New model and heuristics for safety stock placement in general acyclic supply chain networks

Haitao Li a,*, Dali Jiang b

a Department of Logistics and Operations Management, College of Business Administration, University of Missouri—St. Louis, One University Blvd, St. Louis, MO 63121, USA
b Research Institute of Logistics, Chongqing Logistics Engineering University, Chongqing 400016, P.R. China

ARTICLE INFO

Keywords:
Safety stock placement
Inventory positioning
General acyclic network
Project scheduling
Constraint programming (CP)
Genetic algorithm (GA)

ABSTRACT

We model the safety stock placement problem in general acyclic supply chain networks as a project scheduling problem, for which the constraint programming (CP) techniques are both effective and efficient in finding high quality solutions. We further integrate CP with a genetic algorithm (GA), which improves the CP solution quality significantly. The performance of our hybrid CP–GA algorithm is evaluated on randomly generated test instances. CP–GA is able to find optimal solutions to small problems in fractions of a second, and near optimal solutions of about 5% optimality gap to medium size problems in several minutes on average.

© 2011 Elsevier Ltd. All rights reserved.

1. Introduction

Supply chains are complex networks integrating suppliers/vendors, manufacturers, distributors and retailers to enable physical entities (raw materials, parts, components, semi-finished and finished products) to be produced and distributed at the right quantities, to the right locations and at the right time. Demand uncertainty requires safety stock to be kept through the supply chain network to meet customers’ order and delivery needs. The problem of determining the location and level of safety stock to keep in the supply chain is known as the inventory positioning or safety stock placement problem (cf. [1,2]). Unlike the traditional strategic supply chain network design problems, the safety stock placement problem addresses the tactical level design for an existing physical supply chain, by optimally balancing system-wide inventory cost and service level.

Optimizing the supply chain system-wide safety stock is crucial in today's complex, competitive and volatile market environment. Since a modern supply chain rarely involves a single isolated firm, the focus here is not limited to coordinating internal materials and inventory management in the traditional material requirement planning (MRP) setting, but also involves external suppliers, vendors, logistics providers and manufacturers ([3], p.1). With the rapid growth of business-to-business (B2B) e-commerce facilitated by the electronic data interchange (EDI) technology and availability of point of sales (POS) data, the ability to redefine inter-firm relationships and coordinate activities among supply chain players becomes a key for supply chain success. Such external coordination calls for collaboration among supply chain players, so that decisions can be made in a centralized way. Sometimes the centralized decision-making paradigm is natural. For instance, the logistics of the U.S. Army's spare parts [4] and the U.S. Navy's depot level repairable (DLR) line items [5] require inventory positioning decisions to be made in a centralized fashion to support military missions. In the business environment, however, external collaboration and coordination need a well-defined incentive and often comes at a price. Examples of real world practices include the continuous replenishment process (CRP), vendor managed inventory (VMI) [6] and supply chain partnership [7]. In the safety stock placement problem addressed in this paper, we assume that supply chain players fully collaborate such that centralized safety stock placement decision can be made. The issues of creating incentive, negotiating and managing such collaborative relationship go beyond the scope of this paper.

Safety stock placement is also related to determining the push-pull boundary or implementing the delay differentiation (postponement) strategy in multi-echelon supply chains [8]. An isolated manufacturer often operates either make-to-stock (MTS) or make-to-order (MTO). In an MTS process, components and finished goods are manufactured based on long-term forecasting and planning, and held as inventories to meet the anticipated future demand. MTS is sometimes referred to as the push strategy. In an MTO process, manufacturing is only triggered by a realized order. MTO is often referred to as the pull strategy. In the setting of an integrated supply chain, firms do not have to implement a pure push or pull strategy. By properly coordinating the activities of supply chain players, firms may benefit from storing inventories at certain intermediate stages in the network, giving rise to the hybrid MTS/MTO or push–pull strategy. We refer to [9] for an application of configuring a laptop PC supply chain.
We consider a supply chain network consisting of functions, e.g., parts, components or processes, and the demand dependency among them as characterized by a bill-of-materials (BOM). We assume that the network is acyclic, i.e., no directed cycle (“feedback”) exists. Each sink node faces a Gaussian normal-distributed customer demand to be satisfied. Such external demand will be propagated backward along the supply chain through the demand dependencies in BOM. Each upstream function will quote a guaranteed delivery time, called outbound service time, to satisfy the demand of its immediately downstream function(s). Since the promised delivery time for the finished goods is often smaller than the total supply chain cycle time, certain amount of safety stock needs to be kept at each function to maintain certain service level. We assume that each function operates under a periodic review policy. Accordingly, the level of safety stock for the corresponding function is determined by its net replenishment time. In addition, there might be capacity limit for storage of safety stock that cannot be exceeded at each function. The safety stock placement problem addressed in this paper seeks to find optimal inbound and outbound service times of each function in the supply chain to minimize the system-wide safety stock cost.

Complexity of the safety stock placement problem is directly driven by structure of the supply chain. Fig. 1 illustrates three different supply chain structures. Among them, the serial structure in Fig. 1(a) considers only sequential dependencies among supply chain functions, resulting a linear/serial structure. Fig. 1(b) illustrates the spanning tree structure. Clearly, the spanning tree structure is able to model more complex interactions among supply chain functions than the serial structure. For instance, for every unit of component/part assembled at manufacturer C, 2 units from supplier A and 1 unit from supplier B are required. At the same time, manufacturer C cannot start its inbound service until its demand for all its immediate predecessors, A and B, has been satisfied. Through such demand- and time-dependencies, the decision at one location will be propagated to other functions in the chain. For instance, by promising a prompt delivery lead time to its customers, a car dealer will not only potentially carry more safety stock himself, but would also like to require an early receipt of the order from his warehouse, which creates pressure for the warehouse to carry more inventory as well. Two special cases of the spanning tree structure include the assembly (convergent) network where each node has at most one outgoing arc, and the distribution (divergent) network where each node has at most one incoming arc. The structure of real-life supply chains often goes beyond the complexity of a spanning tree, i.e., the number of arcs in the network will often exceed the number of functions minus one. These supply chains are characterized by a general acyclic network as depicted in Fig. 1(c). More general cases include those where arcs may exist between an arbitrary pair of supply chain functions as shown in Fig. 1(d).

It is well-known that the safety stock placement problem in networks with special structure such as the serial and spanning tree structures can be efficiently solved by dynamic programming (cf. [10,11]). The one with the general acyclic structure involves minimizing a concave function, and it is well-known that the general problem of minimizing a concave function is \( \text{NP-hard} \) [12].

Networks with general structure have been a focal subject in the research field of project scheduling. Since the development of the well-known critical path method (CPM; [13]) and program evaluation and review technique (PERT; [14]) in the early sixties, the project scheduling field has devised numerous efficient solution methods for scheduling activities in large-size general project networks (cf. [15,16]). This has motivated us to resort to the modeling and solution techniques in project scheduling to address the safety stock placement problem in general acyclic supply chain networks.

