Operations Research
Vol. 54, No. 4, July–August 2006, pp. 725–742
ISSN 0030-364X | EISSN 1526-5463 | 06 | 5404 | 0725

DOI 10.1287/opre.1060.0313
© 2006 INFORMS

Optimizing Strategic Safety Stock Placement in Supply Chains with Clusters of Commonality

Salal Humair
Optiant, Incorporated, 4 Van de Graaff Drive, Burlington, Massachusetts 01803, salal.humair@optiant.com

Sean P. Willems
School of Management, Boston University, Boston, Massachusetts 02215, willems@bu.edu

Multiechelon inventory optimization is increasingly being applied by business users as new tools expand the class of network topologies that can be optimized. In this paper, we formalize a topology that we call networks with clusters of commonality (CoC), which captures a large class of real-world supply chains that contain component commonality. Viewed as a modified network, a CoC network is a spanning tree where the nodes in the modified network are themselves maximal bipartite subgraphs in the original network. We first present algorithms to identify these networks and then present a single-state-variable dynamic program for optimizing safety stock levels and locations. We next present two reformulations of the dynamic program that significantly reduce computational complexity while preserving the optimality of the resulting solution. This work both incorporates arbitrary safety stock cost functions and makes possible optimizing a large class of practically useful but previously intractable networks. It has been successfully applied at several Fortune 500 companies, including the recent Edelman finalist project at Hewlett Packard described in detail in Billington et al. (2004).

Subject classifications: multiechelon inventory system; safety stock optimization; dynamic programming application; component commonality; networks with clusters of commonality.
Area of review: Manufacturing, Service, and Supply Chain Operations.
History: Received August 2003; revisions received June 2004, March 2005; accepted June 2005.

1. Introduction

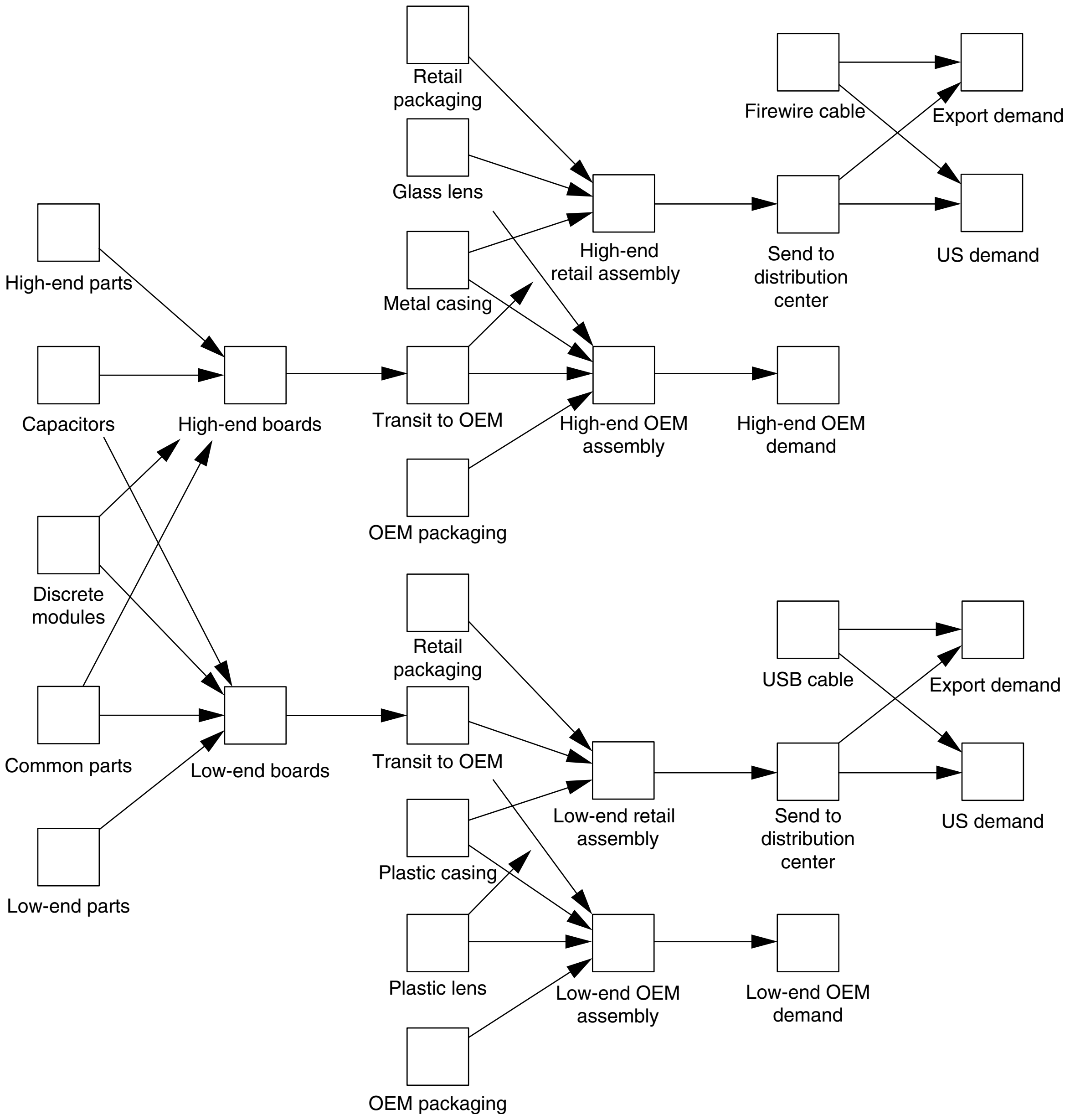
The past few years have seen a proliferation of successful multiechelon inventory optimization projects implemented by industry. There are two primary reasons for this. First, the past decade has seen a diffusion of supply chain knowledge throughout the organization; prime examples include the identification of shortfalls in supply chain planning (Lee and Billington 1992), and the identification of the importance of the bullwhip effect (Lee et al. 1997). This diffusion of knowledge has served as a catalyst for companies to put in place the people and infrastructure to analyze and model supply chains. Second, there has been significant advancement in the tools that business users can use to actually perform the analysis. Two recent finalists for the Edelman competition, Lin et al. (2000) and Billington et al. (2004), both present examples where multiechelon inventory theory was successfully distributed to a large set of business users within a company.

In the work summarized in Billington et al. (2004), multiechelon inventory optimization tools have been transferred to more than 100 business users within Hewlett Packard, producing savings in excess of $150 million. From a modeling perspective, the most critical success factor was allowing the user to "build what she can draw." That is, the model had to solve networks that the planner encountered in her daily life. The difficulty that arose was that the networks encountered in reality do not exhibit a standard network topology for which practical optimization algorithms are readily available; in particular, they are not assembly, distribution, or spanning trees. That said, it was also the case that the supply chains in question did possess special structure. For instance, the commonality is likely to be limited to adjacent echelons and the number of such echelons may itself be limited. Therefore, while the networks did not fit into a standard network classification, it was also the case that they were not completely arbitrary general networks.

Figure 1 depicts a supply chain that produces a high- and low-end consumer product that sells to both retail and OEM (Original Equipment Manufacturer) markets. The components of the high-end assembly are superior to the low-end offering; in particular, a glass lens is used in place of a plastic one and the casing is metal versus plastic. The circuit boards share many common components, but have processors with different speeds. The retail and OEM products differ only by their packaging. In summary, this is a very typical supply chain for a company that serves multiple markets and customer-demand classes. As can be seen from Figure 1, this supply chain does not correspond to any standard supply chain network topology.

The problem of interest for this paper is to obtain the optimal safety stock levels within the modeling framework of Graves and Willems (2000) for such networks, several real instances of which are discussed in Billington et al.

Figure 1. An example of a CoC network.



(2004). We refer to these chains that contain commonality only between adjacent echelons as *networks with clusters of commonality* (CoC). For manufacturing echelons, examples of commonality include parts or subassemblies shared among different items. For distribution echelons, commonality refers to different end items that are kitted together or bundled as assortments to customers. While a precise characterization will be presented in §4, CoC networks can be imagined as trees of bipartite subnetworks.

Our primary observation is that while optimizing safety stock locations and levels in arbitrary directed acyclic networks is hard, most practical supply chains do exhibit some special structure. Our main modeling contributions are (i) the formalization of a rich network structure that captures the majority of practically occurring supply chains, and (ii) the consideration of arbitrary stage-cost functions to capture real-world supply chain costs. Our main algorithmic contributions are (i) algorithms to efficiently characterize these networks, (ii) a dynamic program based on topological properties of these networks, and (iii) an algorithm to obtain the optimal safety stock levels and locations. Our algorithm reduces to manageable levels the complexity of the dynamic program, which can be prohibitive, and substantially increases the range of networks we can reasonably optimize.

The outline of this paper is as follows. The remainder of this section presents a brief literature review. Section 2 provides background on the type of multiechelon inventory problem addressed in this paper. It first introduces the major assumptions and the underlying inventory dynamics. Next, it summarizes the mathematical programming formulation and the spanning-tree dynamic program in Graves and Willems (2000). Section 2 serves as the building block for this paper's extension of the guaranteed-service model to arbitrary stage-cost functions and a more general supply chain network topology. Section 3 presents the need for considering arbitrary rather than only concave cost functions. In §4, we define CoC networks, outline the problems in extending the spanning-tree approach to more general networks, list some properties of CoC networks, and present an algorithm for identifying clusters in the network. In §5, we present the dynamic programming formulation for optimizing safety stock levels and locations in CoC networks with comments, illustrative examples, and a note on its complexity. Section 6 is devoted to two algorithmic improvements that significantly reduce the complexity of the dynamic program, extending the range of real-world chains that can be practically optimized. Section 7 summarizes the paper. Proofs for the correctness of the labeling algorithm and optimality of the dynamic program are included in the appendix.

1.1. Literature

We refer the reader to the literature review in Graves and Willems (2003b) for a larger discussion of how this research stream fits into the overall body of work on multiechelon inventory systems. In brief, literature that focuses on setting safety stock levels in real-world multiechelon systems subject to demand variability can be segmented according to how delivery (or service) times between adjacent stages are modeled. The works of Lee and Billington (1993), Glasserman and Tayur (1995), and Ettl et al. (2000) all develop stochastic-service models where the service time from a supplier will vary based on the supplier's material availability. As such, a customer stage needs to set its safety stock level explicitly considering the probability that a supplier will not be able to meet its quoted service time. An alternative research stream that began with a 1955 manuscript by Kimball, later reprinted (Kimball 1988), assumes a guaranteed-service model (GSM) where each stage quotes a service time it can meet with certainty. Optimizing safety stock levels and locations in the GSM is the focus of this paper.

Simpson (1958) formulated the GSM for serial-line networks. The works of Inderfurth (1991), Inderfurth and Minner (1998), and Graves and Willems (1996, 2000) extend Simpson's work to supply chains modeled as assembly networks, distribution networks, and spanning trees. For each network topology, a dynamic program is formulated to solve the resulting nonlinear-integer problem of minimizing total safety stock cost. The challenge is to determine an efficient approach to traverse the state space of the dynamic program. In the case of assembly (distribution) networks, the dynamic program can proceed with a forward (backward) pass through the network. When the network is a spanning tree, a more complex ordering is required where the nodes are solved as either forward or backward nodes depending on their orientation in the network. An attractive aspect of all these formulations is that they require only one state variable, which is the service time, either incoming or outgoing depending on the node's orientation in the network.

Minner (2000) does consider safety stock placement in general acyclic structures. In contrast to the formulation presented in this paper, Minner (2000) augments the state space of a node to consider all-or-nothing stocking policies at upstream or downstream nodes, depending on the node's relation to the subgraph containing commonality. Due to the combinatorial nature of the exact formulation, Minner (2000) investigates several heuristic approaches, finding that tabu search performs best.

