{t,j}. \tag{A.3}$$

Unlike outside demands where at least one of the on-hand inventory and outside backlog have to be zero (see Equations (2) and (3)), it is possible to have positive on-hand inventory and positive internal backlog at the same time, because the backlog of the downstream item may be caused by the shortage of some other components.

According to Equation (6), we have

$$IP_{t,i} = I^0_{t-1,i} - \sum_j a_{ij} M_{t-1,j} + OO_{t,i} - B^{out}_{t-1,i} - B^{in}_{t-1,i}. \tag{A.4}$$

Based on Equations (2) and (3), we have

$$I^0_{t-1,i} - B^{out}_{t-1,i} = I_{t-2,i} + P_{t-1,i} - B^{out}_{t-2,i} - D^{out}_{t-1,i}.$$

Thus,

$$\begin{aligned}
IP_{t,i} &= I_{t-2,i} + P_{t-1,i} - B^{out}_{t-2,i} - D^{out}_{t-1,i} - \sum_j a_{ij} M_{t-1,j} + OO_{t-1,i} \\
&\quad + O_{t-1,i} - P_{t-1,i} - \sum_j a_{ij} O^b_{t-1,j} \\
&= I_{t-2,i} + OO_{t-1,i} - (B^{out}_{t-2,i} + B^{in}_{t-2,i}) + B^{in}_{t-2,i} - D^{out}_{t-1,i} + O_{t-1,i} \\
&\quad - \sum_j a_{ij}(M_{t-1,j} + O^b_{t-1,j}) \\
&= IP_{t-1,i} + O_{t-1,i} - D^{out}_{t-1,i} + \sum_j a_{ij} O^b_{t-2,j} \\
&\quad - \sum_j a_{ij}(O_{t-1,j} + O^b_{t-2,j}) \\
&= IP_{t-1,i} + O_{t-1,i} - D^{out}_{t-1,i} - D^{in}_{t-1,i}.
\end{aligned}$$

Therefore, Equation (1) holds.

References

Abbasi H (2018) Sparse: A more modern sparse array library. Akici F, Lippa D, Niederhut D, Pacer M, eds. *Proc. 17th Python in Sci. Conf.* (SciPy, Austin, TX), 27–30.

Albert R, Barabási AL (2002) Statistical mechanics of complex networks. *Rev. Modern Phys.* 74(1):47.

Arulkumaran K, Deisenroth MP, Brundage M, Bharath AA (2017) Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine* 34(6):26–38.

Atchadé YF, Fort G, Moulines E (2017) On perturbed proximal gradient algorithms. *J. Machine Learn. Res.* 18(10):1–33.

Baydin AG, Pearlmutter BA, Radul AA, Siskind JM (2018) Automatic differentiation in machine learning: A survey. *J. Machine Learn. Res.* 18(153):1–43.

Beck A, Teboulle M (2009) A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. Imaging Sci.* 2(1): 183–202.

Bottou L, Curtis FE, Nocedal J (2018) Optimization methods for large-scale machine learning. *SIAM Rev.* 60(2):223–311.

Bühlmann P, Van De Geer S (2011) *Statistics for High-Dimensional Data: Methods, Theory and Applications* (Springer Science & Business Media, New York).

Buluç A, Fineman JT, Frigo M, Gilbert JR, Leiserson CE (2009) Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks. auf der Heide FM, Bender MA, eds. *Proc. 21st Annual Sympos. on Parallelism in Algorithms and Architectures* (ACM, New York), 233–244.

Chen W, Dawande M, Janakiraman G (2014) Integrality in stochastic inventory models. *Production Oper. Management* 23(9):1646–1663.

Clark AJ, Scarf H (1960) Optimal policies for a multi-echelon inventory problem. *Management Sci.* 6(4):475–490.

Gallego G, Zipkin P (1999) Stock positioning and performance estimation in serial production-transportation systems. *Manufacturing Service Oper. Management* 1(1):77–88.

Giles M, Glasserman P (2006) Smoking adjoints: Fast Monte Carlo Greeks. *Risk* 19(1):88–92.

Glasserman P (2003) *Monte Carlo Methods in Financial Engineering*, vol. 53 (Springer Science & Business Media, New York).

Glasserman P, Tayur S (1994) The stability of a capacitated, multi-echelon production-inventory system under a base-stock policy. *Oper. Res.* 42(5):913–925.

Glasserman P, Tayur S (1995) Sensitivity analysis for base-stock levels in multiechelon production-inventory systems. *Management Sci.* 41(2):263–281.

Goodfellow I, Bengio Y, Courville A (2016) *Deep Learning* (MIT Press, Cambridge, MA).

Graham RL, Hell P (1985) On the history of the minimum spanning tree problem. *Ann. History Comput.* 7(1):43–57.

Graves SC, Willems SP (2000) Optimizing strategic safety stock placement in supply chains. *Manufacturing Service Oper. Management* 2(1):68–83.

Graves SC, Willems SP (2003) Supply chain design: Safety stock placement and supply chain configuration. *Supply Chain Management: Design, Coordination and Operation*, vol. 11 of Handbooks in Operations Research and Management Science (Elsevier, New York), 95–132.