In our modeling framework, an acyclic supply chain network is transformed into a project scheduling network, with each supply chain function being modeled as a project activity. We show that time-related dependencies among inbound and outbound service times can be modeled as temporal relationships among project activities. Then the problem of finding optimal inbound and outbound service times becomes equivalent to the problem of scheduling project activities’ starting times. Such a transformation enables us to borrow a large set of well-developed algorithms in the project scheduling field (cf. [15]) to tackle the safety stock placement problem. In this paper, we devise a constraint-programming (CP; [17]) based algorithm to solve the transformed problem. We embed heuristic branching rules exploiting the scheduling characteristics of the problem to enhance CP’s performance. One advantage of the CP algorithm is its ability to generate high-quality feasible solutions fast. When used as initial solutions, these feasible solutions obtained by CP can be improved by a number of local search [18] and metaheuristic methodologies [19]. We design a hybrid framework that integrates CP and genetic algorithm (GA), called CP–GA, to enhance solution quality. Computational results show that our CP–GA algorithm is able to find optimal solutions to small instances of the problems within fractions of a second, and obtain near optimal solutions to medium size problems in reasonable computational time.

The remainder of our paper is organized as follows. Section 2 reviews the related literature. We present the project-scheduling-based modeling framework and CP model in Section 3. Section 4 describes the CP solution procedures including constraint propagation and various search strategies devised in this study. The hybrid CP–GA algorithm is presented in Section 5. The computational results are provided in Section 6. Section 7 draws conclusions and suggests future research directions.

2. Related literature

The safety stock problem with simple supply chain structure, such as the serial and spanning tree structures, has been extensively studied. For a serial supply chain, an “all or nothing” strategy is suggested by [20]. The effect of stochastic lead time on inventory positioning is studied in [21]. Inventory positioning in multiple
product supply chains with serial structure has been studied in [22]. A serial structure network can be extended to an assembly network where each node has at most one outgoing arc, or a distribution network where each node has at most one incoming arc. Both are special cases of the spanning tree structure [23]. Efficient dynamic programming algorithms have been proposed for serial, divergent (distribution), or convergent (assembly) networks ([10,24]). A pseudo-polynomial dynamic programming algorithm was proposed by [11] for a spanning tree supply chain that consists of a mixture of assembly and distribution sub-networks.

Recent research efforts have focused on the more challenging and realistic general acyclic supply chain networks. It is shown that the inventory positioning problem in a general supply chain network is NP-hard [25]. Exact methods such as enumeration and branch-and-bound [25] are often not able to handle real-world problems due to prohibitive amount of computational time for large instances. A piecewise linear approximation algorithm was devised by [1] to iteratively generate linear pieces to approximate the concave objective function. Some redundant constraints are formulated to generate strong flow cover cuts. Two heuristics based on linear approximation and truncated piecewise MIP formulation (with only two linear pieces) were developed in [2], which are shown to be efficient for obtaining near optimal solutions: the iterative LP method can efficiently solve problems with size as large as 8000 nodes and 32,000 arcs.

Project scheduling based approaches have been proposed for various manufacturing operations such as order selection, manufacturing planning and scheduling in the make-to-order environment [26]. A project scheduling network has been used to model a generalized supply chain to minimize the supply chain cycle time [27].

By transforming a supply chain network into a project scheduling network, a large set of well-developed project scheduling algorithms can be applied. Effectively and efficiently solving complex scheduling problems can often be achieved by constraint programming (CP) based methods. CP originated in the artificially intelligence (AI) area for solving constraint satisfaction problem (CSP). We refer to [17] for a general introduction to CP. With effective domain reduction (also called constraint propagation) algorithms have been successfully applied for solving a variety of NP-hard scheduling problems (cf. [30]).

For large size scheduling problems, however, CP alone is often not efficient due to the burden of searching the entire solutions space (although reduced through constraint propagation). There exist a number of integration schemes to take advantage of the complementary strengths of CP and other solution methods. CP has been used as a preprocessing procedure to reduce the size of project scheduling problems with resource constraints [31]. CP can also be integrated with mixed-integer linear programming (MILP) in the frameworks of Benders decomposition [32], Lagrangian relaxation [33] or column generation [34]. Schemes for integrating CP and local search (LS) have also been proposed [35]. We refer to [36,37] and [38] for surveys on CP-based hybrid algorithms.

3. Problem description

We now formally describe the safety stock placement problem in general acyclic supply chain networks, then present its CP formulation.

3.1. Problem description

Consider a general acyclic supply chain network $G(N,A)$, with the node set $N$ representing the set of supply chain functions, and the arc set $A$ denoting the set of demand dependencies and temporal relationships among functions. A function can be a part, component or a production/assembly process as characterized by the BOM associated with the product (or family of products). A typical example of general acyclic supply chain network is illustrated in Fig. 2, which consists of 5 stages (echelons) and 17 supply chain functions. It is a “general” network in which it includes the spanning tree network as a special case (i.e. the number of arcs exceeds the number of nodes minus one), and also allows that arbitrary arcs exist among supply chain functions. For instance, Part-1 purchased at the procurement stage is required for manufacturing Component-1 at the manufacturing stage, as well as assembling finished product A1 at the assembly stage. The network in Fig. 2 is “acyclic” as no directed cycle exists.

An arc $(ij) \in A$ specifies the demand dependencies between functions $i$ and $j$. For instance, one unit of Component-1 may require 2 units of Part-1 and 1 unit of Part-2. An arc $(ij)$ also specifies the temporal (time-related) relationships between function $i$ and $j$. We assume that the downstream function $j$ cannot start its $inbound$ service time until its upstream (predecessor) function $i$ has been delivered (i.e. its $outbound$ service has been fulfilled). Following [1] and [2], each function $j$ is assumed to have a constant lead time $T_j$ and a safety stock holding cost rate $h_j$ per item per time unit. There is a capacity limit of $C_j$ time periods of safety stock for each function $j$. The set $D$ of end products at the demand stage are assumed to face stochastic external demands bounded by a non-decreasing concave function, which is needed to establish a meaningful upper bound on demand over varying horizon for each end product. We refer to [11] for detailed justifications for this bounded demand assumption. When the external demand is normally distributed $(\mu, \sigma^2)$ and i.i.d., the demand bound can be expressed as $\tau \mu + k \sigma \sqrt{\tau}$, where $\tau$ is the net replenishment time, and $k$ is the $z$-value associated certain service level. Then the safety stock takes the form $k \sigma \sqrt{\tau}$ and is proportional to $\sigma \sqrt{\tau}$, which is commonly used in the literature (cf. [1–3]). In our example, the set $D$ includes three different retailers.
Instead of relying on the above nonlinear programming formulation, the capacitated safety stock placement problem can be written as

\[ \min \sum_{j} h_{j} (S_{j} + T_{j} - S_{j}) \]  

subject to:

\[ S_{j} - S_{i} \leq T_{i}, \forall j \in N \]  

\[ S_{j} - T_{j} \geq 0, \forall (i,j) \in A \]  

\[ S_{j} \leq d_{j}, \forall j \in D \]  

\[ S_{j} + T_{j} - S_{j} \leq C_{j}, \forall j \in N \]  

\[ S_{j}, S_{i} \in Z^{+}, \forall j \in N \]  