While there is a small amount of research focused on GSMs, there is a vast body of literature on multiechelon inventory systems (MES) using stochastic-service concepts. A summary of these approaches can be found in Axsäter (2003), Song and Zipkin (2003), and de Kok and Fransoo (2003). In general, the focus of MES research is to discover structural insights into the workings of specialized network topologies that represent important components of more complicated systems. For instance, Clark and Scarf (1960) establishes the optimality of echelon base-stock policies for serial lines. The structural form of this optimal policy carries over to assembly networks, and to several relaxations of the original assumptions in Clark and Scarf (1960). However, computation of optimal policies is a significant issue even for serial lines, and for more complicated topologies, not only the computation but the exact form of the optimal policy remains a challenging question.

Similarly, assemble-to-order (ATO) systems under stochastic service have been analyzed for useful structural insights; c.f. Song and Zipkin (2003). These systems deal with common components used by multiple products, and are topologically subsets of the networks treated in this paper. Results from this area establish the optimality of base-stock policies for restricted forms of the problem and analytically prove when common components will require less inventory than dedicated components. In the general case, the approach is to use heuristics—for instance, using base-stock policies with some allocation rule to assign components to products.

The primary distinction between GSM and MES research involves the trade-off between exactness and network topology. MES rigorously accounts for time, whereas GSM looks at time more abstractly. For MES, requirements in a time period are satisfied or not, with a resulting impact on inventory position, production requirements, and fill rate for that time period. This level of accounting is not present in GSM models. Instead, inventory levels are designed to meet requirements over an interval without accounting for whether all constraints within the interval are satisfied. The same situation applies to demand. A significant amount of complexity in MES work is derived from the need to account for the case where demand is not met; in effect, the challenge in MES work is dealing with the unfill rate. In GSM work, this is obviated by casting the problem in terms of requirements over an interval. There are pros and cons to both approaches. MES work allows for a more exact system understanding that can serve as the building block for more complex systems. GSM work models the entire system, which allows a planner to make tactical decisions without the need to approximate portions of the system that are not captured by a simplified topological representation. In networks where most commonality exists between only two echelons, the richness of the MES models may outweigh the value of modeling the entire network in an approximate fashion.

2. Background

This section presents a review of the modeling assumptions, inventory dynamics, safety stock cost function, mathematical programming formulation, and solution technique for the guaranteed-service stream of research. The formulation in this paper will follow the notation in Graves and Willems (2000, 2003a), which we will hereafter refer to as GW. We begin with an overview of the model's assumptions in §2.1. The inventory dynamics and the safety stock cost for a single stage are illustrated in §§2.2 and 2.3. The mathematical programming formulation is presented in §2.4, and the central ideas of the dynamic program are reviewed in §2.5.

2.1. Model Assumptions

The supply chain is represented as a network. The nodes in the network correspond to stages in the supply chain and arcs denote the precedence relationship between stages. A stage represents a processing or transformation activity in the supply chain. Depending on the scope and granularity of the analysis being performed, the stage could represent anything from a single step in a manufacturing or distribution process to a collection of such steps to the entire process itself. Regardless of the choice of scope and granularity, a stage is a candidate location for the placement of a safety stock of inventory. In this framework, safety stocks are strategic in the sense that they can decouple stages in the supply chain from one another.

Each stage in the supply chain operates according to a base-stock policy. In each period, the stage observes demand and places a replenishment order on its suppliers equal to the observed demand. Each stage also has a deterministic processing time (or lead time) that includes all of the time required to transform its inputs to its outputs. Once all of the stage's required inputs are available, the processing time includes any waiting time, manufacturing time, and transportation time at the stage.

Service time is the amount of time that elapses between when a downstream stage places an order on an upstream stage and when the order is delivered by the upstream stage to the downstream stage. Each stage in the supply chain quotes a service time to its adjacent downstream (customer) stages, and it is quoted service times from its adjacent upstream (supplier) stages. We define the service time that a stage quotes its customers as the stage's outgoing service time and the maximum service time quoted to the stage by its adjacent supplier stages as the stage's incoming service time.

Each stage sets its base stock so as to guarantee that it can meet its outgoing service time to its customers. That is, each stage will quote a guaranteed outgoing service time to its downstream customers, who know that this commitment will be met with certainty. Stages that have no adjacent downstream customers serve external customers; these stages are denoted as demand stages. The maximum allowable outgoing service time for demand stages, or alternatively, the amount of time that an outside customer will wait for delivery, is an input to the model. The outgoing service time for all other stages is a decision variable, and we will see in §2.2 how service times dictate the safety stock requirements at each stage.

GW specify a function for each stage that represents the maximum demand over any interval of time for which the stage's outgoing service time is guaranteed. Such a maximum demand function does not imply that demand can never exceed the bound, but for the purposes of setting safety stocks in the supply chain, only inventory-stocking policies that satisfy demand realizations within the demand bound are considered. When demand exceeds the bound, then the safety stock in the system will not be adequate to assure the outgoing service times. In this case of extraordinary demand, some correspondingly extraordinary measures are taken to augment the safety stock so that the demand can be served within the specified outgoing service times. Alternatively, one might view demand in excess of this bound as being lost or somehow being served from another source. In effect, the GSM assumes that other countermeasures beyond just safety stock placement are used to handle the spectrum of variability a supply chain faces. In the GSM, the safety stock will handle a predefined portion of the variability, but additional countermeasures like expediting, spot-market purchases, contracted overflow capacity, and forgoing sales, will be employed when demand exceeds the bound.

2.2. Single-Stage Inventory Dynamics

For the GSM, the net replenishment time drives base-stock requirements. If we denote the processing time at stage i by T_i, the incoming service time by SI_i, the outgoing service time by S_i, and the net replenishment time by τ_i, then $\tau_i = SI_i + T_i - S_i$. Net replenishment time reflects the amount of time a stage is responsible for covering changes in demand from inventory.

Suppose that stage i starts at time 0 with initial inventory $I_i(0)$ equal to the stage's base stock B_i. Given the assumptions of guaranteed service and the definition of the service times, the inventory at time t, $I_i(t)$, equals the following, where $d_i(t)$ denotes the demand in period t:

$$I_i(t) = B_i - \sum_{v=0}^{t-S_i} d_i(v) + \sum_{w=0}^{t-T_i-SI_i} d_i(w)$$
$$= B_i - \sum_{v=t-T_i-SI_i+1}^{t-S_i} d_i(v). \quad (1)$$

The above equation follows from the fact that in period t, stage i completes into its inventory the replenishment order that was placed in period $t - T_i - SI_i$. Correspondingly, in period t, stage i must serve the replenishment orders placed by its customers in period $t - S_i$.

To satisfy the service-time guarantee, the base stock B_i is set so that the inventory on hand, $I_i(t)$, is always nonnegative. That is, one wants to set

$$B_i \geqslant \sum_{v=t-T_i-SI_i+1}^{t-S_i} d_i(v). \quad (2)$$

In words, the base stock should equal (or exceed) the maximum possible demand for which one wants to satisfy all variability from safety stock that can occur over the stage's net replenishment time. However, given the assumption of a bounded demand process, we can define $D_i(x)$ as the maximum demand at stage i over a time interval of length x. Then, to assure that Equation (2) holds for all t, we need to set $B_i = D_i(SI_i + T_i - S_i) = D_i(\tau_i)$, which ties the base-stock requirements to the net replenishment time.

2.3. Stage-Cost Functions

Stage i's cost function $c_i(SI_i, S_i)$ captures the safety stock cost incurred at stage i if it has to cover a net replenishment time equal to $SI_i + T_i - S_i$. In GW and the other cited papers addressing the GSM, $c_i(SI_i, S_i)$ is assumed to be concave in S_i and SI_i; in the case of GW, this assumption is an artifact of the functional form assumed for the demand bound and the per-unit linear holding cost. To make the formulation concrete, if C_i denotes the per-unit inventory holding cost at stage i and μ_i the average demand per period, then $c_i(SI_i, S_i)$ can be represented as $c_i(SI_i, S_i) = C_i(D_i(SI_i + T_i - S_i) - \mu_i(SI_i + T_i - S_i))$. That is, the safety stock cost at stage i is the cost of the maximum demand over the net replenishment time minus the expected demand over that same time frame.

If one further assumes that the per-period demand is independent and normally distributed with standard deviation σ_i, one can choose a number z_i to reflect the percentage of time that safety stock covers demand variation, and write $c_i(SI_i, S_i) = C_i(z_i\sigma_i\sqrt{SI_i + T_i - S_i} + \mu_i(SI_i + T_i - S_i) - \mu_i(SI_i + T_i - S_i)) = C_i z_i \sigma_i \sqrt{SI_i + T_i - S_i}$.

2.4. The Mathematical Programming Formulation

Let graph $G = (V, \mathcal{A})$ represent a directed network where V is the set of all nodes and $\mathcal{A}$ is the set of all arcs. Let $V_d \subset V$ be the set of demand nodes and $V_s \subset V$ be the set of supply nodes (defined as those nodes with no incoming arcs in the graph). Assume that the network is acyclic as well as connected. The decision variables for the model, as discussed above, are SI_i and S_i, where SI_i is node i's incoming service time and S_i is node i's outgoing service time. The maximum outgoing service time a demand node i can quote is denoted by E_i. Then, assuming T_i, SI_i, S_i, and E_i are integral and $|V| = n$, the problem of interest **P** is as follows:

$$\begin{aligned}
\mathbf{P}: \quad \min \quad & \sum_{i=1}^{n} c_i(SI_i, S_i) \\
\text{s.t.} \quad & S_i - SI_i \leqslant T_i, \quad i = 1, \ldots, n, \\
& S_j - SI_i \leqslant 0, \quad \forall (j, i) \in \mathcal{A}, \\
& S_i \leqslant E_i, \quad \forall i \in V_d, \\
& SI_i = 0, \quad \forall i \in V_s, \\
& S_i, SI_i \geqslant 0, \text{ integral}, \quad i = 1, \ldots, n.
\end{aligned}$$

The constraints of **P** simply ensure that the outgoing service time at each node does not exceed the sum of its incoming service time and processing time, the incoming service time for node i is no less than the maximum outgoing service times quoted by the nodes directly supplying i, and the outgoing service times to end-item customers are bounded by the E_is. We refer the reader to GW, Graves and Willems (2003b), and Billington et al. (2004) for examples of how this model has been applied in practice.

2.5. The Dynamic Program for Spanning Trees

When the network is a spanning tree, two properties of **P** are central to its reformulation as a dynamic program.

(1) It is possible to order the nodes such that each node is assigned a unique index, and every node except one (which is called the *root* node) has at most one adjacent node with a higher-valued index than its own. We call the higher-index node adjacent to a node its *parent* and all adjacent nodes with lower indices its *children*; note that a node's children can be upstream or downstream to the node. We define a *child subgraph* as all nodes reachable from a node through one of its children. The child subgraphs for spanning trees are trees themselves, and the child subgraphs are disjoint, i.e., they share no nodes or arcs. For spanning trees, we use the term *child subtree* instead of child subgraph to distinguish it from later cases in this paper.

(2) Once the parent's decision variables are fixed, the constraints between the child subtrees are separable and the optimal cost-to-go functions of the child subtrees are additive. This is a direct consequence of the spanning-tree property because the only path connecting the child subtrees is through the parent itself. This property is key to the dynamic programming formulation for spanning trees.

Let $k = 1, \ldots, n$ be the index assigned to each node and reindex all problem parameters in terms of k—i.e., assume from now on that whenever S_k, SI_k, T_k, E_k, and c_k are used or any new node-specific notation defined, it refers to a node with index k, not the original node label i used for the node in §2.4. The arc set $\mathcal{A}$ is similarly updated to use the new indices, so (i, j) refers to an arc between nodes with indices i and j. Let p_k be the index of node k's parent. Then, GW define the following two optimal cost-to-go functions: If node k has a directed arc from it to its parent, i.e., $(k, p_k) \in \mathcal{A}$, $f_k(S)$ is the optimal cost of the subnetwork including k and its child subtrees if k were to quote an outgoing service time S; if node k has a directed arc to it from its parent, i.e., $(p_k, k) \in \mathcal{A}$, $g_k(SI)$ is the optimal cost of the subnetwork including k and its child subtrees when the incoming service time to k is SI. The cost-to-go function of the root node $(k = n)$ is defined as $g_n(SI)$ by convention.

