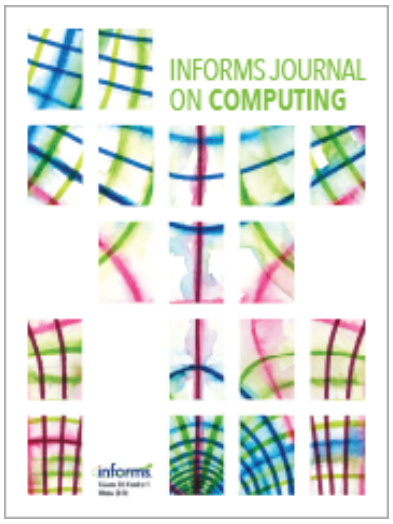


INFORMS Journal on Computing

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

Multilevel Lot-Sizing with Inventory Bounds

Hark-Chin Hwang, Wilco van den Heuvel, Albert P. M. Wagelmans

To cite this article:
Hark-Chin Hwang, Wilco van den Heuvel, Albert P. M. Wagelmans (2023) Multilevel Lot-Sizing with Inventory Bounds. INFORMS Journal on Computing 35(6):1470-1490. https://doi.org/10.1287/ijoc.2022.0216

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

INFORMS JOURNAL ON COMPUTING
Vol. 35, No. 6, November–December 2023, pp. 1470–1490
ISSN 1091-9856 (print), ISSN 1526-5528 (online)

Multilevel Lot-Sizing with Inventory Bounds

Hark-Chin Hwang,[a] Wilco van den Heuvel,[b,*] Albert P. M. Wagelmans[b]

[a] School of Management, Kyung Hee University, Seoul 130-701, Republic of Korea; [b] Econometric Institute, Erasmus University Rotterdam, 3062 Rotterdam, Netherlands

*Corresponding author

Contact: hchwang@khu.ac.kr, https://orcid.org/0000-0002-1181-0514 (H-CH); wvandenheuvel@ese.eur.nl, https://orcid.org/0000-0002-6633-5941 (WvdH); wagelmans@ese.eur.nl, https://orcid.org/0000-0003-4803-8262 (APMW)

Received: July 22, 2022
Revised: April 21, 2023; June 30, 2023
Accepted: June 30, 2023
Published Online in Articles in Advance: August 17, 2023

https://doi.org/10.1287/ijoc.2022.0216

Abstract. We consider a single-item multilevel lot-sizing problem with a serial structure where one of the levels has an inventory capacity (the bottleneck level). We propose a novel dynamic programming algorithm combining Zangwill's approach for the uncapacitated problem and the basis-path approach for the production capacitated problem. Under reasonable assumptions on the cost parameters the time complexity of the algorithm is $\mathcal{O}(LT^6)$ with L the number of levels in the supply chain and T the length of the planning horizon. Computational tests show that our algorithm is significantly faster than the commercial solver CPLEX applied to a standard formulation and can solve reasonably sized instances up to 48 periods and 12 levels in a few minutes.

History: Accepted by Andrea Lodi, Area Editor for Design & Analysis of Algorithms–Discrete.
Funding: This work was supported by the National Research Foundation of Korea (NRF) funded by the Ministry of Education [Grant 2014R1A1A2058513].

Keywords: inventory bound • multilevel lot-sizing • dynamic programming • inventory-production: deterministic

1. Introduction

One of the important tasks in supply chain management is to have a smooth flow of goods from the raw material supplier to the end customer. Anything that prevents this smooth flow of items through the different levels of the supply chain can be considered a bottleneck. The supply chain problem we consider has a serial structure and a bottleneck in terms of inventory capacity. To be precise, there is a single level (organization or process) in the supply chain that has a time-invariant inventory capacity. This will be called the *bottleneck level*.

Inventory capacities can be observed in a variety of industries for several reasons. In logistics, where the length of the supply chain may be long, the storage facilities affect the flow of items in the supply chain. In particular, the warehouse capacity at certain levels restricts the number of units to be held in stock and there may be a single level with the lowest warehouse storage capacity causing the bottleneck. This is in particular the case for bulk products that cannot be mixed in the same tank or silo. Moreover, the investment in storage capacity can be significant, and thus may be constraining in the supply chain. When there are multiple stages in the supply chain, not all of them include bottleneck storage capacity. This is because, as a product is transformed, tanks or silos are more costly. This is for instance the case for the products considered in Dauzère-Pérès et al. (2007), where the raw materials have no real storage capacity constraints, whereas the slurry products, obtained through a manufacturing process, require dedicated expensive silos, and thus in limited number.

Although there are several settings in which an inventory capacity arises, it turns out that there are hardly any studies in this area. In fact, we have found no study in which an exact approach is developed to solve this type of problem in a multilevel lot-sizing setting for a general number of levels. In this paper, we develop a polynomial time algorithm to solve a serial multilevel lot-sizing problem with an inventory bound under certain cost assumptions. In particular, we assume that there are nonspeculative motives for holding inventory, and that setup costs are nonincreasing at the bottleneck level. Our approach can be viewed as a novel combination of Zangwill's approach for the uncapacitated problem (Zangwill 1969) and the basis-path approach for the production capacitated problem (Hwang et al. 2013). Furthermore, an algorithmic speed-up is obtained by deriving and exploiting a structural property of an optimal solution. Although the complexity of the algorithm is high (but polynomial), we show in our numerical experiments that the algorithm is much faster than the commercial solver CPLEX applied to a standard formulation, solving instances up to 48 periods and 12 levels in a few minutes.