The objective function (1) minimizes the total safety stock cost throughout the entire supply chain, where \( h_{j} \Phi_{j} \) denotes the level of safety stock at function \( j \) in terms of days of demand as a function of net replenishment lead time, and is assumed to be non-negative. Constraint (2) enforces that the net replenishment lead time \( S_{j} + T_{j} - S_{j} \) of each function \( j \) must be no less than the outgoing service time of the upstream function \( i \). Constraint (3) specifies the precedence relationship between a pair of functions \( (i,j) \in A \). The inbound service time of downstream function \( j \) is constrained to be no less than the outbound service time of upstream function \( i \). Constraint (4) ensures finished products are delivered to customers within the deadline on lead time. Constraint (5) satisfies the capacity constraint, i.e., the net replenishment time of function \( j \) cannot exceed \( C_{j} \). When the lead time is long and/or deadline is tight, but the available capacity is small, a problem instance might be infeasible. Following [9] and [11], we require that both \( S_{j} \) and \( S_{i} \) be integer, which is in line with the fundamental periodic review policy assumed for each \( j \). We refer to [9] for detailed justifications on the rationale of this integrality requirement. Note that the integrality constraints make our model more restrictive than the formulation in [1] and [2], although integer solutions can often be guaranteed as long as all the input data are integers, given the intrinsic pure network structure of the problem [1].

The safety stock placement problem described by (1)–(6) is a separable concave minimization problem, which is well-known to be NP-hard [12]. Instead of relying on the above nonlinear programming formulation, our approach transforms the addressed safety stock placement problem into a project scheduling problem, and utilizes constraint programming (CP) based methods to solve it. The CP formulation is presented next.

### 3.2. Constraint programming formulation

Our project-scheduling-based framework for the safety stock placement problem models the entire supply chain as a project. For each supply chain function \( j \in N \), we define a real variable \( a[j] \) representing the execution of function \( j \), and a dummy activity \( b[j] \) representing the outbound service of \( j \). For illustration purpose, we use the CP constructs available in ILOG OPL Studio [39] and

\[ S_{j} \] represents the execution of function \( j \), and a dummy activity \( b[j] \) representing the outbound service of \( j \). For illustration purpose, we use the CP constructs available in ILOG OPL Studio [39] and

\[ b[j] \] is the promised outbound service time (represented by a triangle in the chart). For instance, the promised outbound service time of downstream function \( j \) is to find feasible starting times of all real and dummy activities.

The value in parenthesis specifies the duration of an activity, i.e., the lead time \( T_{j} \) for a real activity \( a[j] \), and 0 for a dummy activity \( b[j] \).

Such a modeling framework is depicted by the Gantt Chart in Fig. 3. Consider a pair of directly connected functions Function—1 and Function—2. Each function is associated with a real activity representing the actual execution of the function (represented by a rectangle in the chart) and a dummy activity denoting the promised outbound service time (represented by a triangle in the chart). The dummy activity of each function has to start before the end of the corresponding real activity to ensure a non-negative net replenishment time. The length of net replenishment time is represented by a grayed rectangle in the chart. For instance, the net replenishment time of Function—1 equals the difference between the end of the corresponding real activity and the start of the dummy activity, i.e., \( S_{1} + T_{1} - S_{1} \). At the same time, the downstream Function—2 cannot start before the start of the dummy activity of the upstream Function—1. Such a time-dependency is represented by a solid arrow in the chart. The goal is to find feasible starting times of all real and dummy activities satisfying the temporal constraints, such that the total inventory cost expressed as a nonlinear function of these starting times is minimized.

Based on the above modeling framework, we now present the CP formulation.

#### 3.2.1. Non-negative net replenishment time

To ensure non-negative net replenishment time for each supply chain function \( j \in N \), the associated dummy activity \( b[j] \) must start before the end of real activity \( a[j] \):

\[ b[j].startsBeforeEnd(a[j]), \forall j \in N \]  

Constraint (9) is equivalent to Constraint (2).

#### 3.2.2. Precedence relationships

For every pair of arc \( (i,j) \in A \), the real activity \( a[j] \) associated with function \( j \) must not start until the start of the dummy activity \( b[i] \) associated with function \( i \):

\[ a[j].startsAfterStart(b[i]), \forall (i,j) \in A \]  

Constraint (10) is equivalent to Constraint (3).
3.2.3. Guaranteed customer delivery lead time
For each finished product \( j \in D \), the associated dummy activity \( b[j] \) must start before the customer-specified delivery deadline \( d[j] \) on delivery lead time:
\[
b[j] \text{ startsBefore } (d[j]) \quad \forall j \in D
\]
Constraint (11) is equivalent to Constraint (4).

3.2.4. Capacity constraints
In our CP formulation, the capacity constraint (5) becomes
\[
b[j] \text{ startsAfterStart } (a[j], T_j - C_j), \quad \forall j \in N
\]
where \( T_j - C_j \) is a minimum time-lag between the start of dummy activity \( b[j] \) and the start of real activity \( a[j] \) associated with supply chain function \( j \).

3.2.5. Objective function
Using CP constructs, the objective function (1) can be written as
\[
\min \sum_{j \in N} \phi_j (a[j], \text{start} + T_j - b[j], \text{start})
\]
where \( a[j], \text{start} \) and \( b[j], \text{start} \) denote the starting time of real activity \( a[j] \) and dummy activity \( b[j] \), respectively.

4. CP algorithm
This section describes the CP algorithms to solve the safety stock placement problem. A CP algorithm consists of two main solving techniques: constraint propagation and search. Constraint propagation is a problem reduction technique that transforms problems into equivalent problems, which are easier to solve. It modifies the domains of all variables in a constraint given the change of one of the variables in that constraint. We refer to [41] for theoretical background of constraint propagation and definitions of different consistency concepts. A search procedure is needed to explore the remaining solution space, as problem reduction through constraint propagation alone is often NP-hard.

4.1. Constraint propagation
Two types of constraint propagation are implemented in our CP algorithm: I. the initial constraint propagation to reduce the solution domain before the search starts; II. constraint propagation during the search.