The following dynamic program **SDP** is then a reformulation of **P**. Here $(S - T_k)^+$ denotes $\max(0, S - T_k)$ and M_k is the largest possible service time node k can quote (defined below).

If $(k, p_k) \in \mathcal{A}$ for $S = 0, \ldots, M_k$,

$$f_k(S) = \min_{SI:\,(S-T_k)^+ \leqslant SI \leqslant M_k - T_k} \left\{ c_k(SI, S) + \sum_{\substack{(i,k)\in\mathcal{A} \\ i<k}} \min_{x_i:\, x_i \leqslant SI} f_i(x_i) + \sum_{\substack{(k,j)\in\mathcal{A} \\ j<k}} \min_{y_j:\, y_j \geqslant S} g_j(y_j) \right\}.$$

If $(p_k, k) \in \mathcal{A}$, and letting $E_k = \infty$ if $k \notin V_d$ for $SI = 0, \ldots, M_k - T_k$,

$$g_k(SI) = \min_{S:\, 0 \leqslant S \leqslant \min(SI + T_k, E_k)} \left\{ c_k(SI, S) + \sum_{\substack{(i,k)\in\mathcal{A} \\ i<k}} \min_{x_i:\, x_i \leqslant SI} f_i(x_i) + \sum_{\substack{(k,j)\in\mathcal{A} \\ j<k}} \min_{y_j:\, y_j \geqslant S} g_j(y_j) \right\}.$$

Some comments and observations on **SDP** follow:

- The optimal cost-to-go functions are evaluated in the order $k = 1, \ldots, n$, guaranteeing that when node k needs to be evaluated, all cost-to-go functions for its children are available.
- M_k is the longest processing-time path to node k including it own processing time, and is found by the recursion $M_k = T_k + \max_{j:\,(j,k)\in A} \{M_j\}$.
- The optimal cost for **P** can be obtained by minimizing $g_n(SI)$ over $SI = 0, \ldots, M_n - T_n$.
- It is relatively straightforward to evaluate the functions f_k and g_k in $O(M_k^2)$ operations, even though the recursions seem to suggest $O(M_k^3)$, by intelligently minimizing the expression inside the braces. Letting $m = \max_k(M_k)$, the overall complexity of **SDP** is bounded by $O(nm^2)$.
- The optimal policy for node k is defined as the function $\mu_k(S)$ if $f_k(S)$ is evaluated and the function $\nu_k(SI)$ if $g_k(SI)$ is evaluated. $\mu_k(S)$ maps S to the set (SI^*, x_i^*, y_j^*). Here SI^* is the optimal incoming service time for k if it quotes service time S to its downstream stages; x_i^* is the optimal service time for each adjacent upstream stage (excluding the parent) to k, and y_j^* is the optimal incoming service time assumed by each adjacent (excluding the parent) downstream stage. $\mu_k(S)$ results in the minimum value of the expression on the right-hand side for $f_k(S)$. An exact analogue for $\nu_k(SI)$ is true that maps SI to (S^*, x_i^*, y_j^*), the optimal values of decision variables for k and adjacent nonparent nodes that result in the minimum value of the expression on the right-hand side for $g_k(SI)$. These functions allow us to backtrack and obtain the optimal service times for the entire tree once the minimum of **P** has been obtained from $g_n(SI)$.

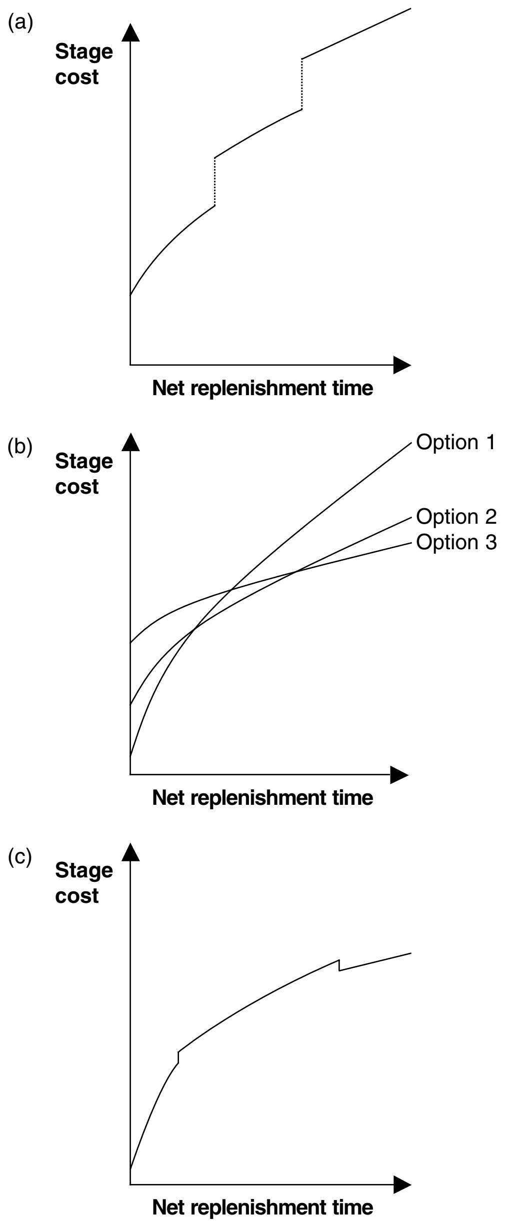
3. Arbitrary Stage-Cost Functions

In this paper, we let $c_i(SI_i, S_i)$ be arbitrary functions of SI_i and S_i. Allowing arbitrary stage-cost functions for optimization becomes necessary when adding real-world modeling enhancements to the problem formulation **P**. For instance, some extensions to the basic GW framework incorporate stochastic stage-processing times, batch ordering, stage capacities, and correlated demand. Most such extensions result in stage-cost functions that, although easily computable, are not well structured and sometimes not even closed form for a given SI_i and S_i. Further, these extensions are usually not esoteric, and are cited as a major contributor to success in Billington et al. (2004). The practical value of optimizing networks involving more general cost functions is therefore significant. Our work can be used to optimize any CoC network as long as $c_i(SI_i, S_i)$ depend only on stage i's decision variables, regardless of the specific structure of the costs themselves. While discussing the treatment of stochastic stage-processing times or order batching requires the introduction of notation that is beyond the scope of this paper, a simple yet prevalent example of arbitrary stage cost involves capturing the fixed costs associated with holding inventory at a stage. We will see that these examples break either the concavity or the monotonicity assumption of the cost functions.

The first example involves a single stocking location that incurs a fixed cost to expand available storage capacity. Recall from §2.2 that the amount of safety stock required at a stage is a function of the net replenishment time at the stage. These fixed costs could involve the purchase of additional equipment, the hiring of additional workers, or the incremental expansion of the warehouse's capacity. Graphically, this discontinuous cost function is presented in Figure 2(a).

When there are multiple possible storage locations for safety stock, it is necessary to consider the fixed and

Figure 2. Example of nonconcave, piecewise concave, and nonmonotone cost functions.



Notes. In (a), the cost function experiences discontinuous upward jumps. In (b), the cost function is the lower envelope of three concave cost functions. In (c), the cost function reflects the cost structure of (b) with capacity restrictions imposed on each option.

variable costs for the various storage options. Figure 2(b) shows a stage's inventory cost function as a function of net replenishment time when there are three storage options: a temporary offsite location that has a low fixed cost but a high variable cost, a shared offsite facility that has a higher fixed cost but a lower per-unit cost, and a dedicated offsite storage facility that has the highest fixed cost and lowest variable cost. The safety stock cost at the stage equals the lower envelope of these three cost functions, which is piecewise concave.

If we add the real-world constraint that each of the options mentioned in Figure 2(b) either has capacity limitations or is only available if a commitment is made to hold a certain amount of stock at the location, then the cost function itself can be nonconcave, discontinuous, and nonmonotone as demonstrated in Figure 2(c). Real-world examples of this inventory cost structure include supply chains that rely on third-party logistics providers to coordinate inventory and shipping between locations, consumer packaged goods companies that store finished goods in warehouses close to customer locations, and raw material hubs colocated with manufacturing sites.

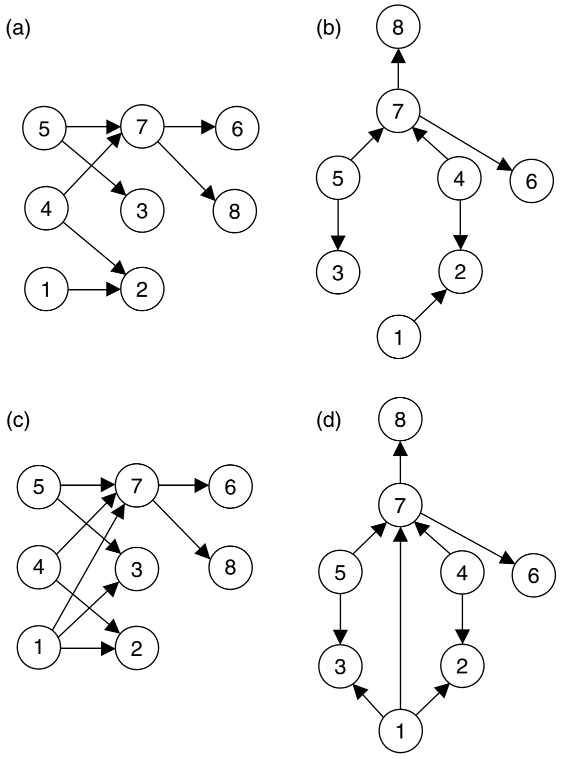
In terms of the mathematical program and resulting dynamic programming formulation, arbitrary stage costs have two major ramifications. First, because we no longer have concavity, the optimal solution may no longer be an extreme point of **P**. Therefore, the all-or-nothing stocking policies considered in previous guaranteed-service work need not hold in this paper. Second, we explicitly require that the feasible region be contained in a bounded region, i.e., $SI_i \leqslant M_i - T_i$ for all i. GW do not require that the feasible region for **P** be bounded because the structure of their cost functions guarantees that the optimal solution exists in a bounded region given by $SI_i \leqslant M_i - T_i$. To see why we require a boundedness assumption, consider the functions $c_i(S_i, SI_i) = e^{-SI_i}$, and note that the optimal solution cannot be bounded. However, these examples are esoteric and, in fact, realistic modeling enhancements almost always result in the service times being contained in a bounded region. To maintain consistency with previous guaranteed-service papers, we keep our bounds consistent with GW.

4. Networks with Clusters of Commonality

Here we make precise the definition of networks with clusters of commonality and list their topological properties relevant for the dynamic programming formulation presented in §5. We define a cluster as any directed bipartite subgraph G' of G that is maximal, and between every pair of nodes in G', there exists more than one undirected path within G'. A bipartite subgraph is considered *maximal* if and only if adding any node $k \in G \backslash G'$ to G', along with its arcs that go to nodes in G', results in at least one pair of nodes in the resulting subgraph having only one undirected path between them. We define a *network with clusters of commonality* (CoC network) as a network comprised of disjoint clusters and singleton nodes (nodes that are not part of any cluster) such that a modified network consisting of singleton nodes and each cluster aggregated as a node forms a spanning tree.

To motivate the above definition of clusters, we first note the difficulty in extending the spanning-tree dynamic programming approach to more general networks. We then present algorithms to recognize a CoC network and identify the clusters in the network.

Figure 3. Illustration of the difficulty in formulating a dynamic program for general networks.



4.1. Difficulty in Extending the Spanning-Tree Approach

Note that the formulation **SDP** in §2.5 is only possible because once the incoming service time SI and outgoing service time S are fixed for a node, the constraints of its child subtrees are separable and their optimal cost-to-go functions are additive. Consider Figure 3(a), for instance; Figure 3(b) is just a graphical depiction of Figure 3(a) hung from its root node. At node 7, the recursion for every SI_7 and S_7 constrains the outgoing service times for nodes 4 and 5 to be less than or equal to SI_7 and constrains the incoming service time for nodes 6 and 8 to be greater than or equal to S_7. The optimal costs and the associated optimal decision variables for child subtrees rooted at nodes 4 and 5 do not affect each other. Node 7's values for SI and S effectively decouple the child subtrees from each other and allow obtaining the optimal combination of service times for all nodes in the subtrees.