Hall NG, Liu Z (2010) Capacity allocation and scheduling in supply chains. *Oper. Res.* 58(6):1711–1725.

Harris CR, Millman KJ, Van Der Walt SJ, Gommers R, Virtanen P, Cournapeau D, Wieser E, et al. (2020) Array programming with NumPy. *Nature* 585(7825):357–362.

Ho YC, Cao X, Cassandras C (1983) Infinitesimal and finite perturbation analysis for queueing networks. *Automatica J. IFAC* 19(4):439–445.

Humair S, Willems SP (2011) Optimizing strategic safety stock placement in general acyclic networks. *Oper. Res.* 59(3):781–787.

Humair S, Ruark JD, Tomlin B, Willems SP (2013) Incorporating stochastic lead times into the guaranteed service model of safety stock optimization. *Interfaces* 43(5):421–434.

Kosasih EE, Brintrup A (2021) Reinforcement learning provides a flexible approach for realistic supply chain safety stock optimisation. Preprint, submitted July 2, https://arxiv.org/abs/2107.00913.

Lesnaia E, Vasilescu I, Graves SC (2005) The complexity of safety stock placement in general-network supply chains. Working paper, Massachusetts Institute of Technology, Cambridge, MA.

Meinshausen N (2007) Relaxed lasso. *Comput. Statist. Data Anal.* 52(1):374–393.

Miao Y, Gowayyed M, Metze F (2015) EESEN: End-to-end speech recognition using deep RNN models and WFST-based decoding. *Proc. IEEE Workshop on Automatic Speech Recognition and Understanding* (IEEE, New York), 167–174.

Mikolov T, Karafiát M, Burget L, Černockỳ J, Khudanpur S (2010) Recurrent neural network based language model. Kobayashi T, Hirose K, Nakamura S, eds. *Proc. 11th Annual Conf. of the Internat. Speech Comm. Assoc.* (ISCA, Baixas, France), 1045–1048.

Nesterov Y (1983) A method of solving a convex programming problem with convergence rate $o(1/k^2)$. *Soviet Math. Doklady* 27:372–376.

Oroojlooyjadid A, Nazari M, Snyder LV, Takáč M (2022) A deep q-network for the beer game: Deep reinforcement learning for inventory optimization. *Manufacturing Service Oper. Managment* 24(1):285–304.

Parikh N, Boyd S (2014) Proximal algorithms. *Foundations Trends Optim.* 1(3):127–239.

Pirhooshyaran M, Snyder LV (2020) Simultaneous decision making for stochastic multi-echelon inventory optimization with deep neural networks as decision makers. Preprint, submitted June 10, https://arxiv.org/abs/2006.05608.

Robbins H, Monro S (1951) A stochastic approximation method. *Ann. Math. Statist.* 22(3):400–407.

Rosling K (1989) Optimal inventory policies for assembly systems under random demands. *Oper. Res.* 37(4):565–579.

Rumelhart DE, Hinton GE, Williams RJ (1986) Learning representations by back-propagating errors. *Nature* 323(6088): 533–536.

Salim A, Hachem W (2019) On the performance of the stochastic fista. Working paper, King Abdullah University of Science and Technology, Thuwal, Kingdom of Saudi Arabia.

Shor NZ (2012) *Minimization Methods for Non-Differentiable Functions*, vol. 3 (Springer Science & Business Media, New York).

Simpson KF (1958) In-process inventories. *Oper. Res.* 6(6):863–873.

Snyder LV, Shen ZJM (2019) *Fundamentals of Supply Chain Theory* (John Wiley & Sons, Hoboken, NJ).

Soltanolkotabi M, Javanmard A, Lee JD (2018) Theoretical insights into the optimization landscape of over-parameterized shallow neural networks. *IEEE Trans. Inform. Theory* 65(2):742–769.

Teodoro G, Sachetto R, Sertel O, Gurcan MN, Meira W, Catalyurek U, Ferreira R (2009) Coordinating the use of GPU and CPU for improving performance of compute intensive applications. *Proc. IEEE Internat. Conf. on Cluster Comput. and Workshops* (IEEE, New York), 1–10.

Tibshirani R (1996) Regression shrinkage and selection via the lasso. *J. Royal Statist. Soc. B* 58(1):267–288.

Ugander J, Karrer B, Backstrom L, Marlow C (2011) The anatomy of the facebook social graph. Preprint, submitted November 18, https://arxiv.org/abs/1111.4503.

Wainwright MJ (2009) Sharp thresholds for high-dimensional and noisy sparsity recovery using ℓ_1-constrained quadratic programming (lasso). *IEEE Trans. Inform. Theory* 55(5):2183–2202.

Wang XF, Li X, Chen GR (2012) *Network Science: An Introduction* (Higher Education Press, Beijing).

Werbos PJ (1990) Backpropagation through time: What it does and how to do it. *Proc. IEEE* 78(10):1550–1560.

Xiao L (2009) Dual averaging method for regularized stochastic learning and online optimization. *Adv. Neural Inform. Processing Systems* 22:2116–2124.

You Q, Jin H, Wang Z, Fang C, Luo J (2016) Image captioning with semantic attention. *Proc. IEEE Conf. on Comput. Vision and Pattern Recognition* (IEEE, New York), 4651–4659.