OPERATIONS RESEARCH
Vol. 59, No. 3, May–June 2011, pp. 781–787
ISSN 0030-364X | EISSN 1526-5463 | 11 | 5903 | 0781

DOI 10.1287/opre.1100.0913

TECHNICAL NOTE

Optimizing Strategic Safety Stock Placement in General Acyclic Networks

Salal Humair
School of Science and Engineering, Lahore University of Management Sciences, Lahore, Pakistan;
Department of Global Health and Population, Harvard School of Public Health, Boston, Massachusetts 02215,
shumair@hsph.harvard.edu

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

We present two significant enhancements to the guaranteed-service (GS) model for multiechelon safety stock placement. First, we let each stage's expected inventory cost be a generalized nonconcave non-closed-form function of its incoming and outgoing service time. This allows the GS model to incorporate important phenomena such as variable stage times and nonnested review periods, which previous GS literature has not allowed. Second, we optimize the generalized cost GS model for directed acyclic networks, rather than assembly/distribution networks or trees. For the resulting NP-hard optimization problem, we present a provably optimal algorithm that runs within minutes for 29 chains from a data set of 38 real-world supply chains ranging from 8 to 2,025 stages. We also present two significantly faster yet near-optimal heuristics. One heuristic is motivated by the structure of the formulation's dual space, whereas the other heuristic simply terminates the optimization algorithm after a fixed number of iterations. As a performance benchmark, on the 38 chains, the first heuristic has an average optimality gap of approximately 1.1% and average run time of 88 seconds, whereas the second heuristic has an average optimality gap of 2.8% and an average run time of 5.9 seconds.

Subject classifications: multiechelon inventory system; safety stock optimization; dynamic programming application; general acyclic networks.
Area of review: Manufacturing, Service, and Supply Chain Operations.
History: Received November 2007; revisions received August 2008, January 2009, March 2009, April 2009; accepted May 2009.

1. Introduction

In this paper, we expand the applicability of the guaranteed-service (GS) model for safety stock optimization by optimizing directed general acyclic networks for stage costs that are generalized functions of a stage's service times. What allows us to handle such general costs while moving beyond simpler network topologies is an algorithm that optimizes spanning tree relaxations of the network, and then intelligently exploits the spanning tree algorithm's properties to move from tree-optimal solutions to the network-optimal solution.

Minner (2000), Karimi et al. (2004), Lesnaia (2004), and Magnanti et al. (2006) all consider the GS model in general acyclic structures that employ concave stage costs. 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, he investigates several heuristic approaches, finding that tabu search performs best. Karimi et al. (2004) develop two heuristic algorithms for dense acyclic networks ranging from 10 to 8,000 stages. The heuristics employ a continuous and two-piece linear estimator, respectively, for the concave cost functions, successively iterating to refine the quality of the approximation. Lesnaia (2004) shows that the general network problem with concave costs is NP-hard. She uses the extreme point properties of the optimal solution within a branch-and-bound scheme to find the optimal solution, and considers the application of their algorithm for two-layer networks. Magnanti et al. (2006) develop a piecewise-linear approximation to concave stage cost functions that effectively solves sparse acyclic networks ranging from 10 to 100 stages. Their solution method relies on adding redundant inequalities to the LP relaxation and refining the piecewise-linear approximation in an iterative fashion. Humair and Willems (2006) consider nonconcave stage costs in a network topology they define as *networks containing clusters of commonality*; informally described as networks of bipartite subgraphs that form spanning trees when each bipartite subgraph is collapsed to a single node.

We present the mathematical program underlying the GS model and examples of generalized cost functions in §2, our algorithm and its performance in §3, and some faster heuristics in §4. The online supplement (OLS) is available as an electronic companion to this article at

http://or.journal.informs.org/. It contains a literature review, overview of the GS model, additional run-time results, discussion of the mathematical program's dual space, and the optimality proof for the modified spanning-tree algorithm.