We use temporal analysis in project scheduling to deduce the time windows of both real and dummy activities, i.e. the type I constraint propagation. Following [16], we solve the following two linear programs (LP):

**ES LP**
\[
\begin{align*}
\text{min} \quad & \sum_{j \in N} (S_j + S_j) \\
\text{s.t.} \quad & \text{Constraints (2)-(6)} \\
\end{align*}
\]
and

**LS LP**
\[
\begin{align*}
\text{max} \quad & \sum_{j \in N} (S_j + S_j) \\
\text{s.t.} \quad & \text{Constraints (2)-(6)} \\
\end{align*}
\]

ES LP obtains the earliest start (ES) times of real and dummy activities, and LS LP obtains the latest start (LS) times. The time window \([ES_j, LS_j] \) of real/dummy activity \( j \) reduces the domain of decision variables.

Constraint propagation during the CP search, the type II constraint propagation, is based on the fundamental concept of consistency, and specifically, the arc-consistency (AC) for binary constraints in constraint satisfaction problems (CSP, [17,41]).

**Definition 1.** A constraint is called binary constraint if it involves exactly two decision variables.

It is important to note that our entire constraint system consists of mainly binary constraints, i.e. (9)-(12), for which AC algorithms will be especially effective [37]. We now define AC in the setting of our CP model.

**Definition 2.** Let \( X \rightarrow a \) denotes an assignment of value \( a \) to variable \( X \). A binary constraint \( C \) involving two start times \( SL \) and \( SL_y \) of functions \( x \) and \( y \), respectively, is arc-consistent, if \( \forall t \in [ES_x, LS_x], \exists t' \in [ES_y, LS_y] \) such that \( (SL_x - t, SL_y - t') \) satisfies \( C \), and \( \forall t' \in [ES_y, LS_y], \exists t \in [ES_x, LS_x] \) such that \( (SL_x - t, SL_y - t') \) satisfies \( C \).

In other words, constraint \( C \) is said to be arc-consistent, if for every value \( t \) in the time window of \( x \), there always exists a value \( t' \) in the time window of \( y \) satisfying \( C \); at the same time, for every \( t' \) in the time window of \( y \), there exists a value \( t \) in the time window of \( x \) satisfying \( C \).

Define \( S \) as the set of all decision variables, i.e. \( S = SL_x \cup LS_x \cup SL_y \cup LS_y \). and \( W \) as the set of time windows (domains) of variables. Let \( C \) consist of all binary constraints of (9)-(12). The fundamental AC algorithm, known as AC-1 [41], for achieving arc-consistency of the safety stock placement problem is described below.

**Procedure:** AC-1 (S,W,C)

**Step 1.** Construct constraint list: \( \mathcal{Q} \leftarrow (C_{x,y}) | (C_{x,y}) \in C \); Step 2. Examine each constraint in the list: Repeat

- DomainReduced \( \leftarrow \) false;
- For each \( C_{x,y} \in \mathcal{Q} \) Do
- DomainReduced \( \leftarrow \) Update Domain \( (C_{x,y}), S, W, C \)
- Until Not DomainReduced
- Return \( (S,W,C) \).

Step 1 constructs a list \( \mathcal{Q} \) of binary constraints, each of which involves two variables \( x \) and \( y \), to be checked for consistency. Step 2 examines each constraint \( C_{x,y} \) in the list \( \mathcal{Q} \), and removes all values, which do not satisfy \( C_{x,y} \) by calling the sub-procedure Update Domain. If the domain can be reduced, Step 2 is repeated; otherwise, the AC-1 procedure is terminated. The sub-procedure Update Domain is described below:

**Sub-Procedure:** Update Domain \( (C_{x,y}), S, W, C \)

**Step 1.** Initialization: Deleted \( \leftarrow \) false;
- Step 2. Check domain of variables:
- For each \( t \in W_x \) Do
- If \( \exists t' \in W_y \) such that \( (x \leftarrow t, y \leftarrow t') \) satisfies \( C_{x,y} \).
- Then \( W_x \leftarrow W_x \setminus \{t\} \);
- Deleted \( \leftarrow \) true;
- Return Deleted.

In Step 2 of the sub-procedure, each value \( t \) in the current domain \( W_y \) of \( x \) is scanned. If for a specific \( t \), there does not exist a value \( t' \) in the current domain \( W_y \) of \( y \), such that the assignment \( (x \leftarrow t, y \leftarrow t') \) is feasible for \( C_{x,y} \), the value \( t \) is removed from \( W_x \). If the maximum number of elements in the domains is \( M \), it is straightforward to show that AC-1 has a time complexity of \( O(MP \cdot |S| \cdot |C|) \). Several improved AC algorithms are available in the CSP literature [41]. In our implementation, the AC-5 arc-consistency algorithm introduced in [42] is used. AC-5 improves over earlier versions of AC algorithm.
by working with a queue that iteratively removes a constraint/value pair and performs consistency tests, and is significantly more efficient [43].

4.2. Branching rules

An important component in a CP algorithm is its efficiency to search the solution space. We implement two heuristic branching rules to control the order for selecting variables/values during the CP search.

4.2.1. First-fail rule

The first-fail rule selects the most “difficult” variable first, i.e. the one with the smallest cardinality of domain [41]. It is generally believed to result in a smaller search tree, as some variables can be bounded early during the search. As a generic search strategy, however, the first-fail rule does not explore the features of the safety stock placement problem.

4.2.2. Maximal-regret rule

We also implement a branching rule based on the idea of maximal-regret principle, which first chooses the variable most promising in improving solution quality. The term regret refers to the difference between what would have been the best decision in a scenario and what was the actual decision. The key of such a branching rule is that it allows one to exploit the structure of the addressed problem, which may lead to finding quality solutions early during the search. It has been shown that finding good feasible solutions early will often reduce the size of CP search tree [41]. The design of maximal-regret rule relies on the following properties due to the scheduling characteristics of the safety stock placement problem.

Property 1. Given a fixed schedule for dummy activities, the objective function is regular (non-decreasing) w.r.t. the starting times of real activities.

Proof. Let $\tau_j$ represent the net replenishment time of supply chain function $j \in N$, i.e. $\tau_j = d(j) \cdot start + T_j - b(j) \cdot start$. Since $\Phi_j(\cdot)$ is a non-decreasing function of $\tau_j$, the objective function $f \cdot \cdot$ is also non-decreasing. Thus $\tau_j \leq \tau'_j$ implies that $f(\tau_j) \leq f(\tau'_j)$. Since both $T_j$ and $b(j) \cdot start$ are fixed, $f$ is non-decreasing in $d(j) \cdot start$. Q.E.D.

Property 2. Given a fixed schedule for real activities, the objective function is non-regular (non-increasing) w.r.t. the starting times of dummy activities.

Proof. Similar to the proof of Property 1. Q.E.D.

Property 1 and Property 2 imply that an early start time for a real activity, and a late start time for a dummy activity are likely to produce quality solutions. Such insights lead to the following composite maximal-regret branching rule:

i) For real activities, we first choose $d(j)$, for which the difference between the earliest possible start time and the next earliest start time is maximal. Then assign the earliest possible start time to $d(j) \cdot start$.

ii) For dummy activities, we first choose $b(j)$, for which the difference between the latest possible start time and the next latest start time is maximal. Then assign the latest possible start time to $b(j) \cdot start$.