This structure breaks down when there are additional paths between children that do not pass through the parent. Now consider Figures 3(c) and 3(d), which are just Figure 3(a) with two additional arcs: $(1, 3)$ and $(1, 7)$. At node 7, there is now a path between nodes 4 and 5 that does not pass through node 7. In particular, SI_7 now constrains the maximum possible outgoing service time for node 1. Therefore, the cost for node 1 cannot be included in the optimal cost for node 4 because the optimal combination of service times reported by node 4 for subtree 1, 2, 4 may not satisfy the constraint imposed by node 7 on node 1. However, if the cost of node 1 must be taken into account separately from node 4, then the cost from node 2 cannot be included in the cost of node 4 and the cost of node 3 cannot be included in the cost of node 5 because the cost and the associated outgoing service time reported by node 1 changes the incoming service time for nodes 2 and 3, and the previously optimal outgoing service times from nodes 4 and 5 may violate this new constraint imposed by node 1.

This figure tells most of the story regarding the difficulty in applying the standard spanning-tree approach to more general network topologies. While this is a simple illustration, the problem is general. In effect, whenever there is a group of nodes such as nodes 1, 2, 3, 4, 5, and 7 in Figure 3(c), obtaining the optimal cost function at their root—in this case node 7—requires minimizing the sum of the cost-to-go functions of the other nodes—in this case nodes 1, 2, 3, 4, 5—over all feasible combinations of their incoming and outgoing service times—S_1, SI_2, SI_3, S_4, S_5—because the choice of any of the variables in the set influences the feasible range of the others. This coupling of the service-time constraints is the primary difficulty in formulating a dynamic program for CoC networks.

4.2. Labeling the Nodes

We first use a breadth-first traversal to label the nodes that will help us formulate a dynamic program for CoC networks. Choose some node $i \in V_d$, and call it the *root node*. Then, assign two integers (the *relative depth* and the *index*) plus the *parent's index* to each node as follows.

(1) For the root node, assign 0 for the relative depth and n for the index where n is the number of nodes in the network. Set an index counter $\bar{k} = n - 1$. Insert the root node into a FIFO queue of nodes that have been assigned integers.

(2) If the queue is empty, return. The algorithm has assigned index values to all nodes.

(3) Remove a node from the queue, call it i, and let d and k be its assigned values for the relative depth and index. To every node j adjacent to i that has not yet been assigned an index, insert j into the queue after carrying out the following three steps.

(a) Assign $d-1$ if $(i, j) \in \mathcal{A}$ or $d+1$ if $(j, i) \in \mathcal{A}$ as j's relative depth.

(b) Assign $\bar{k}$ as j's index and decrement $\bar{k}$ by 1.

(c) Assign i as j's parent by setting $p_j = k$.

(4) Go to 2.

Intuitively, relative depth is the net number of forward arcs to the root node, on the path through the parent nodes, and is needed for identifying clusters. Depending on which node is picked as the root, this could be positive or negative. The index value serves the same purpose as in GW, providing a node ordering that allows the network to be optimized with a single iteration through the nodes. Given these labels, the following properties of CoC networks are relatively easy to see.

(1) The minimum size of a cluster must be four because no bipartite subgraph of smaller size can have enough arcs to contain multiple undirected paths between nodes.

(2) All nodes in a cluster have one of two relative depth values. For ease of exposition, we will define each node in a cluster as being a *forward* or *backward* node where the forward nodes are those nodes that have a relative depth one lower than the backward nodes. Visually, the forward nodes form the head of the arcs in the cluster and the backward nodes form the tail of the arcs in the cluster.

(3) If the cluster does not contain the root node, then the cluster has exactly one node with a parent outside the cluster. We call this node the *cluster root* and its parent the *cluster parent*. All other nodes in the cluster have a parent that is in the cluster. This follows from Lemma A.1.

(4) If a cluster has a cluster parent, then there are four possible orientations for the cluster root and cluster parent, which are graphically presented in Figure 4. The cluster root can be either a forward or backward node and the cluster parent can have either an incoming or outgoing arc to the cluster root.

(5) All possible undirected paths between any node i and the root node must produce the same value of the relative depth at that node i. This is stated formally in Lemma 4.2.

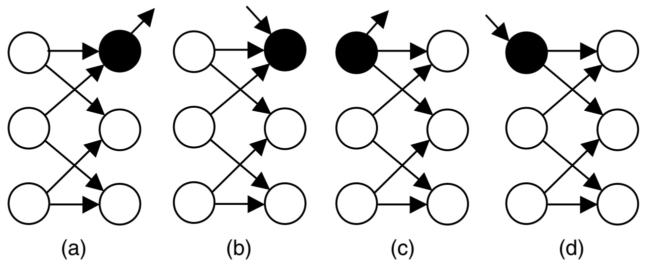
4.3. Labeling the Clusters of Commonality

We present an algorithm to determine if a given directed network satisfies the definition of CoC networks and, if so, identify all clusters in the network.

First, we need the notion of an echelon of commonality (EOC), which is the potential location for a cluster. An EOC is a subgraph that contains all nodes having the same relative depth, say d, and all nodes with relative depth $d+1$, e.g., a directed bipartite network has one EOC that is the entire network. An EOC need not be connected, but a cluster can only reside within an EOC, and an EOC can contain multiple clusters. For a CoC network, if no EOCs have clusters, then the network is a spanning tree; otherwise, we need to identify the clusters within the EOCs.

Given the labeling algorithm of §4.2 and a directed network, the following algorithm outputs FALSE if the network is not a CoC network, and TRUE otherwise, along with an assignment of a unique cluster number to each node that is in a cluster.

Figure 4. The four possible network orientations for a cluster root (colored black) and its parent.



Note. (a) and (b) illustrate a forward cluster and (c) and (d) illustrate a backward cluster.

(1) Verify that for all arcs $(i, j) \in \mathcal{A}$, the relative depth of node i equals the relative depth of node j plus 1. If not, return FALSE.

(2) Set a global cluster number $\bar{x} = 1$. Let l_k be a list of cluster numbers for each node k in the network and set $l_k = \varnothing$.

(3) For each EOC in the network, call routines $\mathcal{R}_1$, $\mathcal{R}_2$, and $\mathcal{R}_3$ below, in order. $\mathcal{R}_1$ assigns the value one to arc (i, j) if i and j belong to the same cluster and zero if it does not. $\mathcal{R}_2$ then assigns a unique cluster number to each node and determines the size of possible clusters. $\mathcal{R}_3$ inserts these new cluster numbers into the cluster lists l_k if needed. Let $\mathcal{P}$ denote the set of all arcs in the EOC under consideration.

```
$\mathcal{R}_1$: For each arc $(i, j)$ in $\mathcal{P}$:
    (i) Remove $(i, j)$ from the EOC.
    (ii) With $(i, j)$ removed, find an undirected path from $i$ to $j$ within the EOC.
    (iii) If a path exists, assign one to the arc $(i, j)$. Otherwise, assign zero.
$\mathcal{R}_2$: Let $x_i$ be a temporary cluster number associated with each node $i$ in the EOC, and let $s_{x_i}$ denote the size of the cluster with number $x_i$. Initialize $x_i = 0$ for all nodes $i$ in the EOC. Then, for each node $i$ in the EOC:
    (i) If $x_i \neq 0$, do nothing and move to the next node. If all nodes have $x_i \neq 0$, stop.
    (ii) Otherwise, set $x_i = \bar{x}$, $s_{x_i} = 1$, and increment $\bar{x}$ by 1.
    (iii) Set $q = i$, if every pair $(q, j)$ or $(j, q) \in \mathcal{P}$ that has been assigned one has $x_j \neq 0$, stop and go to step (i) of $\mathcal{R}_2$. Otherwise, for each pair $(q, j)$ or $(j, q) \in \mathcal{P}$ that has been assigned one, if $x_j = 0$, set $x_j = x_q$, set $s_{x_q} = s_{x_q} + 1$, and repeat this step by setting $q = j$.
$\mathcal{R}_3$: For each node $i$ in the EOC, if $s_{x_i} > 1$, insert $x_i$ into $l_i$.
```

(4) If any list l_k contains more than one element, return FALSE. The network contains clusters that share nodes.

(5) Otherwise, let n_c be the total number of distinct cluster numbers in all lists l_k plus singleton nodes (nodes with $l_k = \varnothing$). Let n_a be the total number of arcs for which both nodes are not in the same cluster.

(a) If $n_a < n_c - 1$ or $n_a > n_c - 1$, return FALSE. In the first case, the network is not connected, and in the second case, it contains a cycle between clusters/nodes.

(b) Otherwise, return TRUE and the lists l_k. The network is a CoC network, and all clusters have been identified.

The following lemmas show the correctness of the labeling algorithms in identifying CoC networks, where d_i is the relative depth of node i. The first lemma shows that even though the labels produced by the algorithm of §4.2 are not unique because the order of picking adjacent nodes in Step 3 is not specified, every labeling produced has the properties we desire. We state the second lemma without proof because it is relatively easy to see.

Lemma 4.1. *For any connected directed acyclic G and a fixed root node $k \in V_d$, either every labeling produced by the algorithm of* §4.2 *results in assigning the relative depth $d_i = d_j + 1$ for all arcs $(i, j) \in \mathcal{A}$, or every labeling results in some arc $(i, j) \in \mathcal{A}$ such that $d_i \neq d_j + 1$.*

Proof. Suppose that some labeling results in all $(i, j) \in \mathcal{A}$ having the property that $d_i = d_j + 1$. Denote the network's minimum depth $\underline{d} = \min_{i\in V} d_i$ and the maximum depth $\bar{d} = \max_{i\in V} d_i$, and partition V into sets of nodes V_r, $r = \underline{d}, \ldots, \bar{d}$, such that $i \in V_r \Leftrightarrow d_i = r$. Now, assuming that $\underline{d} \neq \bar{d}$, note that the network only contains directed arcs going from nodes in V_{r+1} to V_r for all $r = \underline{d}, \ldots, \bar{d}-1$. This makes it easy to see that for every path between any node $j \in V_r$ and the root node i, the net number of forward arcs must remain r. Therefore, no matter how the adjacent nodes are picked in Step 3 of the labeling algorithm, the labeling algorithm will assign the same depth r to j. However, if that is the case, then after running that algorithm, j will always remain in V_r. This is true of all nodes, which means that all arcs will remain between V_{r+1} and V_r.

The converse of this is the phrase that if some labeling results in some arc $(i, j) \in \mathcal{A}$ such that $d_i \neq d_j + 1$, then every labeling must result in some such arc. $\square$

Lemma 4.2. *If G is a CoC network, then for a fixed $i \in V_d$, every possible run of the labeling algorithm of* §4.2 *results in assigning the relative depth $d_i = d_j + 1$ for all arcs $(i, j) \in \mathcal{A}$.*

Lemma 4.2 implies that a directed acyclic graph can be a CoC network only if the labeling algorithm of §4.2 results in relative depths $d_i = d_j + 1$ for all arcs $(i, j) \in \mathcal{A}$. Therefore, Step 1 of our cluster-labeling algorithm above correctly eliminates from consideration all networks that cannot be CoC networks. The proof of Lemma 4.1 demonstrates that the labeling algorithm §4.2 separates the network into EOCs which themselves are directed bipartite subgraphs in the given network. Then, Step 3 simply determines the maximal subgraphs within each EOC, and Steps 4 and 5 determine if the graph is a spanning tree of clusters and singleton nodes.

4.4. Notation for Clusters of Commonality

We adopt the following notation for CoC networks:

C_k: The subgraph of G containing all nodes that are in the same cluster as k. For convenience, we will treat single nodes as clusters as well, i.e., whenever k is not part of a cluster with size $\geqslant 4$, $C_k = \{k\}$ and k is the cluster root for C_k. To distinguish these singleton clusters from clusters of size $\geqslant 4$, we call them *trivial clusters*.