2. The GS Model for General Acyclic Networks and Generalized Cost Functions

The GS model consists of an acyclic connected graph $G = (V, \mathcal{A})$, $|V| = n$, where V is the set of all stages and $\mathcal{A}$ the set of all arcs. S_i, SI_i, and T_i denote stage i's stage time, outgoing service time, and incoming service time. $V_e \subset V$ and $V_s \subset V$ denote sets of demand and supply stages, respectively. E_i is the maximum S_i quotable at a demand stage. With time discrete, per-unit inventory cost C_i, maximum demand over any interval of time x a concave function $D_i(x)$, and the average demand μ_i for stage i, the standard optimization problem **P** considered in the GS literature is

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

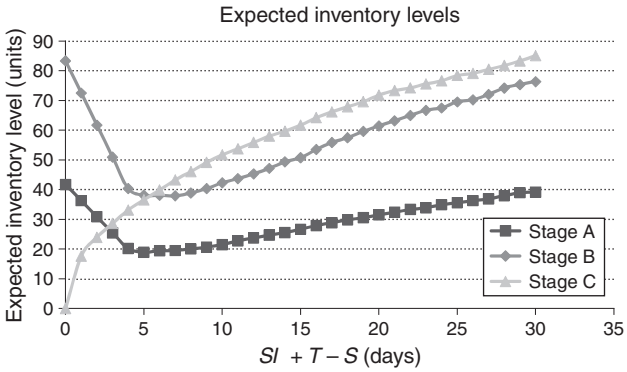
We allow stage costs of any generalized form $c_i(SI_i, S_i)$ in **P** instead of only the concave form $C_i\{D_i(SI_i + T_i - S_i) - \mu_i(SI_i + T_i - S_i)\}$. There are no structural limitations on $c_i(SI_i, S_i)$ as long as a stage's cost depends on only its own service times, i.e., concavity, monotonicity, or closed-formedness is not required. With generalized stage costs, the GS model can capture common and important real-world phenomena such as variable stage times, nonnested review periods and nonnested batch ordering. Variable stage times are the most common, as the 38 industrial supply chains in Willems (2008) show, where 26 chains from 22 different companies have variable stage times. Expected inventory levels under these phenomena produce nonconcave costs as shown in Figures 1 and 2, and in the OLS, where it is also explained why these costs cannot be handled by existing GS algorithms and common nonconvex optimization approaches.

3. The General Networks Algorithm

3.1. A Modified Spanning-Tree Algorithm

Our general networks algorithm uses a modified form of the dynamic program in Graves and Willems (2000, 2003) (GW). Suppose **P** includes two extra constraints of the form $L_i \leqslant SI_i \leqslant U_i$ and $l_i \leqslant S_i \leqslant u_i$ for each stage i, where L_i, U_i, l_i, and u_i are constants. Call the modified problem **P**′.

Figure 1. Simulated cost functions under variable stage times.

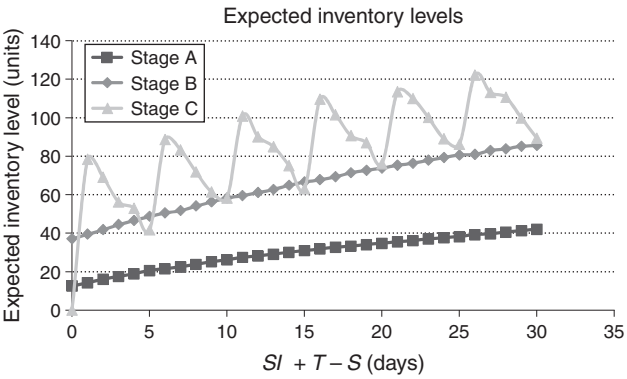


Assume that the constraints involving E_i are collapsed into the more general upper bound u_i. If G is a spanning tree, the stages can be labeled with an index $k = 0, \ldots, n$ such that every stage except one (called *root*) has at most one adjacent stage with an index higher than its own (called *parent*). Let p_k be the index of stage k's parent. With a slight abuse of notation, let $L_k(S) = \max(S - T_k, L_k)$ and $\bar{U}_k = \min(M_k - T_k, U_k)$. Similarly, let $u_k(SI) = \min(SI + T_k, u_k)$. If $(k, p_k) \in \mathcal{A}$, let $f_k(S)$ be the optimal cost of the subtree connected to k after removing p_k if k's outgoing service time is S. If $(p_k, k) \in \mathcal{A}$, let $g_k(SI)$ be the optimal cost of the subtree connected to k after removing p_k if the incoming service time to k is SI. Then the following dynamic program **SDP**′ solves **P**′ correctly if G is a spanning tree, as the theorem below states (proof in the OLS).