5. Hybrid CP–GA algorithm

Our computational experiences indicate that CP is quite effective for small safety stock placement instances. When the problem size increases, it becomes expensive for CP alone to search the solution space. This has motivated us to design hybrid algorithms to enhance the quality of CP solutions. Our current research develops a scheme to integrate CP with genetic algorithm (GA). GA is a random search technique that mimics processes observed in natural evolution [44]. It combines survival of the fittest (or best) among solutions with a structured yet randomized exchange (cf. [45,46]).

In our hybrid CP–GA algorithm, GA is used as a metaheuristic strategy to improve the CP solutions; on the other hand, CP provides high-quality solutions, which can construct the initial population or serve as elite individuals in the initial population for GA. In addition, CP also provides tighter bounds on the decisions variables, i.e. the inbound and outbound service times, which can significantly reduce the GA search space. The hybrid CP–GA framework is described next, followed by details about the GA components.

5.1. Hybrid CP–GA framework

Our proposed hybrid CP–GA framework is depicted in Fig. 4. The initial population is composed of CP solutions and also by a set of randomly generated feasible solutions. CP solutions are obtained by running the CP algorithm described in Section 4 with certain limit on the CP search time. We include the best CP solution together with some intermediate feasible solutions found during the CP search. Randomly generated feasible solutions are also included to introduce diversity to the initial population. Standard penalty function is employed to penalize infeasible solutions. In addition to the initial solutions, CP also provides tight bounds of the decision variables, which can significantly reduce the solution space for GA.

Reproduction, crossover and mutation are implemented in a parallel fashion to generate offspring. Crossover enables the algorithm to extract the best genes from different individuals and recombine them into potentially superior children. The crossover operator is applied to each selected pair of parent chromosomes with probability $p_{crossover}$. Mutation adds diversity to the population, and thereby increases the likelihood of generating individuals with better fitness values. The mutation operator is applied to each individual with probability $p_{mutation}$. Thus the entire new generation consists of $np_{crossover}$ individuals from crossover, $np_{mutation}$ individuals from mutation, and the elite individuals from reproduction. We set $n_{max}$ as the maximum number of generations to run. That is, the GA terminates if the number of generations exceeds $n_{max}$.

5.2. Encoding scheme

Each chromosome is encoded as a vector of positive integer values, which directly represents the decision variables $S_i$ and $S_j$. Fig. 5 illustrates such encoding scheme, with each chromosome having a length of $2n$, where $n$ is the number of nodes in the supply chain network.

5.3. Genetic operators

5.3.1. Parent selection

Parent selection is critical to guide GA toward promising regions in the solution space. In our implementation, we employ the roulette wheel selection method [47] to keep diversity of chromosomes.
5.3.2. Crossover operator
The crossover operator generates new children by combining information contained in the chromosomes of the parents so that new chromosomes will have the best parts of the parents’ chromosomes. As the solution space is large, it is crucial to avoid spending excessively search effort evaluating infeasible solutions. This has motivated us to consider the arithmetic operator as described in [48] (p. 246), which is a linear (convex) combination of two feasible solutions. When a problem involves only linear constraints, the solution space is convex, so that an arithmetic operator always yields a feasible solution in the convex search space. It has proved to be effective for many linearly constrained problems [49].

Although our safety stock placement problem involves only linear constraints (2)–(5), the solution space is not convex due to the existence of integer constraint (6), thus the elementary arithmetic operator is not immediately applicable. We employ a modified arithmetic crossover operator to create offspring \( x_0 \) from its parents \( x_1 \) and \( x_2 \) in the following way:

\[
x_0 = \lambda x_1 + (1 - \lambda) x_2,
\]

where \( \lambda \) is a random real number in \([0, 1]\), \( x' \) is rounded down into nearest integer through the floor function \( \lfloor \cdot \rfloor \). If \( x_1 \) and \( x_2 \) are feasible to the constraint system (2)–(6), \( x' \) is also feasible. A detailed proof of this property is provided in the Appendix.

This modified arithmetic crossover operator enables us to reduce the number of infeasible solutions to be evaluated, enhancing the efficiency and effectiveness of GA significantly.

5.3.3. Mutation operator
The role of mutation is to provide a small amount of randomness to explore new solution space and prevent premature convergence. In our GA, the mutation operator first randomly selects one gene of decision variables on a chromosome, and then replaces the selected gene value with a random integer within the solution bound. In order for the child through mutation to be feasible, the following steps are implemented.

Step 1. Select a parent and let \( \text{child} = \text{parent} \).
Step 2. Set the mutation position: \( \text{Position} = \lambda_1 \times 2n \), where \( \lambda_1 \) is a real random number in \([0, 1]\), and \( 2n \) is the length of chromosome.
Step 3. Set the value of \( \text{Position} \) by following formula:

\[
\text{Child}(\text{Position}) = \text{LB}(\text{Position}) + \lambda_2 (\text{UB}(\text{Position}) - \text{LB}(\text{Position})),
\]

where \( \lambda_1 \) is a real random number in \([0, 1]\), \( \text{LB}(\cdot) \) and \( \text{UB}(\cdot) \) refer to the lower bound and upper bound, respectively, of the value of \( \text{Position} \) obtained by CP.

Fig. 4. Sketch of the hybrid CP–GA framework.

Fig. 5. Representation of a chromosome.
Step 4. Check if child is feasible. If so, terminate; otherwise, let child = parent and go to step 2.

To avoid excessive computational effort to obtain a feasible child, the procedure is terminated when reaching a certain number of iterations.

6. Computational experiments

The CP model and algorithms were implemented in C++ using constraint programming libraries ILOG CP Optimizer 2.3 [40], which support the implementation of tree search algorithms that apply constraint propagations at the nodes of the tree [50]. The important features of these CP libraries include the following: (1) embedded constraint propagation algorithm, which is a variant of the AC-5 arc consistency algorithm [42]; (2) support for standard backtracking during CP search; (3) C++ classes for defining and implementing CP branching schemes. The GA code was implemented in MATLAB using the Genetic Algorithm Toolbox [51].

We generate the problem instances in the following way:

The underlying general acyclic network is generated using the project scheduling problem generator called ProGen/Max [52]. We generate networks with 10, 20, 40 and 80 nodes. For each network size, 15 problem instances are randomly generated.

We use the standard functional form \( \Phi_j(x) = \sqrt{x} \) as in the literature (cf. [1,2,9,11]), where \( X \) is the net replenishment time. When demand follows the assumed normal distribution \( N(\mu, \sigma^2) \), the safety stock required to achieve certain service level is proportional to \( \sigma \sqrt{X} \).