C_k^-: The set of all nodes in cluster C_k, not including the cluster root, that have the lower relative depth. Note that this set is empty if C_k contains only k. Also note that C_k^- is a set of nodes, while C_k is a subgraph.

C_k^+: The set of all nodes in cluster C_k, not including the cluster root, that have the higher relative depth; i.e., if the nodes in C_k^- have a relative depth of d, then the nodes in C_k^+ have a relative depth of $d + 1$. This set is empty if C_k contains only k.

Finally, we need to define sets of nodes similar to those implicit in the **SDP** formulation from GW (c.f. §2.5). In GW, sets of nodes N_k are defined as the union of node k and all subtrees rooted at k that do not include p_k. These sets are important because the optimal cost-to-go function for node k captures the costs of all nodes in N_k.

We define analogous sets N_k for CoC networks with some modifications. If node k is a trivial cluster, the definition remains unchanged from GW except that N_k is now the union of k and all its child subgraphs instead of child subtrees. If node k is in a nontrivial cluster but not its cluster root, N_k consists of k and all child subgraphs which do not contain any node in the same cluster as k. If node k is in a nontrivial cluster and is its root, N_k represents all nodes connected to k by some undirected path that does not pass through k's parent. This allows the optimal cost-to-go function of the cluster root to capture the cost of all nodes within the cluster plus any other child subgraphs of the cluster root. Lemma A.2 shows that the following sets can be constructed for CoC networks:

$$N_k = \{k\} \bigcup_{\substack{i<k\\ i\notin C_k\\ (i,k)\in\mathcal{A}}} N_i \bigcup_{\substack{j<k\\ j\notin C_k\\ (k,j)\in\mathcal{A}}} N_j \begin{cases} \displaystyle\bigcup_{i\in C_k^+} N_i \bigcup_{j\in C_k^-} N_j & \\ \qquad \text{if } k \text{ is a cluster root}, \\ \cup\varnothing \quad \text{otherwise}. \end{cases}$$

Because the cluster root has a higher index than all other nodes in the cluster, our definition of N_k also allows us to maintain the desirable property that a single pass through the nodes can optimize the network. In short, whenever a cluster root is reached, all subnetworks for which it can report an optimal cost-to-go are already solved.

5. A Dynamic Program for Networks with Clusters of Commonality

A dynamic programming formulation can now be described for CoC networks. As in the case for spanning trees, the dynamic program still has two forms for the optimal cost-to-go functions, depending on the node's orientation to its parent: $f_k(S)$ is the minimum cost of the subnetwork N_k if k were to quote an outgoing service time S; and $g_k(SI)$ is the minimum cost of the subnetwork N_k when the incoming service time to k is SI. In particular, our formulation retains the very attractive property that the dynamic program involves only a single-state variable for each node.

While the high-level description of $f_k(S)$ and $g_k(SI)$ are identical to GW, there are some substantive differences in the dynamic programming formulation between spanning trees and CoC networks. Our dynamic program encounters three types of nodes: singleton nodes, cluster nodes that are not the cluster root, and cluster roots of nontrivial clusters. For singleton nodes, the functional equations remain as before. For all cluster nodes excluding the cluster root, the functional equations remain as before except that we need to explicitly exclude adjacent nodes that are in the same cluster. For cluster roots, the functional equations include the cost-to-go of all cluster nodes and all other adjacent subgraphs excluding its parent. Thus, the functional equations change depending on the node type. Nodes that are cluster roots need to perform a more complex optimization to fully capture the optimal cost-to-go for their adjacent subgraphs.

The impact of these differences is made clear in the following dynamic program **DP**, which reformulates **P** (c.f. §2.4) for a CoC network:

If $(k, p_k) \in \mathcal{A}$, for $S = 0, \ldots, M_k$,

$$f_k(S) = \min_{SI:\,(S-T_k)^+ \leqslant SI \leqslant M_k - T_k} \{c_k(SI, S) + c'_k(SI, S) + h_k^+(S) + h_k^-(SI)\}. \quad (3)$$

If $(p_k, k) \in \mathcal{A}$, and letting $E_k = \infty$ if $k \notin V_d$, for $SI = 0, \ldots, M_k - T_k$,

$$g_k(SI) = \min_{S:\,0 \leqslant S \leqslant \min(SI+T_k,\, E_k)} \{c_k(SI, S) + c'_k(SI, S) + h_k^+(S) + h_k^-(SI)\}. \quad (4)$$

In both functional equations above, $c_k(SI, S)$ is again an arbitrary function of SI and S for node k. $c'_k(SI, S)$ is the minimum cost of all child subgraphs that do not share any node with k's cluster, given an outgoing service time S and an incoming service time SI for k:

$$c'_k(SI, S) = \sum_{\substack{i<k \\ i \notin C_k \\ (i,k)\in\mathcal{A}}} \min_{x_i:\, x_i \leqslant SI} f_i(x_i) + \sum_{\substack{j<k \\ j \notin C_k \\ (k,j)\in\mathcal{A}}} \min_{y_j:\, y_j \geqslant S} g_j(y_j).$$

The functions $h_k^+(S)$ and $h_k^-(SI)$ are zero unless k is a cluster root for a nontrivial cluster. Furthermore, even for cluster roots, one of the functions h_k^+ or h_k^- must always be zero. For the cluster root, its functional equation must simultaneously minimize the cost-to-go of all nodes in the cluster plus their child subgraphs, necessitating the h_k^+ and h_k^- functions. While §4.2 demonstrates that a cluster root can be oriented to its cluster parent in one of four possible ways, there are only two variants of the cluster-combining function, $h_k^+(S)$ and $h_k^-(SI)$, which depend on the cluster root's relative depth in the cluster.

If k is a cluster root of a nontrivial cluster and k has the higher relative depth in the cluster, then for $S = 0, \ldots, M_k$,

$$h_k^+(S) = \min_{\substack{x_i,\, y_j: \\ y_j \geqslant S,\, (k,j)\in C_k \\ x_i \leqslant y_j,\, (i,j)\in C_k,\, i \neq k}} \left\{ \sum_{i \in C_k^+} f_i(x_i) + \sum_{j \in C_k^-} g_j(y_j) \right\}.$$

When the cluster root is a backward node, $h_k^+(S)$ describes the minimization of the optimal cost-to-go functions for the cluster nodes, over all feasible outgoing service times for the backward nodes and incoming service times for the forward nodes, given the outgoing service time S for the cluster root. These decision variables cannot be separated completely because the range of incoming service times feasible for the forward nodes is a function of the outgoing service times for the backward nodes.

If k is a cluster root of a nontrivial cluster and k has the lower relative depth in the cluster, then for $0 \leqslant SI \leqslant M_k - T_k$,

$$h_k^-(SI) = \min_{\substack{x_i,\, y_j: \\ x_i \leqslant SI,\, (i,k)\in C_k \\ x_i \leqslant y_j,\, (i,j)\in C_k,\, j \neq k}} \left\{ \sum_{i \in C_k^+} f_i(x_i) + \sum_{j \in C_k^-} g_j(y_j) \right\}.$$

When the cluster root is a forward node, $h_k^-(SI)$ describes the minimization of the optimal cost-to-go functions for the cluster nodes, over all feasible outgoing service times for the backward nodes and incoming service times for the forward nodes, when the incoming service time for the cluster root itself is SI.

The nodes are evaluated in the order $k = 1, \ldots, n$, guaranteeing that when node k needs to be evaluated, all cost-to-go functions for its children are available. The optimal cost for **P** can then be obtained by minimizing $g_n(SI)$ over $SI = 0, \ldots, M_n - T_n$. The proof of the correctness of **DP** appears in the appendix.

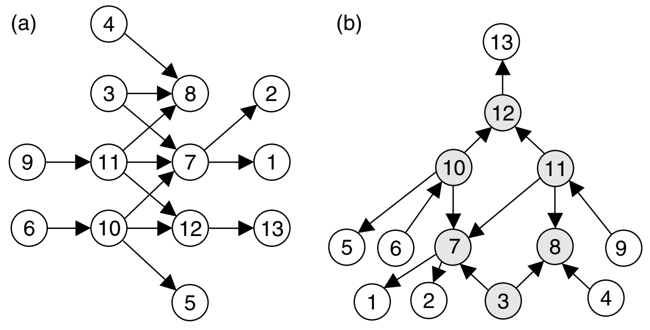
Finally, a note on the complexity of the evaluation. Obtaining $h_k^-(SI)$ when evaluating $f_k(S)$ requires evaluating $h_k^-(SI)$ inside the inner minimization in (3), whereas if $h_k^+(S)$ were required, it could be moved out of the inner braces on the right-hand side. The reverse is true for $g_k(SI)$, i.e., computing $h_k^-(SI)$ is easier than computing $h_k^+(S)$. Therefore, the first and the last networks in Figure 4 have worse complexity than the middle two.

5.1. Illustrative Example

An example helps illustrate the formulation **DP** as well as the nature of the h_k^+, h_k^- functions. Figure 5(a) shows a CoC network and Figure 5(b) shows the same network hung by the root node, which is node 13. Nodes 3, 7, 8, 10, 11, and 12 form the only nontrivial cluster in the network, and node 12 is their cluster root.

Evaluation of the functional equations proceeds as follows: g_1, g_2, f_3, f_4, g_5, f_6, g_7, g_8, f_9, f_{10}, and f_{11} are obtained easily as h_k^+, h_k^- are 0 for all of these nodes. To obtain $f_{12}(S)$, we need the function $h_{12}^-(SI)$, which minimizes the sum of f_3, g_7, g_8, f_{10}, and f_{11} over all feasible outgoing and incoming service times for nodes 3, 7, 8, 10, and 11. When the incoming service time for node 12 is SI, the feasible range of outgoing service times for nodes 10 and 11 are $S_{10} \leqslant SI$ and $S_{11} \leqslant SI$. For every value of S_{10} and S_{11} chosen, the feasible range of incoming

Figure 5. (a) Depiction of a CoC network. In (b), the network has been hung from the root node to illustrate the solution sequence as we step through the dynamic program (nodes in the cluster are shaded).



service times for nodes 7 and 8 changes; i.e., $SI_7 \geqslant S_{10}$, $SI_7 \geqslant S_{11}$, and $SI_8 \geqslant S_{11}$. These service-time constraints cascade through the cluster. For every choice of SI_7 and SI_8, the feasible range of outgoing service time for node 3 changes, i.e., $S_3 \leqslant SI_7$ and $S_3 \leqslant SI_8$. This problem does not arise in the case of noncluster nodes like nodes 1 and 2 because their feasible range of incoming service times are simply a function of the outgoing service time from node 7. Therefore, the minimization expression for a node like node 1 does not involve the nodes in the cluster, whereas obtaining $f_{12}(S)$ requires minimizing the sum of f_3, g_7, g_8, f_{10}, and f_{11} over all combinations of variables S_3, SI_7, SI_8, S_{10}, and S_{11}. The latter minimization problem is described by the function $h^-_{12}(SI)$ for every value of SI at node 12. g_{13} is then obtained without involving any h^+_k, h^-_k functions because node 13 is not a nontrivial cluster's root.

5.2. Complexity

It is now possible to get a sense of the complexity of **DP**. Assume that $c_k(SI, S)$ evaluations are $O(1)$ for each k in the network. For most realistic models of supply chains, this is a very reasonable assumption because the cost functions are usually obtained through a preprocessing algorithm, as in GW, for example. Also assume that the c'_k functions can be computed and stored so that obtaining the cost-to-go function of a trivial cluster's root k has complexity $O(M_k^2)$, similar to GW (c.f. §2.5).

Then, if we consider enumeration as the only mechanism for obtaining functions h^+_k and h^-_k, whenever k is the root of a nontrivial cluster the worst-case complexity for obtaining its functional equation over the needed range is $O(M_k^2 + \prod_{i\in C_k} M_i)$. When M_i for $i \in C_k$ are of comparable magnitude, because the sets C^-_k, C^+_k are never empty, the complexity is $O((\max_{i\in C_k} M_i)^{|C^-_k|+|C^+_k|+1})$, i.e., exponential in the number of nodes in the cluster. The worst-case complexity of the **DP** formulation is then exponential in the number of nodes in the largest-sized cluster in the network, which makes the solution of **DP** computationally burdensome for large clusters.