If $(k, p_k) \in \mathcal{A}$ and $k \leqslant n - 1$, then $f_k(S) = \infty$ if $S < l_k$, $S > u_k$, or if $L_k(S) > \bar{U}_k$. Otherwise, for $l_k \leqslant S \leqslant u_k$,

$$f_k(S) = \min_{\substack{SI: \\ L_k(S) \leqslant SI \leqslant \bar{U}_k}} \left\{ c_k(SI, S) + \sum_{\substack{i<k \\ (i,k)\in\mathcal{A}}} \min_{x: x \leqslant SI} f_i(x) + \sum_{\substack{j<k \\ (k,j)\in\mathcal{A}}} \min_{y: y \geqslant S} g_j(y) \right\}.$$

Figure 2. Simulated cost functions under nonnested review periods.



If $(p_k, k) \in \mathcal{A}$ or if $k = n$, $g_k(SI) = \infty$ if $SI < L_k$, $SI > U_k$, or if $l_k > u_k(SI)$. Otherwise, for $L_k \leqslant SI \leqslant U_k$,

$$g_k(SI) = \min_{\substack{S: \\ l_k \leqslant S \leqslant u_k(SI)}} \left\{ c_k(SI, S) + \sum_{\substack{i<k \\ (i,k)\in\mathcal{A}}} \min_{x: x \leqslant SI} f_i(x) + \sum_{\substack{j<k \\ (k,j)\in\mathcal{A}}} \min_{y: y \geqslant S} g_j(y) \right\}.$$

THEOREM 3.1. **SDP′** *correctly solves any instance of* **P′**, *i.e.,* **P** *with additional bounds of the form* $l_i \leqslant S_i \leqslant u_i$, $L_i \leqslant SI_i \leqslant U_i$, $i = 1, \ldots, n$, *if the underlying network for* **P′** *is a spanning tree.*

3.2. The General Networks Algorithm

First, even for concave costs the general network optimization problem cannot be reduced to the problem of finding an optimal spanning tree, i.e., whose solution is network optimal, as an easily constructed example in the OLS demonstrates. However, spanning tree relaxations are still useful if (i) solved using **SDP′**, which can handle nonconcave stage costs; and (ii) **SDP′**'s properties are appropriately exploited to move to better network solutions iteratively, although how to do so is not obvious, as comments in §3.3 show.

Our general networks algorithm (GNA) consists of a recursive routine $\mathcal{R}$ that uses **SDP′** to solve **P′**. Every call to $\mathcal{R}$ contains the description of a new problem **P′** to which more upper- and lower-bound constraints have been added. We augment our notation with Ω to refer to a set of constraints, and vector $\mathbf{S} = (S_1, SI_1, \ldots, S_n, SI_n)$ to compactly describe the set of inbound and outbound service times, the decision variables for **P′** and **P**.

Routine $\mathcal{R}$ takes as arguments the cost of the best solution found so far $\underline{z}$, a spanning tree T, and two sets of constraints Ω^G and Ω. It calls itself recursively and returns the optimal value z^* within the region described by Ω^G (for the cost function of **P**), as well as the associated solution $\mathbf{S}^*$ if $z^* \neq \infty$.

3.3. Optimality

Unlike a standard branch and bound, which bounds a single variable in each iteration, GNA bounds two or more variables in each iteration of $\mathcal{R}$, because **SDP′** cannot be used to get an optimal solution by bounding a single variable. For example, consider a network in which removing only one link (j, i) from the network forms a spanning tree. Suppose that the maximum possible value for S_j is 1 and that optimizing the spanning tree produces $S_j^* = 1$ and $SI_i^* = 0$. Then it is easy to see that following all the steps in GNA exactly, but using a standard branch and bound by bounding $S_j \leqslant 0$ first and then $S_j > 0$ instead of bounding $S_j \leqslant 0$ first and then $SI_i > 0$ as GNA would, will cause the algorithm to go into an infinite recursion. This is because the optimal solution produced by **SDP′** for the bound $S_j > 0$ remains unchanged from $S_j^* = 1$ and $SI_i^* = 0$. The claims below demonstrate the optimality of GNA, where F_Ω denotes the set of feasible solutions for the constraint set Ω.