- Following [1] and [2], we generate lead time and capacity from uniform distribution. Specifically, lead time of each supply chain function is generated randomly from \( U[50,70] \), and capacity from \( U[2,60] \).
- There appear to be three ways to generate \( h_j \) in the literature: (i) it can be independently generated from a uniform distribution as in [2]; (ii) in the main computational experiments of [1], the holding cost rate \( h_j \) of a downstream function \( k \) is obtained by adding a random number \( U \) to the largest holding cost rate \( h_j \) of its immediate upstream function \( j \), i.e., \( h_k = h_j + U \). This is to reflect the fact that the downstream functions are more costly due to the value-adding process; (iii) a more realistic way to model such value-adding process, as in [9], is to calculate \( h_k \) as the sum of holding cost rate \( h_j \) of all its upstream functions, i.e.

\[
h_k = \sum_{j \in A} h_j
\]  

(16)

As shown in [1], instances generated through (16) can have a slower convergence rate and lead to a larger LP gap. We use (16) in our experiments to generate \( h_j \).

As CP is often able to reach high-quality solutions fast, a time limit of 60 s is imposed for the pure CP algorithm: both the Default CP with the first-fail search strategy, and the Max-Reg CP with the maximal-regret search strategy. The CP search in the hybrid CP–GA algorithm is limited to 5 s. Of course, there is a trade-off between the initial CP search effort and solution quality. Our computational experience has shown that 5 s is usually sufficient for CP to generate high-quality initial population for GA. In order to evaluate the solution quality of our algorithms, we solve all problem instances to optimality through the exact piecewise linear MIP method proposed by [1]. We also compare our algorithms with the iterative LP heuristic by [2]. Both the exact MIP and Iterative LP methods are implemented with the same stopping criteria as in [1] and [2] using the latest version of CPLEX 12.1 [53]. All computations were performed on a Pentium IV PC with 3.2 GHz CPU and 1 G RAM.

6.1. Tuning GA Parameters

Our computational experience indicates that, setting the total number of generations \( N_{\text{max}} \) to be 1000 and the population size to be 300 achieves a good balance between solution quality and computational efficiency.

The default parallel reproduction operation in the Matlab GA Toolbox is employed for the same pool of population in each generation, such that \( p_{\text{cross}} + p_{\text{mut}} + \#\text{Elite}/300 = 1 \) [51], where \#Elite stands for the number of elite solutions directly kept to the next generation. In our implementation, we always keep the current best solution, i.e., we set \#Elite = 1. Fig. 6 shows the solution quality of pure GA for five selected instances with different value of \( p_{\text{cross}} \) ranging from 0.1 to 0.9 (with an increment of 0.1). It appears that a crossover rate of 0.3 leads to better solution quality of pure GA.

Fig. 7 illustrates the search process of CP–GA on the 11th 80-node instance. The evolution process indicates that it improves the CP initial solutions effectively and quickly. Fig. 8 depicts the...
average distance between individuals. It shows that our crossover and mutation operators are quite effective in keeping the population diversified during the search.

6.2. Computational results

Deviations of solutions found by our algorithm to optimal solution are reported as the following gap: \(\frac{\text{Obj} - \text{Obj}_{\text{MIP}}}{\text{Obj}_{\text{MIP}}}\), where \(\text{Obj}_{\text{MIP}}\) denotes optimal objective value obtained by the exact MIP method. We also report the CPU time (in seconds) for finding best solutions. Due to the probabilistic nature of GA, the pure GA and hybrid CP–GA were run five times for each instance. Both the best and average solution quality (in the format of \([\text{best gap}, \text{average gap}]\)) and CPU time are reported. Complete computational results are provided in Tables 1–4.

The pure CP algorithm itself performs well for the small 10-node instances. Both the Default CP (using the first-fail rule) and the Max-Reg CP (using the maximal-regret rule) find optimal solutions to all fifteen instances. Max-Reg CP is more efficient than Default CP using only fractions of a second on average. The pure GA finds quality solutions within 1% gap, but spends significantly more computational time. The average gap of the Iterative LP method is 1.22%.

For the fifteen 20-node instances, Max-Reg CP has an average gap of 1.70%, which clearly outperforms the Default CP in both solution quality and computational time. Such better performance is achieved by the maximal-regret rule for the CP tree search, which often finds good quality solutions early during the search. The hybrid CP–GA

---

**Table 1**

Computational results for the 10-node problem instances.

<table>
<thead>
<tr>
<th>Instance #</th>
<th>No. Arcs</th>
<th>Iterative LP</th>
<th>Default CP</th>
<th>Max-Reg CP</th>
<th>Pure GA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Gap % CPU # Iter</td>
<td>Gap % CPU</td>
<td>Gap % CPU</td>
<td>Gap % CPU</td>
</tr>
<tr>
<td>1</td>
<td>17</td>
<td>0.12 &lt; 0.01 3</td>
<td>0</td>
<td>40.55 0</td>
<td>0.28 [0.12, 0.62] 22.61</td>
</tr>
<tr>
<td>2</td>
<td>17</td>
<td>1.76 &lt; 0.01 2</td>
<td>0</td>
<td>0.14 0</td>
<td>0.31 [1.30, 2.31] 16.33</td>
</tr>
<tr>
<td>3</td>
<td>17</td>
<td>4.05 &lt; 0.01 2</td>
<td>0</td>
<td>0.13 0</td>
<td>0.13 [0.00, 0.82] 40.66</td>
</tr>
<tr>
<td>4</td>
<td>19</td>
<td>1.87 &lt; 0.01 2</td>
<td>0</td>
<td>0.06 0</td>
<td>0.05 [0.00, 2.67] 10.64</td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>0.00 &lt; 0.01 2</td>
<td>0</td>
<td>0.02 0</td>
<td>&lt; 0.01 [0.00, 0.42] 6.203</td>
</tr>
<tr>
<td>6</td>
<td>25</td>
<td>0.00 &lt; 0.01 2</td>
<td>0</td>
<td>0.19 0</td>
<td>&lt; 0.01 [0.00, 0.16] 11.66</td>
</tr>
<tr>
<td>7</td>
<td>21</td>
<td>1.02 &lt; 0.01 2</td>
<td>0</td>
<td>0.20 0</td>
<td>1.77 [1.49, 1.76] 25.95</td>
</tr>
<tr>
<td>8</td>
<td>15</td>
<td>2.05 &lt; 0.01 2</td>
<td>0</td>
<td>0.00 0</td>
<td>0.03 [0.00, 0.40] 8.438</td>
</tr>
<tr>
<td>9</td>
<td>25</td>
<td>0.00 &lt; 0.01 2</td>
<td>0</td>
<td>0.05 0</td>
<td>0.34 [1.07, 2.95] 50.47</td>
</tr>
<tr>
<td>10</td>
<td>15</td>
<td>0.00 &lt; 0.01 2</td>
<td>0</td>
<td>0.02 0</td>
<td>0.00 [0.00, 0.42] 11.64</td>
</tr>
<tr>
<td>11</td>
<td>25</td>
<td>0.00 &lt; 0.01 2</td>
<td>0</td>
<td>7.38 0</td>
<td>0.45 [0.00, 0.05] 67.2</td>
</tr>
<tr>
<td>12</td>
<td>17</td>
<td>1.55 &lt; 0.01 2</td>
<td>0</td>
<td>0.02 0</td>
<td>0.14 [0.66, 1.34] 25.86</td>
</tr>
<tr>
<td>13</td>
<td>15</td>
<td>5.83 &lt; 0.01 2</td>
<td>0</td>
<td>0.19 0</td>
<td>&lt; 0.01 [0.00, 0.23] 25.02</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
<td>0.00 &lt; 0.01 2</td>
<td>0</td>
<td>0.06 0</td>
<td>0.06 [0.24, 0.83] 19.67</td>
</tr>
<tr>
<td>Average</td>
<td>19</td>
<td>1.22 &lt; 0.01 2</td>
<td>0</td>
<td>3.26 0</td>
<td>0.24 [0.33, 1.06] 24.03</td>
</tr>
</tbody>
</table>

---

**Table 2**

Computational results for the 20-node problem instances.