The rest of this paper is dedicated to reducing this complexity several-fold so that clusters of reasonable sizes can be practically solved. Also note that when there are no clusters, the complexity of **DP** reduces to the complexity of **SDP**, which is reasonable to expect.

5.3. The Optimal Policy

A final note concerns obtaining the optimal policy for **DP**. This is exactly similar to **SDP** except additional information must be maintained for roots of nontrivial clusters, i.e., the optimal policy for node k is still a function $\mu_k(S)$ if $f_k(S)$ needs evaluation and $\nu_k(SI)$ if $g_k(SI)$ needs evaluation. $\mu_k(S)$ maps S to the set: incoming service time SI^* for k, outgoing service times x^*_i, and incoming service times y^*_j for all nodes that result in the minimum value of the expression on the right-hand side for $f_k(S)$. For cluster roots, this set contains x^*_i and y^*_j, which achieve the minimum of $h^-_k(SI)$ or $h^+_k(S)$, whichever is relevant, for the given S. Intuitively, this corresponds to storing the optimal decision variables at the cluster root for all cluster nodes as well as all adjacent nodes (excluding the parent) not in the same cluster.

The same is true for $\nu_k(SI)$, which maps SI to the outgoing service time S^* for k, outgoing service times x^*_i, and incoming service times y^*_j for all nodes that result in the minimum value of the expression on the right-hand side for $g_k(SI)$. When k is a cluster root, this contains all optimal decision variables for the cluster nodes that are not the root. These functions allow us to backtrack and obtain the optimal service times for the entire network once the minimum of **P** has been obtained from $g_n(SI)$.

6. Reducing the Complexity of the Dynamic Program

The complexity of the **DP** formulation outlined in §5.2 is dominated by the subproblem of obtaining $h^-_k(SI)$ and $h^+_k(S)$, i.e., obtaining the minimum cost-to-go of cluster roots. Here, we present two improvements that significantly reduce the computational burden of obtaining these functions, allowing us to optimize practical-sized chains. In this paper, we only consider improvements possible in enumeration schemes. Faster exact algorithms or approximation schemes for obtaining functions h^+_k, h^-_k could be another promising research direction.

To place these improvements in context, we remark that many real-world supply chains do not have symmetric clusters, i.e., it is often the case that many common components go into a few subassemblies, or a few end-items proliferate through a large distribution network. It is when asymmetries of this kind exist in clusters that the most dramatic savings occur from our proposed improvements.

6.1. Iterating over a Subset of the Nodes

Consider the function $h_k^-(SI)$ and note that it can be written in two alternate forms:

$$h_k^-(SI) = \min_{y_j: y_j \leqslant M_j - T_j,\, j \in C_k^-} \Bigg\{ \sum_{i \in C_k^+} \min_{\substack{x_i: x_i \leqslant y_j,\, (i,j) \in C_k,\, j \neq k \\ x_i \leqslant SI,\, (i,k) \in C_k}} f_i(x_i) + \sum_{j \in C_k^-} g_j(y_j) \Bigg\}, \tag{5}$$

$$h_k^-(SI) = \min_{\substack{x_i: x_i \leqslant SI,\, (i,k) \in C_k \\ x_i \leqslant M_i,\, (i,k) \notin C_k,\, i \in C_k^+}} \Bigg\{ \sum_{i \in C_k^+} f_i(x_i) + \sum_{j \in C_k^-} \min_{y_j: y_j \geqslant x_i,\, (i,j) \in C_k} g_j(y_j) \Bigg\}. \tag{6}$$

The expressions look complicated, but the intuition is relatively straightforward, as Figure 6 demonstrates. Figure 6(a) corresponds to expression (5). We see that every feasible combination of the decision variables SI and y_j for the forward nodes $j \in C_k^-$ fixes the ranges over which the minimum of the cost-to-go function for all backward nodes $i \in C_k^+$ needs to be obtained, and these ranges are then independent of each other. The implications are immediate. One does not need to enumerate all combinations of x_i to obtain the minimum of the functions $f_i(x_i)$. It is sufficient to enumerate the combinations y_j for each j that is a forward node, and for each such combination use the minimum of the functions $f_i(x_i)$ over the implied range for each i that is a backward node.

Figure 6(b) corresponds to expression (6). Exactly analogous to Figure 6(a), here every feasible combination of decision variables for the backward nodes fixes the range over which the minimum of the cost-to-go functions for the forward nodes needs to be obtained, excluding the cluster root. These ranges make the minimizations of the functions $g_j(y_j)$ completely independent of each other, and the functions can be minimized separately for each $j \in C_k^-$.

Similar observations for the function $h_k^+(S)$ are demonstrated in Figure 7, allowing us to rewrite $h_k^+(S)$ as either

$$h_k^+(S) = \min_{x_i: x_i \leqslant M_i,\, i \in C_k^+} \Bigg\{ \sum_{i \in C_k^+} f_i(x_i) + \sum_{j \in C_k^-} \min_{\substack{y_j: y_j \geqslant x_i,\, (i,j) \in C_k,\, i \neq k \\ y_j \geqslant S,\, (k,j) \in C_k}} g_j(y_j) \Bigg\} \quad \text{or}$$

$$h_k^+(S) = \min_{\substack{y_j: y_j \geqslant S,\, (k,j) \in C_k \\ y_j \leqslant M_j - T_j,\, (k,j) \notin C_k,\, j \in C_k^-}} \Bigg\{ \sum_{i \in C_k^+} \min_{x_i: x_i \leqslant y_j,\, (i,j) \in C_k} f_i(x_i) + \sum_{j \in C_k^-} g_j(y_j) \Bigg\}.$$

Figure 6. Demonstrating iteration over the decision variables in only C_k^+ or C_k^- when solving for h_k^-.

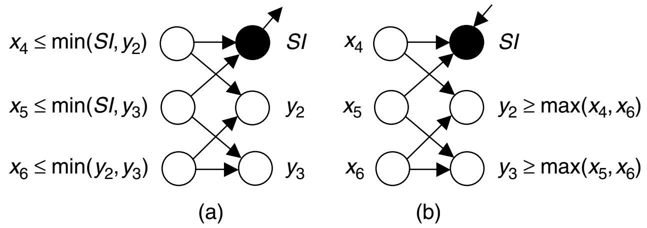
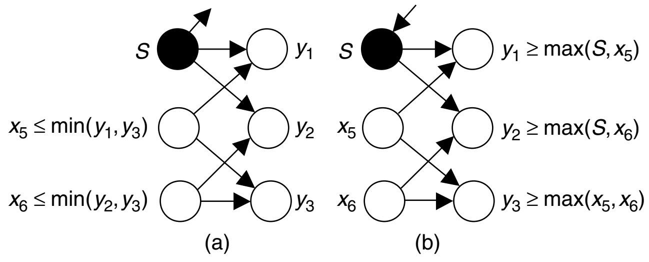


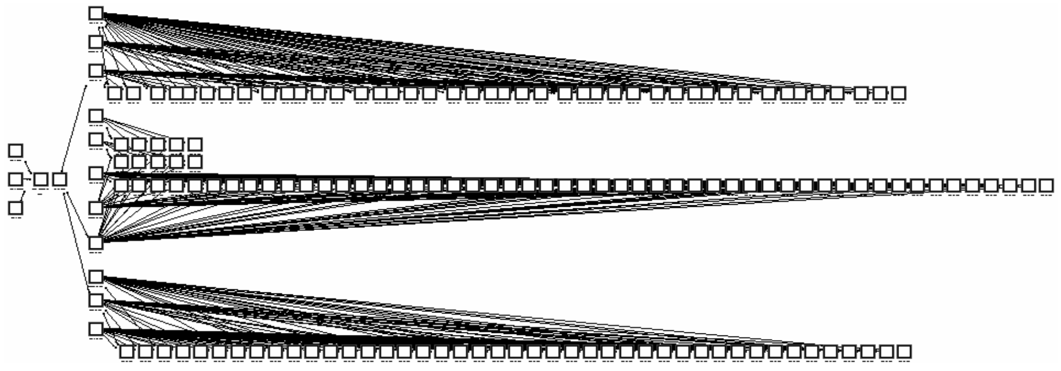
Figure 7. Demonstrating iteration over the decision variables in only C_k^+ or C_k^- when solving for h_k^+.



The manipulations above significantly reduce the complexity of obtaining $h_k^-(SI)$ or $h_k^+(S)$. Specifically, the complexity of solving the functional equation is reduced from $O((\max_{i \in C_k} M_i)^{|C_k^-|+|C_k^+|+1})$ (c.f. §5.2) to $O(\min[(\max_{i \in C_k^+} M_i)^{|C_k^+|+1}, (\max_{j \in C_k^-} M_j)^{|C_k^-|+1}])$. To see its impact on the range of chains we can optimize, consider Figure 8, which shows a real-world consumer-products supply chain optimized using the dynamic programming formulation presented in this paper. Keeping consistent with the paper's focus on modeling and algorithmic insights, we will only present a high-level summary of this supply chain. The network has 163 nodes and 301 arcs. There are three clusters, corresponding to the product offerings in each of three separate geographic regions of the world. The top and bottom clusters are identical; they each have 46 nodes and 86 arcs; there are 43 forward nodes and 3 backward nodes. The middle cluster has 66 nodes and 122 arcs; there are 61 forward nodes and 5 backward nodes. The longest processing-time path at the backward nodes is 57 days and at the forward nodes is 67 days. With our improved algorithm, the time for obtaining the h_k functions is bounded by $O(57^4)$ and $O(57^6)$, respectively, compared with the unimproved complexity, $O(67^{47})$ and $O(67^{67})$. For this problem, the dynamic program solves in less than 30 seconds.

In practice, we have numerous examples where the entire supply chain consists of hundreds of nodes spanning 10 or more echelons. Within those supply chain networks, it is not uncommon to see 50 common components go into three subassemblies and four or fewer end items get bundled (with unique items) into tens of specialized SKUs specific to individual retailers. The improvement outlined in this section allows these kinds of models to be solved easily in practice.

Figure 8. Asymmetric clusters frequently encountered in practice.



6.2. Recursively Obtaining the h_k^+ and h_k^- Functions

Another improvement that allows us to drop an exponent from $O(\min[(\max_{i\in C_k^+} M_i)^{|C_k^+|+1}, (\max_{j\in C_k^-} M_j)^{|C_k^-|+1}])$ is motivated by the nature of the linear minimizations in expression (3) for the function $c'_k(SI, S)$ inside the braces. Evaluating $c'_k(SI, S)$ requires minimizing f_i over $[0, SI]$ and minimizing g_j over $[S, M_j]$. Obtaining a single value $f_k(S)$ is then $O(M_k^2)$, and computing $f_k(S)$ for all S is $O(M_k^3)$ for all nodes except the roots of nontrivial clusters.

We can, however, get by with complexity $O(M_k^2)$. More precisely, defining the minimum of the functions as $F_k(S) = \min_{x:0\leqslant x\leqslant S} f_k(x)$, we can obtain the function $F_k(S)$ in $O(M_k)$ time by using the recursion $F_k(S+1) = \min(F_k(S), f_k(S+1))$ in the order $S = 0, \ldots, M_k - 1$, with the initial condition $F_k(0) = f_k(0)$. A similar function can be defined as $G_k(SI) = \min_{y:SI\leqslant y\leqslant M_k-T_k} g_k(y)$ and obtained in linear time using $G_k(SI) = \min(G_k(SI+1), g_k(SI))$ in the order $SI = M_k - T_k - 1, \ldots, 0$, with the initial condition $G_k(M_k - T_k) = g_k(M_k - T_k)$. Then, $c'_k(SI, S) = \sum_{i<k,\, i\notin C_k,\, (i,k)\in\mathcal{A}} F_i(S) + \sum_{j<k,\, j\notin C_k,\, (k,j)\in\mathcal{A}} G_j(SI)$ is a simple summation without any minimizations. These manipulations capitalize on the fact that successive increasing values of S and decreasing values of SI are relaxations on the minimizations required for $c'_k(SI, S)$.