GNA: Preprocessing and Initialization

Preprocessing: Select a spanning tree T from the given network G.

Initialization: Set $\underline{z} = \infty$ and both Ω^G, Ω to be the set of constraints of **P**.

Call $\mathcal{R}$: Call routine $\mathcal{R}$ with $\underline{z}$, T, Ω^G, and Ω as inputs. Let z^* be the optimal value returned by $\mathcal{R}$ and let $\mathbf{S}^*$ be the optimal solution if one exists. This is the solution to **P**.

REMARK 3.2. Every execution of routine $\mathcal{R}$ results in exactly one of the following outcomes:

(1) **SDP′** determines that F_Ω contains no better solution than one already found.

(2) **SDP′** determines that $F_\Omega = \varnothing$.

(3) **SDP′** finds a solution that is in F_Ω.

(4) **SDP′** finds a solution that is not in F_Ω. However, then after choosing an infeasible link $(j, i) \in \mathcal{A}$, $F_{\Omega \cup \{S_j \leqslant \bar{S}\}} \subset F_\Omega$, and $F_{\Omega \cup \{SI_k > \bar{S}, \forall k: (j,k)\in\mathcal{A}\}} \subset F_\Omega$.

GNA: Routine $\mathcal{R}$ (arguments $\underline{z}$, T, Ω^G, Ω)

Step 1: Let Ω^T be the subset of constraints in Ω that correspond to T, i.e., drop all constraints from Ω that correspond to links not in T. Call $\mathbf{P}^T$ the problem of optimizing the cost function of **P** subject to Ω^T.

Step 2: Use **SDP′** to obtain an optimal solution to $\mathbf{P}^T$. Let z^T be the optimal value of $\mathbf{P}^T$, and $\mathbf{S}^T$ the optimal solution if one exists.

Step 3: If $z^T \geqslant \underline{z}$ or $z^T = \infty$, return ∞ for the cost.

Step 4: Else, if $\mathbf{S}^T$ satisfies the constraints in Ω^G, set $\underline{z} = z^T$ and return z^T, $\mathbf{S}^T$.

Step 5: Else, $\mathbf{S}^T$ does not satisfy the constraints in Ω^G. Let $\hat{z}$ equal the cost of the solution $\hat{\mathbf{S}}$, which has $\hat{S}_i = S_i^T$, and $\widehat{SI}_i = \max_{(j,i)\in\mathcal{A}} \hat{S}_j$ for all stages i. If $\hat{z} < \underline{z}$, set $\underline{z} = \hat{z}$; otherwise, leave $\underline{z}$ unchanged. Then carry out the steps below.

(a) Choose a link $(j, i) \in \mathcal{A}$ for which $S_j^T > SI_i^T$, i.e., a constraint in Ω^G is violated. Let $\bar{S} = SI_i^T + \lfloor (S_j^T - SI_i^T)/2 \rfloor$.

(b) Carry out the *Upperbound* and *Lowerbound* steps below in order.

Upperbound: Let $z_u, \mathbf{S}_u$ be the solution returned by $\mathcal{R}$ for arguments $\underline{z}$, T, Ω^G, and the updated constraint set $\Omega \cup \{S_j \leqslant \bar{S}\}$.

Lowerbound: Let $z_l, \mathbf{S}_l$ be the solution returned by $\mathcal{R}$ for arguments $\underline{z}$, T, Ω^G, and the updated constraint set $\Omega \cup \{SI_k > \bar{S},\ \forall k\colon (j,k) \in \mathcal{A}\}$.

Return $\min(\hat{z}, z_u, z_l)$ and the associated solution (breaking any tie arbitrarily).

Outcomes 1, 2, and 3 in Remark 3.2 follow from Theorem 3.1 because $\mathbf{P}^T$ is a relaxation of $\mathbf{P}'$. Outcome 4 is easy to verify from how $\bar{S}$ is constructed in Step 5(a) in $\mathcal{R}$, and states that $\mathcal{R}$ calls itself with strictly smaller feasible regions in each iteration, so the algorithm is finite. The only remaining requirement for GNA's optimality is to show that the smaller feasible regions do not omit any part of the original feasible region, shown by the following proposition (proof in OLS).