<table>
<thead>
<tr>
<th>Instance #</th>
<th>No. Arcs</th>
<th>Iterative LP</th>
<th>Default CP</th>
<th>Max-Reg CP</th>
<th>Pure GA</th>
<th>CP–GA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Gap % CPU # Iter</td>
<td>Gap % CPU</td>
<td>Gap % CPU</td>
<td>Gap % CPU</td>
<td>Gap % CPU</td>
</tr>
<tr>
<td>1</td>
<td>32</td>
<td>5.83 0.02 3</td>
<td>0.00 3.08</td>
<td>0.77 0.56</td>
<td>[0.00, 0.10] 7.34</td>
<td>[0.00, 0.00] 2.22</td>
</tr>
<tr>
<td>2</td>
<td>35</td>
<td>2.80 0.05 5</td>
<td>6.17 41.27</td>
<td>1.41 0.11</td>
<td>[1.12, 1.29] 47.72</td>
<td>[1.04, 1.04] 6.00</td>
</tr>
<tr>
<td>3</td>
<td>36</td>
<td>1.41 0.02 3</td>
<td>0.03 0.45</td>
<td>0.00 0.05</td>
<td>[0.00, 0.00] 2.00</td>
<td>[0.00, 0.00] &lt; 0.01</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td>0.00 0.06 5</td>
<td>3.47 0.75</td>
<td>1.46 19.09</td>
<td>[1.33, 1.41] 9.85</td>
<td>[1.33, 1.46] 20.72</td>
</tr>
<tr>
<td>5</td>
<td>27</td>
<td>3.13 0.11 4</td>
<td>9.20 0.33</td>
<td>0.31 15.42</td>
<td>[0.11, 0.20] 53.64</td>
<td>[0.11, 0.11] &lt; 0.01</td>
</tr>
<tr>
<td>6</td>
<td>50</td>
<td>0.21 0.11 3</td>
<td>0.94 43.14</td>
<td>1.58 0.13</td>
<td>[0.00, 0.01] 19.85</td>
<td>[0.00, 0.06] 1.72</td>
</tr>
<tr>
<td>7</td>
<td>30</td>
<td>0.00 0.03 3</td>
<td>8.00 3.16</td>
<td>2.33 11.06</td>
<td>[0.44, 1.22] 45.22</td>
<td>[0.44, 0.94] 10.56</td>
</tr>
<tr>
<td>8</td>
<td>39</td>
<td>3.01 0.06 3</td>
<td>0.13 1.47</td>
<td>0.03 2.53</td>
<td>[0.03, 0.03] 2.00</td>
<td>[0.03, 0.03] &lt; 0.01</td>
</tr>
<tr>
<td>9</td>
<td>24</td>
<td>0.03 0.02 3</td>
<td>10.34 0.97</td>
<td>9.24 58.16</td>
<td>[5.68, 6.61] 49.39</td>
<td>[1.37, 3.13] 45.84</td>
</tr>
<tr>
<td>10</td>
<td>33</td>
<td>10.59 0.08 4</td>
<td>4.28 &lt; 0.01 1.47 1.42</td>
<td>[1.46, 1.46] 2.00</td>
<td>[1.46, 1.46] 6.55</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>28</td>
<td>1.16 0.14 3</td>
<td>2.90 0.69</td>
<td>0.12 12.66</td>
<td>[0.00, 0.03] 7.45</td>
<td>[0.00, 0.11] 4.80</td>
</tr>
<tr>
<td>12</td>
<td>35</td>
<td>2.24 0.02 3</td>
<td>3.17 0.67</td>
<td>4.15 &lt; 0.01</td>
<td>[3.87, 4.03] 56.16</td>
<td>[3.86, 3.92] 3.86</td>
</tr>
<tr>
<td>13</td>
<td>44</td>
<td>3.17 0.11 3</td>
<td>0.00 0.22</td>
<td>1.22 0.05</td>
<td>[0.00, 0.31] 37.8</td>
<td>[0.00, 0.82] 3.44</td>
</tr>
<tr>
<td>14</td>
<td>43</td>
<td>3.55 0.03 2</td>
<td>0.00 36.72</td>
<td>0.33 0.94</td>
<td>[0.00, 0.13] 22.28</td>
<td>[0.00, 0.00] &lt; 0.01</td>
</tr>
<tr>
<td>15</td>
<td>40</td>
<td>8.40 0.08 3</td>
<td>0.01 0.27</td>
<td>1.11 0.00</td>
<td>[0.00, 0.22] 20.03</td>
<td>[0.01, 0.01] 2.02</td>
</tr>
<tr>
<td>Average</td>
<td>35</td>
<td>2.99 0.06 3</td>
<td>3.24 8.88</td>
<td>1.70 8.15</td>
<td>[0.94, 1.14] 24.92</td>
<td>[0.64, 0.88] 6.78</td>
</tr>
</tbody>
</table>

algorithm is able to further improve the CP solution quality to 0.64% on average. Notably, CP–GA finds optimal solutions to four instances for which the pure CP did not find optimal solutions. It also significantly outperforms the pure GA, finding better quality solutions using less than a quarter of GA's computational time. The Iterative LP appears to be the most efficient one, spending only 0.06 s on average. Notably, CP–GA finds optimal solutions to four instances using less than a quarter of GA's computational time. The pure CP did not find optimal solutions. It also finds optimal solutions to all the ten instances.

For the fifteen 40-node instances, it takes the exact MIP method 1800 s (8 iterations) on average to reach optimal solutions. Max-Reg CP has an average gap of 3.11%, which is significantly better than Default CP. CP–GA further improves Max-Reg CP solution quality to less than 1%. It also clearly outperforms the pure GA. The quality of Iterative LP is better than Default CP and pure GA, comparable to Max-Reg CP, but not as good as CP–GA. The Iterative LP again takes the least amount of time: only 0.13 s on average; the best performed CP–GA spends about 40 s on average.