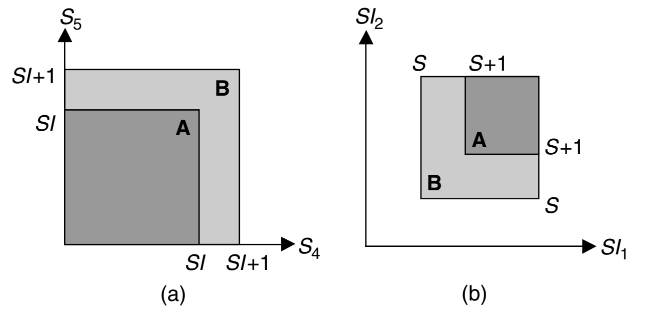
A similar intuition holds in higher dimensions when attempting to obtain the cost-to-go function for the cluster root. Consider, for instance, $h_k^-(SI)$. Here, increasing values of SI are successive relaxations on the service times the cluster nodes connected to the cluster root can quote. Therefore, once one has obtained $h_k^-(SI)$, it is possible to obtain $h_k^-(SI+1)$ by evaluating only an incremental number of points, rather than minimizing again over the entire set of points for $SI+1$. This, in fact, is the case as illustrated in Figure 9(a), based on the example in Figure 6, which shows how obtaining $h_k^-(SI)$ requires minimization over a set **A**, and $h_k^-(SI+1)$ requires minimization over $\mathbf{A}\cup\mathbf{B}$. One needs, therefore, only to minimize the objective function for $h_k^-(SI+1)$ over the set **B** and compare to $h_k^-(SI)$ to obtain $h_k^-(SI+1)$. Note that this requires obtaining $h_k^-(SI)$ in increasing order of SI.

Similarly, for $h_k^+(S)$, Figure 9(b) shows, for the example of Figure 7, how decreasing values of S result in a containment relation, making it possible to obtain $h_k^+(S)$ incrementally once $h_k^+(S+1)$ has been obtained.

While it is formally possible to define functions $h_k^-(SI+1) = \min(h_k^-(SI), \cdot)$ where $\cdot$ would be the minimization problem over an incremental set of points, the constraints for $\cdot$ would be disjunctive because the incremental sets are not convex. Such a formulation for $\cdot$ also makes the complexity improvement hard to see.

An alternative algorithmic definition is more revealing for the functions $h_k^-(SI+1)$ in terms of the complexity of evaluation. It relies on the fact that in Figure 9, the boundary of the sets is the function $\max(S_4, S_5)$. This observation is in fact general. For cluster root k, initialize the functions $h_k^-(SI) = \infty$ for $SI = 0, \ldots, M_k - T_k$. Then, if the set Q_k describes all possible combinations of decision variables for the backward cluster nodes, i.e., $Q_k = \{x_i\colon i \in C_k^+,$

Figure 9. This figure shows successive relaxations over which the minimum cost-to-go from the cluster nodes is required for increasing/decreasing values of the cluster root's SI/S.



Notes. Figure (a) demonstrates relaxations when evaluating $h_k^-(SI)$ and (b) demonstrates relaxations when evaluating $h_k^+(S)$. In both, the sets **B** (lightly shaded) are the incremental points that need to be evaluated for $h_k^-(SI+1)/h_k^+(S)$ once $h_k^-(SI)/h_k^+(S+1)$ have been obtained.

$x_i \leqslant M_i\}$; for every combination of $x_i \in Q_k$, set

$$h_k^-\Big(\max_{i:(i,k)\in C_k} x_i\Big) = \min\Bigg(h_k^-\Big(\max_{i:(i,k)\in C_k} x_i\Big), \sum_{i\in C_k^+} f_i(x_i) + \sum_{j\in C_k^-} \min_{y_j:\, y_j\geqslant x_i,\,(i,j)\in C_k} g_j(y_j)\Bigg).$$

Set $h_k^-(SI+1) = \min(h_k^-(SI), h_k^-(SI+1))$ for $SI = 0, \ldots, M_k - T_k - 1$.

This definition makes obvious that only one complete enumeration of the decision variables for the backward nodes in a cluster is required to obtain the function $h_k^-(SI)$, and storage is trivial because only a single-state variable is involved.

A similar observation in Figure 9(b) is that the boundary of the sets is $\min(SI_1, SI_2)$. Again, paralleling the above argument for defining it algorithmically, the functions $h_k^+(S)$ can be obtained as follows, given the initial conditions $h_k^+(S) = \infty$ for $S = 0, \ldots, M_k$. Here, the set $Q_k = \{y_j\colon j \in C_k^-, y_j \leqslant M_j - T_j\}$, and we need only one enumeration of decision variables for the forward cluster nodes to obtain and store $h_k^+(S)$.

For every combination of $y_j \in Q_k$, set

$$h_k^+\Big(\min_{j:(k,j)\in C_k} y_j\Big) = \min\Bigg(h_k^+\Big(\min_{j:(k,j)\in C_k} y_j\Big), \sum_{i\in C_k^+} \min_{x_i:\, x_i\leqslant y_j,\,(i,j)\in C_k} f_i(x_i) + \sum_{j\in C_k^-} g_j(y_j)\Bigg).$$

Set $h_k^+(S) = \min(h_k^+(S+1), h_k^+(S))$ for $S = M_k - 1, \ldots, 0$.

Finally, it is not too difficult to see that both algorithms for obtaining $h_k^-(SI)$ and $h_k^+(S)$ work if we were to only iterate over a subset of the nodes as mentioned in §6.1, regardless of the side of the cluster we iterate over. As an example, if we were to use expression (5) for enumeration, each combination of decision variables SI_k, y_j, $j \in C_k^-$, would modify $h_k^-(SI_k)$ in the algorithm for $h_k^-(SI)$ above instead of $h_k^-(\max_{i:(i,k)\in C_k} x_i)$.

6.3. Improved Complexity

With both improvements above, the worst-case complexity of obtaining the functional equation for a nontrivial cluster root k over its entire range becomes $O(\min[(\max_{i\in C_k^+} M_i)^{|C_k^+|}, (\max_{j\in C_k^-} M_j)^{|C_k^-|}])$.

The total complexity of **DP** from §5 can then be imagined as being dominated by the largest, in terms of its smaller side, nontrivial cluster. These improvements allow us to handle supply chains with large clusters as described in Billington et al. (2004).

7. Summary

Realizing the practical utility of the modeling framework in Graves and Willems (2000) and a real need for extending the kinds of supply chains that can be optimized, we have contributed the following in this paper.

- We have formalized a network structure (which we call CoC networks) that captures a large class of real-world supply chains that involve common components and cannot be modeled as spanning trees.
- We have presented algorithms to identify these networks and determine a substructure that can be exploited in a dynamic programming algorithm for optimization.
- We have presented a dynamic programming formulation for obtaining the optimal safety stock levels for these networks.
- We have presented two improvements for solving the most computationally expensive subproblem in the dynamic program. This makes optimizing real-world supply chains practical, as has been demonstrated in Billington et al. (2004).

In terms of future research, we see two primary directions. First, there is still a need to determine optimal inventory levels in general acyclic networks. While CoC networks significantly expand the class of supply chains that can be optimized, it does not address all network structures encountered in reality. Extending this approach or developing a new approach is a significant research opportunity. Second, the fact that we can solve arbitrary cost functions of the stage decision variables allows the work to generalize beyond just solving inventory placement problems. It would be good to test where else this kind of modeling framework can be fruitfully applied.

Appendix

The primary proof in this section is that the **DP** formulation of §5 is correct. Assume throughout this section that when we say cluster root, we include roots of trivial clusters.

Lemma A.1. *Nodes in any CoC network can be ordered using an index k such that* (i) *at most one node j adjacent to, but outside the cluster can have an index greater than all the nodes in the cluster*; *and* (ii) *at most one node k in every cluster has an adjacent node not in the same cluster with an index $p_k > k$.*

Proof. Use the labeling algorithm of §4.2 and suppose that (i) is not true. Assume that k has the highest index in its cluster. Then, there exist at least two distinct nodes i and j adjacent to the cluster that are not in the same cluster, such that $i > k$ and $j > k$. Move to nodes i and j. Follow the path that moves to the parent at each node. Because parent indices are strictly greater than the child's index, we cannot cycle back; therefore, at some point we must come to a common parent l that labeled the nodes along these distinct paths, or we terminate at two distinct nodes, both of which do not have parents.

Terminating at two distinct nodes without parents contradicts the labeling algorithm because only one node in the network can exist without a parent. Finding a common node l contradicts the cluster definition in one of two ways. First, if nodes i, j, and l and the nodes along the paths to node l all have the same relative depths as the cluster nodes, then the cluster is not a maximal bipartite subgraph because all nodes in the path can be added to it. Second, if one or more nodes from i, j, and l and the nodes along the paths to node l have a different relative depth than the cluster's relative depths, then the network violates the definition of a CoC network because we have the network not being a spanning tree of nodes and clusters.

The above argument holds whether k was the highest index in a cluster or the index of a trivial cluster. This shows (i).

For trivial clusters, (ii) is trivially true. For nontrivial clusters, suppose that (ii) is not true. Beginning from two separate nodes in the cluster, use exactly the above argument to find a node not in the cluster that violates either the labeling algorithm, the maximality of a cluster, or the spanning-tree condition for a CoC network. □

Lemma A.2. *Given the labeling algorithm of* §4.2, *for any CoC network a set of nodes N_k can be defined for each k as below such that all sets N_i, N_j appearing in the definition are disjoint from each other*:

$$N_k = \{k\} \bigcup_{\substack{i<k\\ i\notin C_k\\ (i,k)\in\mathcal{A}}} N_i \bigcup_{\substack{j<k\\ j\notin C_k\\ (k,j)\in\mathcal{A}}} N_j \begin{cases} \bigcup_{i\in C_k^+} N_i \bigcup_{j\in C_k^-} N_j \\ \qquad \textit{if } k \textit{ is a cluster root}, \\ \cup\varnothing \quad \textit{otherwise}. \end{cases}$$

Proof. We can see this inductively. Start with nodes $N_k = \{k\}$, and the above is trivially true. Use Lemma A.1 to see that each such N_k can only be included in at most one N_{p_k} for both cluster roots and noncluster nodes. Here, we let p_k denote the index of the cluster root whenever k is not the root of its cluster.

Now suppose that the argument is true for a given N_k. Then, using Lemma A.1, again note that N_k can only be included in at most one N_{p_k}. □

Theorem A.3. **DP** *correctly solves* **P** *without the constraints $SI_i = 0\ \forall\, i \in V_s$ if the underlying network for* **P** *is a CoC network.*

Proof. Given N_k, associate the subgraph $G_k = (N_k, \mathcal{A}_k)$ with each k, where $\mathcal{A}_k = \{(i,j)\in\mathcal{A}\colon i\in N_k,\ j\in N_k\}$. Intuitively, if k is not a cluster root, G_k is the connected subgraph that contains k after the arc between k and p_k and any arcs in the same cluster as k are removed from G. If k is the cluster root, it is the connected subgraph that contains k after removing only the arc between k and p_k.

Define a family of optimization problems $\mathbf{P}_k$, one for each node k, as follows:

$$\begin{aligned} \mathbf{P}_k\colon\ \min\ & c_k(S_k, SI_k) + \sum_{i<k:\, i\in N_k} c_i(SI_i, S_i) \\ \text{s.t.}\ & S_i - SI_i \leqslant T_i \quad \forall\, i\in N_k, \\ & S_j - SI_i \leqslant 0 \quad \forall\,(j,i)\in\mathcal{A}_k, \\ & S_i \leqslant \min(M_i, E_i) \quad \forall\, i\in N_k, \\ & SI_i \leqslant M_i - T_i \quad \forall\, i\in N_k, \\ & S_i, SI_i \geqslant 0 \text{ and integral} \quad \forall\, i\in N_k. \end{aligned}$$

Let $\mathscr{C}_k$ denote the cost function and Ω_k denote the set of constraints of $\mathbf{P}_k$. Note that we have explicitly added the constraints for the boundedness of the feasible region, as discussed in §2.4. Also note that $\mathbf{P}_n = \mathbf{P}$ if n is the network root.