PROPOSITION 3.3. *In any iteration of $\mathcal{R}$, for any infeasible link $(j,i) \in \mathcal{A}$, $F_{\Omega \cup \{S_j \leqslant \bar{S}\}} \cup F_{\Omega \cup \{SI_k > \bar{S},\ \forall k:(j,k)\in\mathcal{A}\}} \supset F_\Omega$.*

3.4. Performance

We report GNA's performance for a set of 38 real-world chains presented in Willems (2008). These chains represent actual business problems from 29 companies. Some data were disguised to protect company confidentiality, and some data were standardized to facilitate comparisons between chains, but attributes including the exact supply chain topology and the characterization of stage times were preserved.

Two reasons motivate testing the algorithm on a real data set instead of constructing an artificial test suite. First, the motivation for our algorithm was to optimize chains arising in practical settings. Second, problem **P** for arbitrary cost functions is so general that one can always construct uncommon examples to make the algorithm perform unacceptably. On the other hand, cost functions arising in practical settings, although not always closed form or concave, are still not completely arbitrary. Further, the empirical performance of branch-and-bound algorithms is known to depend strongly on the problem structure, the quality of relaxations used, the methods for solving these relaxations, and the partitioning rules. Testing on artificial problems would therefore give us less confidence about the algorithm's practicality than testing it on real-world problems.

Table 1 presents GNA's performance. The chains tested are all general networks with 15 chains between 100 and 499 stages, 6 between 500 and 999 stages, and 5 above 1,000 stages. They are reasonably dense, 17 having link-to-stage ratios greater than 1.5, 7 more than 3, and 1 greater than 8. Another major determinant of run time is maximum supply chain length. The chains have substantial maximum lengths; only 10 have length less than 50 days and 13 have lengths greater than 99 days.

GNA returns within approximately two minutes on 28 of the 38 chains (within a minute on 26 chains). On most of the chains where it exceeds 5 minutes, it does run very long; but looking at these cases, it seems the algorithm finds the optimal or a near-optimal solution in reasonable time, and spends the rest of the time checking infeasible solutions in the branch-and-bound tree; the *total iters* column is the total number of spanning-tree optimizations tried by the algorithm and the *optimal iter* column is the iteration that produced the optimal cost. This motivates the development of two heuristics in §4. One is based on the structure of the dual space and the other is a trivial heuristic that terminates the algorithm after only 100 spanning tree optimizations. In general, given the hardness of the problem with nonconcave costs and arbitrary acyclic networks, the performance of the algorithm is quite good and is typical of our experience in the last few years. The companies that contributed the data set are using an industrial-strength multiechelon inventory optimization software application based on the GS model that employs the TGNA heuristic defined in §4.1.

4. Faster Heuristics

Results from Table 1 suggest that the optimal solution is often found quickly by GNA. This motivates questions about deeper theoretical results, as well as a search for heuristics that can terminate quickly and still be close to optimal. Although we have not been able to fully characterize the reasons yet, we do propose the following heuristic, which works well in practice. It is based on an understanding of the algorithm's mechanics in the dual space (see the OLS). This heuristic (HGNA) does three things differently from GNA, keeping all other implementation details as described in the OLS.

(1) During the *Initialization* step, creates a set of links Π initialized to $\varnothing$, for which the *Lowerbound* step in routine $\mathcal{R}$ is not to be tried.

(2) Modifies the recursive routine $\mathcal{R}$ to know if it is *being called* from the *Lowerbound* step during an execution of $\mathcal{R}$ (adding a flag to $\mathcal{R}$'s arguments). The first call to $\mathcal{R}$ is with the flag *Upperbound*.

(3) Modifies *Step* 5 of $\mathcal{R}$ as described below.

The heuristic modifies the bounding rule for an infeasible link (j,i) to upper-bound S_k for *all* stages upstream of i and lower-bound *only* SI_i. However, the reason for its efficiency is that it prunes the branch-and-bound tree by never using the *Lowerbound* step again once the link has been added to Π, no matter how many times that link is found infeasible during the branch-and-bound recursion. The OLS presents the motivating intuition for the heuristic.