To directly compare with the heuristic methods in [2], we run the Max-Reg CP algorithm, the Iterative LP and the 2-piece iterative MIP (2P Iterative MIP) algorithms on the 10-node 25-arc dense network instance taken directly from [2]. The comparison results are reported in Table 5. It is observed that the quality of the Iterative LP varies significantly with the delivery deadline of end products: its optimality gap increases from around 1% to 9% as the deadline increases from 0 to 10. The 2P Iterative MIP method performs better, whose gap also tends to increase as the deadline increases. The Max-Reg CP appears to be quite robust with respect to the change in deadline: it finds optimal solutions to all the ten instances.

1 In our implementation of the 2-piece iterative MIP method of Shu and Karimi [2], we corrected a typo in their original paper: the inequality at the bottom of Page 2901 should be \(X_k \leq y \) instead of \( y \leq X_k \). In both methods, the same configuration is used as that in [2]. We further replace the stopping criterion in [2] with a more stringent one: \( 10^{-3} \). The same results were obtained. Additional tuning of algorithm parameters goes beyond the scope of this paper.
Despite its efficiency for solving large problems, one potential difficulty of the iterative heuristic methods is the existence of error bounds. We refer to [54] for empirical analysis of the error bounds incurred by linear approximation, and [55] for theoretical analysis on a priori bound for functional approximation in math programming. For the safety stock problem in generalized networks, the impact of such error bounds is evident in the Table 3 (p. 2903) of [2]: it prevents the iterative LP or the 2P iterative MIP method from finding better solutions for even small size networks (with 10 to 20 nodes) compared to large problems. It is also evident in Table 1 and Table 2 of this paper: the iterative LP method fails to find optimal solution to any of the fifteen 10-node problems, which have all been optimally solved by Max-Reg CP; for the fifteen 20-node problems, the iterative LP yields an average gap of about 3%, whereas Max-Reg CP alone has a gap of 1.7% on average. It has been empirically shown in [1], as well as in this paper, that $h_i$ generated through (16) can lead to a larger error bounds. The results in Table 5 further suggest that the error bounds of both iterative LP and 2P iterative MIP tend to increase as the problem becomes less restrictive with a longer delivery deadline.

### 7. Conclusions and future research

We propose a new modeling framework for the safety stock placement problem in general acyclic supply chain networks. Our approach transforms a general acyclic supply chain network into a project scheduling network. With such transformation, it becomes natural to model dependencies among the inbound and outbound service times, which typically becomes less restrictive with a longer delivery deadline.

The idea of transforming a general acyclic supply chain network into a project scheduling network opens a number of opportunities for modeling and solving the safety stock placement problem. From the modeling perspective, such transformation makes it natural to model various renewable resources ubiquitous in complex manufacturing systems and supply chain networks in general. This provides a feasible and promising way to study safety stock placement in a resource-constrained environment. From the algorithmic perspective, it will be interesting to adapt other scheduling methods and/or local search based meta-heuristics for the safety stock placement problem.

### Acknowledgments

Dr. Jiang’s research was supported by the National Natural Science Foundation of China (Grant # 70871119/0109), the National Social Science Foundation of China (Grant # 07XJY026), and the Science Innovation Foundation of Chongqing Logistics Engineering University. The authors also thank three anonymous referees for their comments that helped improving contents and presentation of this paper.

### Appendix

#### Definition 3

A floor function $x$ is defined as the greatest integer that does not exceed a real number $x$.

#### Definition 4

Let $0 \leq (x) < 1$ denote the fractional part of $x$, such that $x = x + (x)$.

#### Lemma 1

If $x_1$ and $x_2$ satisfy the constraint $x \leq b$, and $x'$ is a convex combination of $x_1$ and $x_2$, then $x' \leq b$ holds.

**Proof.** Since $x'$ is a convex combination of $x_1$ and $x_2$, we have $x' \leq b$. Therefore, $x' \leq x \leq b$. Q.E.D.

#### Lemma 2

If $(x_1, y_1)$ and $(x_2, y_2)$ satisfy the constraint $x - y \leq b$, where $b$ is an integer. Let $(x', y')$ be a convex combination of $(x_1, y_1)$ and $(x_2, y_2)$, then $x' - y' \leq b$ holds.

---

Proof. Since \((x', y')\) is a convex combination of two feasible solutions, it satisfies the constraint, i.e. \(x' - y' \leq b\). If suffices to show that \(x' - y' \leq x - y\). There are the following two possibilities:

(i) If \(y' \leq y\), then \(y' - x' \leq x - y\), thus \(x' - y' \leq x - y\).

(ii) Consider now \(y' > y\). Since \(x' + y' - y \leq x + y\), then \(x' - y \leq x + y - y = x\). Both \(x'\) and \(y'\) are in the range \([0,1]\), thus \(0 < x' - y' < 1\). As \(\cdot + \cdot\) is a monotonic function, \(x' - y' < b + y - y = b\). That is, \(x' - y' \leq b\), Q.E.D.

Property 3. Let \(x_i = (x_{11}, x_{12}, \ldots, x_{1n})\) and \(x_2 = (x_{21}, x_{22}, \ldots, x_{2n})\) be two vector of feasible solutions to the constraint system \((2)-(6)\). If \(x' = (x_{11}, x_{12}, \ldots, x_{1n})\) is a convex combination of \(x_1\) and \(x_2\), then \(x' = (x_{21}, x_{22}, \ldots, x_{2n})\) is feasible to the constraint system \((2)-(6)\).

Proof. By definition, \(x_i\) is a convex combination of \(x_{1i}\) and \(x_{2i}\), for \(i = 1,2,\ldots, n\). Any constraint in the constraint system \((2)-(6)\) takes either the form \(x \leq b\) or the form \(x < b\), where \(b \in \mathbb{R}\) is an integer.

Consider any constraint of the form \(x \leq b\). Without loss of generality, assume it involves the \(i\)th decision variable in solution vector. Since \(x_{1i}\) and \(x_{2i}\) are two feasible solutions to \(x \leq b\), and \(x_i\) is a convex combination of \(x_{1i}\) and \(x_{2i}\), it follows from Lemma 1 that \(x_i \leq b\), i.e. \(x_i \leq b\).

Consider any constraint of the form \(x < b\). Without loss of generality, assume it involves the \(i\)th and \(j\)th decision variables in the solution vector. Since \((x_{1i}, x_{1j})\) and \((x_{2i}, x_{2j})\) are two feasible solutions to \(x < b\), and \((x_i, x_j)\) is convex combination of \((x_{1i}, x_{1j})\) and \((x_{2i}, x_{2j})\), it follows from Lemma 2 that \(x_i - x_j \leq b\), i.e. \((x_i, x_j)\) is feasible to the constraint system \(x < b\).

Therefore, the solution vector \(x' = (x_{11}, x_{12}, \ldots, x_{1n})\) is feasible to the constraint system \((2)-(6)\), Q.E.D.

References