The only constraints $\mathbf{P}_k$ has are those related to node k, the additional arcs needed to produce G_k from its children subgraphs, and the constraints of its constituent N_i, N_js. This follows from the correspondence of the constraints of $\mathbf{P}$ to the network structure because only the following three constraints are associated with each node k: $S_k - SI_k \leqslant T_k$, $0 \leqslant S_k \leqslant \min(M_k, E_k)$, and $0 \leqslant SI_k \leqslant M_k - T_k$. Without loss of generality, we assume that $E_i = \infty$ when $i \notin V_d$. The following constraint is associated with each arc $(j,i)\in\mathcal{A}$: $S_j - SI_i \leqslant 0$. There are no other constraints in $\mathbf{P}$ apart from integrality.

If k is a cluster root and $(k, p_k)\in\mathcal{A}$, or else if k is not a cluster root but a backward node in a nontrivial cluster, let $f_k^*(S)$ be the true optimal value of $\mathbf{P}_k$ if the service time for node k is fixed to S (i.e., the constraint $S_k = S$ is added to Ω_k). Similarly, if k is a cluster root with $(p_k, k)\in\mathcal{A}$, or the network root, or if k is not a cluster root but a forward node in a nontrivial cluster, let $g_k^*(SI)$ be the true optimal value of $\mathbf{P}_k$ if the incoming service time for k is fixed to SI (the constraint $SI_k = SI$ is added to Ω_k). Note that the optimal value to $\mathbf{P}$ can then be obtained as the $\min_{0\leqslant SI\leqslant M_n - T_n} g_n^*(SI)$ if n is the root node for the entire tree.

We will show by induction that the functions $f_k(S)$ and $g_k(SI)$ generated by the dynamic program **DP** (expressions (3) and (4) in §5) equal the optimal functions $f_k^*(S)$ and $g_k^*(SI)$ for every k. The base case is to verify that for all nodes k such that $N_k = \{k\}$, the functions $g_k(SI) = g_k^*(SI)$ and $f_k(S) = f_k^*(S)$, i.e., the functions produced by **DP**, are optimal. This is straightforward to see.

Now consider any k and suppose that for all $i < k$ such that $i \in N_k$, $g_i(SI) = g_i^*(SI)$ or $f_i(S) = f_i^*(S)$, whichever is applicable for i. Then, if either (i) k is a cluster root with $(p_k, k)\in\mathcal{A}$ or the network root, or (ii) k is not a cluster root but a forward node in a nontrivial cluster, we can write for every SI satisfying $0 \leqslant SI \leqslant M_k - T_k$:

$$\begin{aligned} g_k^*(SI) &= \min_{\substack{\Omega_k\\ SI_k = SI}} \left\{ c_k(SI_k, S_k) + \sum_{\substack{i<k\\ i\in N_k}} c_i(SI_i, S_i) \right\} \\ &= \min_{\substack{\Omega_k\\ SI_k = SI}} \left\{ c_k(SI_k, S_k) + \sum_{\substack{i<k\\ i\notin C_k\\ (i,k)\in\mathcal{A}_k}} \mathscr{C}_i + \sum_{\substack{j<k\\ j\notin C_k\\ (k,j)\in\mathcal{A}_k}} \mathscr{C}_j \right. \\ &\qquad \left. + \sum_{i\in C_k^+} \mathscr{C}_i + \sum_{j\in C_k^-} \mathscr{C}_j \right\}, \end{aligned}$$

$$
\begin{aligned}
&= \min_{\substack{SI-S_k+T_k\geqslant 0\\ 0\leqslant S_k\leqslant E_k}} \Bigg\{ c_k(SI,S_k) + \sum_{\substack{i<k\\ i\notin C_k\\ (i,k)\in\mathcal{A}_k}} \min_{\substack{\Omega_i\\ S_i\leqslant SI}} \mathscr{C}_i \\
&\qquad + \sum_{\substack{j<k\\ j\notin C_k\\ (k,j)\in\mathcal{A}_k}} \min_{\substack{\Omega_j\\ SI_j\geqslant S_k}} \mathscr{C}_j \\
&\qquad + \min_{\substack{\Omega_i,\, i\in C_k^+\\ \Omega_j,\, j\in C_k^-\\ S_i\leqslant SI_j,\, i\in C_k^+,\, j\in C_k^-\\ S_i\leqslant SI,\, (i,k)\in C_k\\ SI_j\geqslant S_k,\, (k,j)\in C_k}} \Bigg(\sum_{i\in C_k^+} \mathscr{C}_i + \sum_{j\in C_k^-} \mathscr{C}_j \Bigg) \Bigg\} \\
&= \min_{\substack{SI-S_k+T_k\geqslant 0\\ 0\leqslant S_k\leqslant E_k}} \Bigg\{ c_k(SI,S_k) + \sum_{\substack{i<k\\ i\notin C_k\\ (i,k)\in\mathcal{A}_k}} \min_{x_i\leqslant SI} f_i^*(x_i) \\
&\qquad + \sum_{\substack{j<k\\ j\notin C_k\\ (k,j)\in\mathcal{A}_k}} \min_{y_j\geqslant S_k} g_j^*(y_j) \\
&\qquad + \min_{\substack{x_i\leqslant y_j,\, i\in C_k^+,\, j\in C_k^-\\ x_i\leqslant SI,\, (i,k)\in C_k\\ y_j\geqslant S_k,\, (k,j)\in C_k}} \Bigg(\sum_{i\in C_k^+} f_i^*(x_i) + \sum_{j\in C_k^-} g_j^*(y_j) \Bigg) \Bigg\} \\
&= \min_{\substack{SI-S_k+T_k\geqslant 0\\ 0\leqslant S_k\leqslant E_k}} \Bigg\{ c_k(SI,S_k) + \sum_{\substack{i<k\\ i\notin C_k\\ (i,k)\in\mathcal{A}}} \min_{x_i\leqslant SI} f_i(x_i) \\
&\qquad + \sum_{\substack{j<k\\ j\notin C_k\\ (k,j)\in\mathcal{A}}} \min_{y_j\geqslant S_k} g_j(y_j) \\
&\qquad + \underbrace{\min_{\substack{x_i\leqslant y_j,\, i\in C_k^+,\, j\in C_k^-\\ x_i\leqslant SI,\, (i,k)\in C_k\\ y_j\geqslant S_k,\, (k,j)\in C_k}} \Bigg(\sum_{i\in C_k^+} f_i(x_i) + \sum_{j\in C_k^-} g_j(y_j) \Bigg)} \Bigg\}.
\end{aligned}
$$

The first step above is simply the definition of $g_k^*(SI)$. In the second step, we use Lemma A.2, i.e., that the children subgraphs of k are disjoint, to separate the cost terms. The third step uses the separability of the constraints among the children subgraphs to decompose the constraint set Ω_k and write it under the relevant cost terms. The fourth step uses definitions of f_i^* and g_j^* and rewrites S_i and SI_j as x_i and y_j to differentiate the variables over which the optimal cost-to-go functions need to be minimized. The fifth step uses the inductive hypothesis, as well as rewriting the condition involving $\mathcal{A}_k$ in terms of $\mathcal{A}$, the network arc set.

Now note that the functions $h_k^+(S)$ and $h_k^-(SI)$ in expressions (3) and (4) in §5 are simply a rewriting of the expression in underbraces in the last step above. We use them to explicitly denote which variable—S_k or SI—the constraints of the minimization in underbraces depends on, and they helped us with the intuition needed for the improvements of §§6.1 and 6.2.

Finally, if (i) k is a cluster root with $(k, p_k) \in \mathcal{A}$, or (ii) if k is not a cluster root but a backward node of a nontrivial cluster, we can literally parallel the above steps to show that $f_k(S) = f_k^*(S)$. □

Acknowledgments

The authors thank Dr. John Bossert (Optiant, Inc.) and Professor Brian Tomlin (University of North Carolina, Chapel Hill) for their helpful and detailed comments on an earlier draft of this paper.

References

Axsäter, S. 2003. Supply chain operations: Serial and distribution inventory systems. A. G. de Kok, S. C. Graves, eds. *Handbooks in Operations Research and Management Science*, Vol. 11, Chapter 10. *Supply Chain Management: Design, Coordination and Operation*. North-Holland, Amsterdam, The Netherlands, 525–559.

Billington, C., G. Callioni, B. Crane, J. D. Ruark, J. Unruh Rapp, T. White, S. P. Willems. 2004. Accelerating the profitability of Hewlett-Packard's supply chains. *Interfaces* **34**(1) 59–77.

Clark, A., S. Scarf. 1960. Optimal policies for a multi-echelon inventory problem. *Management Sci.* **6** 475–490.

de Kok, A. G., J. Fransoo. 2003. Planning supply chain operations: Definition and comparison of planning concepts. A. G. de Kok, S. C. Graves, eds. *Handbooks in Operations Research and Management Science*, Vol. 11, Chapter 12. *Supply Chain Management: Design, Coordination and Operation*. North-Holland, Amsterdam, The Netherlands, 597–675.

Ettl, M., G. E. Feigin, G. Y. Lin, D. D. Yao. 2000. A supply network model with base-stock control and service requirements. *Oper. Res.* **49**(2) 216–232.

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

Graves, S. C., S. P. Willems. 1996. Strategic safety stock placement in supply chains. *Proc. 1996 MSOM Conf.*, Hanover, NH.

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

Graves, S. C., S. P. Willems. 2003a. Erratum: Optimizing strategic safety stock placement in supply chains. *Manufacturing Service Oper. Management* **5**(2) 176–177.

Graves, S. C., S. P. Willems. 2003b. Supply chain design: Safety stock placement and supply chain configuration. A. G. de Kok, S. C. Graves, eds. *Handbooks in Operations Research and Management Science*, Vol. 11, Chapter 3. *Supply Chain Management: Design, Coordination and Operation*. North-Holland, Amsterdam, The Netherlands, 95–132.

Inderfurth, K. 1991. Safety stock optimization in multi-stage inventory systems. *Internat. J. Production Econom.* **24** 103–113.

Inderfurth, K., S. Minner. 1998. Safety stocks in multi-stage inventory systems under different service levels. *Eur. J. Oper. Res.* **106** 57–73.

Kimball, G. E. 1988. General principles of inventory control. *J. Manufacturing Oper. Management* **1** 119–130.

Lee, H. L., C. Billington. 1992. Managing supply chain inventory: Pitfalls and opportunities. *Sloan Management Rev.* **33**(3) 65–73.

Lee, H. L., C. Billington. 1993. Material management in decentralized supply chains. *Oper. Res.* **41** 835–847.

Lee, H. L., V. Padmanabhan, S. Whang. 1997. Information distortion in a supply chain: The bullwhip effect. *Management Sci.* **43** 546–558.

Lin, G., M. Ettl, S. Buckley, S. Bagchi, D. Yao, B. Naccarato, R. Allan, K. Kim, L. Koenig. 2000. Extended enterprise supply chain management at IBM personal systems group and other divisions. *Interfaces* **30** 7–25.

Minner, S. 2000. *Strategic Safety Stocks in Supply Chains. Lecture Notes in Economics and Mathematical Systems*, Vol. 490. Springer-Verlag, Berlin, Germany.

Simpson, K. F. 1958. In-process inventories. *Oper. Res.* **6** 863–873.

Song, J.-S., P. Zipkin. 2003. Supply chain operations: Assemble-to-order systems. A. G. de Kok, S. C. Graves, eds. *Handbooks in Operations Research and Management Science*, Vol. 11, Chapter 11. *Supply Chain Management: Design, Coordination and Operation*. North-Holland, Amsterdam, The Netherlands, 561–596.