HGNA: Modified Routine $\mathcal{R}$ (arguments $\underline{z}$, T, Ω^G, Ω, Π, *Upperbound/Lowerbound* flag)

Step 1 to *Step* 4: Same as in **GNA**

Step 5: Else, $\mathbf{S}^T$ does not satisfy the constraints in Ω^G. Let $\hat{z}$ equal the cost of the solution $\hat{\mathbf{S}}$, which has $\hat{S}_i = S_i^T$, and $\widehat{SI}_i = \max_{(j,i)\in\mathcal{A}} \hat{S}_j$ for all stages i. If $\hat{z} < \underline{z}$, set $\underline{z} = \hat{z}$; otherwise, leave $\underline{z}$ unchanged. Then carry out the steps below.

(a) Choose a link $(j,i) \in \mathcal{A}$ for which $S_j^T > SI_i^T$, i.e., a constraint in Ω^G is violated. Let $\bar{S} = SI_i^T + \lfloor (S_j^T - SI_i^T)/2 \rfloor$.

(b) Carry out the *Upperbound*, *Lowerbound*, and *Update* Π steps below in order.

Upperbound: Let $z_u, \mathbf{S}_u$ be the solution returned by $\mathcal{R}$ for arguments $\underline{z}$, T, Ω^G, the updated constraint set $\Omega \cup \{S_k \leqslant \bar{S},\ \forall k:\ (k,i)\in\mathcal{A}\}$, and the *Upperbound* flag.

Table 1. GNA performance.

Chain	Stages	Links	Max chain length	Time (secs)	Optimal cost ($)	Total iters	Optimal iter
01	8	10	38	0.00	3.35E+04	9	1
02	13	13	64	0.03	9.51E+06	9	1
03	17	18	79.8	0.11	6.94E+06	19	17
04	22	39	204	0.63	4.90E+04	19	9
05	27	31	47.35	0.00	2.01E+06	1	1
06	28	28	96	0.00	7.78E+04	1	1
07	38	78	85	1.47	5.22E+06	101	15
08	40	48	91.04	0.28	1.63E+06	11	5
09	49	52	47.38	0.02	1.12E+06	1	1
10	58	176	162	0.08	1.21E+06	15	11
11	68	108	60	9.50	1.12E+07	267	226
12	88	107	108.6	10.67	7.35E+06	391	359
13	108	452	26	22.59	6.09E+06	25,695	132
14	116	119	128.32	0.13	7.16E+04	1	1
15	133	164	26	0.02	1.16E+06	1	1
16	145	224	163	30.03	4.64E+06	79	63
17	152	211	57	0.02	1.09E+06	1	1
18	154	224	100	2.09	9.75E+04	33	26
19	156	263	125	3,103.63	3.15E+05	59,061	58,681
20	156	169	160.9	804.39	2.91E+05	16,699	14,514
21	186	359	96	140.34	1.08E+06	1,983	59
22	253	253	691	12.19	1.94E+06	1	1
23	271	524	77	176,722.45	4.26E+05	10,000,000	108
24	334	1,245	68.53	10,435.67	1.41E+07	127,521	5,107
25	409	853	82	883,312.55	1.14E+06	10,000,000	3,138,047
26	468	605	394.07	8.38	4.53E+06	21	11
27	482	941	105	705,636.17	8.32E+05	10,000,000	8,428,498
28	577	2,262	123	103.02	4.63E+06	389	180
29	617	753	43	0.75	3.46E+05	11	6
30	626	632	71.05	0.69	9.60E+05	5	2
31	706	908	17.92	116.22	3.40E+07	4,065	3,894
32	844	1,685	112.2	3,089,722.84	7.09E+05	10,000,000	5,535,857
33	976	1,009	72.36	0.28	4.22E+05	1	1
34	1,206	4,063	89	12.66	8.64E+05	147	71
35	1,386	1,857	81	37.88	1.79E+06	235	192
36	1,451	4,812	49.55	126,583.03	1.24E+06	781,737	11,235
37	1,479	2,069	27.85	17.28	9.99E+05	253	180
38	2,025	16,225	26.03	577,190.78	1.10E+06	2,839,444	65,507

Notes. All chains were optimized from the starting solution $S_i = 0$, $i = 1, \ldots, n$. *Total Iters* were capped at 10 m.

Lowerbound: If $(j, i) \in \Pi$, let $z_l = \infty$. Otherwise, let z_l, $\mathbf{S}_l$ be the solution returned by $\mathcal{R}$ for arguments $\underline{z}$, T, Ω^G, the updated constraint set $\Omega \cup \{SI_i > \bar{S}\}$, and the *Lowerbound* flag.

Update Π: If $\mathcal{R}$ was called with the *Lowerbound* flag, set $\Pi = \Pi \cup \{(j, i)\}$.

Return $\min(\hat{z}, z_u, z_l)$ and the associated solution (breaking any tie arbitrarily).

4.1. Performance Comparison

Table 2 compares the performance of GNA, HGNA, and TGNA (the GNA algorithm truncated to at most 100 iterations) relative to GNA's results (a table with actual run time is provided in the OLS). HGNA runs within a minute for all 38 chains except 4, for which it runs in approximately 2.4, 3.7, 17.1, and 28.5 minutes, for which GNA run time is of the order of days. HGNA finds the optimal solution for 28 of the 38 chains and is off by more than 1% for only 4 chains (2.32%, 3.61%, 15.67%, and 16.66%). TGNA runs within a minute on all chains, as expected, and finds the optimal solution for 25 of the 38 chains. More importantly, for the remaining chains, TGNA's optimality gap is more than 5% for only 3 chains, although the worst-case optimality gaps are approximately 37% and 39%. However, the chain with 37% optimality gap is optimized by GNA in 132 iterations, so a slight increase in the truncation limit would have allowed TGNA to find the optimal solution as well. TGNA's performance reinforces the results of Table 1, i.e., that GNA finds the optimal or a near-optimal solution quickly, but might take significant time trying to determine if the solution it has found is the globally optimal solution.

5. Summary

We have extended the GS framework along two significant dimensions: incorporating generalized stage costs,

Table 2. Performance of HGNA and TGNA relative to GNA.

				GNA algorithm				HGNA heuristic % change (%)				TGNA heuristic % change (%)			
Chain	Stages	Links	MaxChain length	Time (secs)	Optimal cost ($)	Total iters	Optimal iter	Time	Optimal cost	Total iters	Optimal iter	Time	Optimal cost	Total iters	Optimal iter
01	8	10	38	0.00	3.35E+04	9	1	0	0	0	0	2	0	0	0
02	13	13	64	0.03	9.51E+06	9	1	0	0	0	0	3	0	0	0
03	17	18	79.8	0.11	6.94E+06	19	17	−43	0	−42	−47	−44	0	−42	−47
04	22	39	204	0.63	4.90E+04	19	9	−15	0	−21	−67	2	0	0	0
05	27	31	47.35	0.00	2.01E+06	1	1	0	0	0	0	2	0	0	0
06	28	28	96	0.00	7.78E+04	1	1	2	0	0	0	2	0	0	0
07	38	78	85	1.47	5.22E+06	101	15	−81	0	−80	−60	−1	0	−1	0
08	40	48	91.04	0.28	1.63E+06	11	5	−5	0	0	0	−6	0	0	0
09	49	52	47.38	0.02	1.12E+06	1	1	0	0	0	0	−100	0	0	0
10	58	176	162	0.08	1.21E+06	15	11	−21	0	−27	−18	0	0	0	0
11	68	108	60	9.50	1.12E+07	267	226	−93	0	−93	−96	−73	0	−73	−86
12	88	107	108.6	10.67	7.35E+06	391	359	−91	0	−91	−94	−80	0	−80	−87
13	108	452	26	22.59	6.09E+06	25,695	132	−100	0	−100	−98	−100	36.90	−100	−99
14	116	119	128.32	0.13	7.16E+04	1	1	13	0	0	0	12	0	0	0
15	133	164	26	0.02	1.16E+06	1	1	7	0	0	0	0	0	0	0
16	145	224	163	30.03	4.64E+06	79	63	−17	0	−18	−35	−12	0	−13	−16
17	152	211	57	0.02	1.09E+06	1	1	0	0	0	0	94	0	0	0
18	154	224	100	2.09	9.75E+04	33	26	18	0	9	−19	−26	0	−24	−31
19	156	263	125	3,103.63	3.15E+05	59,061	58,681	−100	0.85	−100	−100	−100	1.44	−100	−100
20	156	169	160.9	804.39	2.91E+05	16,699	14,514	−100	3.61	−100	−100	−99	3.52	−99	−100
21	186	359	96	140.34	1.08E+06	1,983	59	−95	0	−96	−44	−95	0	−95	0
22	253	253	691	12.19	1.94E+06	1	1	−4	0	0	0	0	0	0	0
23	271	524	77	176,722.45	4.26E+05	10,000,000	108	−100	0	−100	−49	−100	0.10	−100	−7
24	334	1,245	68.53	10,435.67	1.41E+07	127,521	5,107	−100	2.32	−100	−95	−100	2.84	−100	−98
25	409	853	82	883,312.55	1.14E+06	10,000,000	3,138,047	−100	0.12	−100	−100	−100	0.75	−100	−100
26	468	605	394.07	8.38	4.53E+06	21	11	−57	0	−57	−55	0	0	0	0
27	482	941	105	705,636.17	8.32E+05	10,000,000	8,428,498	−100	−9.40	−100	−100	−100	2.39	−100	−100
28	577	2,262	123	103.02	4.63E+06	389	180	−82	0.01	−83	−85	−74	2.18	−74	−45
29	617	753	43	0.75	3.46E+05	11	6	0	0	0	0	2	0	0	0
30	626	632	71.05	0.69	9.60E+05	5	2	87	0	80	0	0	0	0	0
31	706	908	17.92	116.22	3.40E+07	4,065	3,894	−95	0.11	−94	−100	−98	2.46	−98	−100
32	844	1,685	112.2	3,089,722.84	7.09E+05	10,000,000	5,535,857	−100	0.01	−100	−100	−100	0.02	−100	−100
33	976	1,009	72.36	0.28	4.22E+05	1	1	6	0	0	0	0	0	0	0
34	1,206	4,063	89	12.66	8.64E+05	147	71	−26	0	−33	−42	−32	0	−32	0
35	1,386	1,857	81	37.88	1.79E+06	235	192	−81	0.15	−82	−97	−57	0.58	−57	−60
36	1,451	4,812	49.55	126,583.03	1.24E+06	781,737	11,235	−99	15.67	−99	−95	−100	15.78	−100	−99
37	1,479	2,069	27.85	17.28	9.99E+05	253	180	−85	0	−85	−86	−60	0	−60	−78
38	2,025	16,225	26.03	577,190.78	1.10E+06	2,839,444	65,507	−100	16.66	−100	−100	−100	38.79	−100	−100

Note. Shows percent reduction in run time, total iterations required and the iteration when the best solution was found, and percent increase in optimal cost.

and optimizing safety stocks in directed acyclic general networks. We have presented the first algorithm in our knowledge for the NP-hard problem of safety stock optimization in general networks with generalized stage costs. We have presented a proof of its optimality. We have presented the first computational results for a published data set of real-world chains (from 8 to 2,025 stages) to demonstrate our algorithm's performance and have found the algorithm's performance consistently good in practice. We have also presented two near-optimal but much faster heuristics and tested them on the same data set. Finally, our investigation of the problem structure in the dual space (in the OLS) has revealed what we believe are promising research directions for fast future algorithms.

6. Electronic Companion

An electronic companion to this paper is available as part of the online version that can be found at http://or.journal.informs.org/.

References

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. 2003. Erratum: Optimizing strategic safety stock placement in supply chains. *Manufacturing Service Oper. Management* **5**(2) 176–177.

Humair, S., S. P. Willems. 2006. Optimizing strategic safety stock placement in supply chains with clusters of commonality. *Oper. Res.* **54**(4) 725–742.

Karimi, I. A., J. Shu, M. Song. 2004. Heuristic methods for inventory placement in general acyclic supply chains. Working paper, National University of Singapore and Logistics Institute–Asia Pacific, Singapore, 25.

Lesnaia, E. 2004. Optimizing safety stock placement in general network supply chains. Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA.

Magnanti, T. L., Z.-J. M. Shen, J. Shu, D. Simchi-Levi, C.-P. Teo. 2006. Inventory placement in acyclic supply chain networks. *Oper. Res. Lett.* **34**(2) 228–238.

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

Willems, S. P. 2008. Real-world multiechelon supply chains used for inventory optimization. *Manufacturing Service Oper. Management* **10**(1) 19–23.