The remainder of the paper is organized as follows. In Section 2, we briefly review the literature on capacitated lot-sizing problems. We formally define the multilevel lot-sizing problem and characterize some optimality

properties in Section 3. The dynamic programming approach is developed in Section 4. We perform a computational study to analyze the scalability of the algorithm in Section 5 and conclude the paper in Section 6.

2. Literature Review

As the problem considered in this paper involves a single item, we restrict the literature review to papers about single-item lot-sizing problems. The study of lot-sizing problems started with the seminal paper of Wagner and Whitin (1958), who propose a dynamic programming algorithm to make optimal production decisions to meet a deterministic multiperiod demand stream when total setup and inventory-holding costs are to be minimized. Computational improvements on solving this uncapacitated lot-sizing problem are made by Federgruen and Tzur (1991), Wagelmans et al. (1992), and Aggarwal and Park (1993). Zangwill (1969) extends the single-level problem to a multilevel setting and presents an algorithm for the multilevel uncapacitated lot-sizing problem. Love (1972) proposes a more efficient algorithm for this problem in case of nonincreasing setup costs, whereas Melo and Wolsey (2010) present a more efficient algorithm for the two-level version of the problem.

The problem with a bottleneck in terms of production capacity has received quite some attention in the lot-sizing literature in case of a single level. This problem is first studied by Florian and Klein (1971). Algorithmic improvements are made by Chung and Lin (1988), van Hoesel and Wagelmans (1996), and Van Vyve (2007). Love (1973) is the first to introduce a lot-sizing problem with a bottleneck in terms of storage capacity. This problem can be shown to be equivalent to the lot-sizing problem with (noninclusive) production time windows (Wolsey 2006) and to the problem with pure remanufacturing or cumulative capacities (van den Heuvel and Wagelmans 2008). Hwang and van den Heuvel (2012) present more efficient algorithms for this single-level inventory bounded problem.

There are a limited number of papers that consider capacities in a multilevel setting. Kaminsky and Simchi-Levi (2003) deal with a two-level problem with production capacities at the first level, whereas Sargut and Romeijn (2007) solve the same problem with a subcontracting option. van Hoesel et al. (2005) extends the two-level problem to a multilevel problem with production capacities at the first level and solve the problem in case of nonspeculative motives in transportation in polynomial time, whereas Hwang et al. (2013) are able to do this for the same multilevel problem efficiently without this assumption on the costs. Lee et al. (2003) solve a two-level problem and deal with the transportation capacity (of vehicles) at the second level. Finally, Phouratsamay et al. (2018) propose an efficient algorithm for the two-level problem with inventory bounds at the second level.

The papers closest to our research are He et al. (2015), Ahmed et al. (2016), and Zhao and Zhang (2020). They consider multilevel lot-sizing problems by viewing them as minimum concave cost network flow problems over a grid network. He et al. (2015) show that this network flow problem is polynomially solvable for a fixed number of levels L in case (i) all sources are at the first level and all sinks (demands) occur at most two levels, or (ii) there is a single source but many sinks at multiple levels. These results are extended by Ahmed et al. (2016) who derive polynomial time results under more general conditions. They also show that their analysis is tight by proving NP-hardness if any of the conditions is relaxed. Zhao and Zhang (2020) consider a multilevel lot-sizing problem with the possibility of intermediate demands. The uncapacitated version is shown to be NP-hard. However, under the assumption of a fixed number of levels, polynomial time algorithms are developed for the uncapacitated problem and for the problem with production capacities at the first level. With the proposed algorithm, they also improve several complexity results from the literature. Furthermore, they derive new valid inequalities for the multilevel lot-sizing problem. When considering the time complexities of these papers in more detail, the degrees of the polynomials depend (linearly) on the number of levels and hence, although polynomial for a fixed number of levels, the algorithms become quickly impractical for a reasonable number of levels. We can argue that our research complements the three papers mentioned above in the sense that our polynomial time result is more general as it holds for a general number of levels (opposed to a fixed number of levels), but on the other hand, it is less general because we have some (reasonable) assumptions on the cost structure.

Table 1 provides a summary of the complexity results in connection with our study. The first part of the table shows complexity results for lot-sizing problems with inventory capacities (ILSP), whereas the second part does this for lot-sizing problems with production capacities (CLSP). The first character in the problem abbreviation shows the number of levels where M (multi) represents a general number of levels L. In the last part of the table, we show more general results. As mentioned earlier, Ahmed et al. (2016) consider a more general problem where the number of capacity levels is at most K. Applying this to the problem of this paper (so $K = 1$) gives a runtime complexity of $O(L^{4L-3}T^{8L-7})$. Although this is polynomial for a fixed number of levels, the running time is already impractical for a low number of levels such as $L = 3$. In turn, this also shows the relevance of our paper, where we solve instances up to 12 levels. Finally, the last complexity result shows that in case the multilevel problem has an assembly structure and is more complex, the problem is already NP-hard without any capacities.

Table 1. Complexity Results of Single and Multilevel Lot-Sizing Problems

Problem	Assumptions/remarks	Complexity	Reference
1ILSP	Varying capacities	$\mathcal{O}(T^3)$	Love (1973)
1ILSP	Varying capacities; concave costs	$\mathcal{O}(T^2)$	Hwang and van den Heuvel (2012)
2ILSP	Capacities at second level	$\mathcal{O}(T^4)$	Phouratsamay et al. (2018)
2ILSP	Varying capacities at both levels	NP-hard	Ahmed et al. (2016)
MILSP	Stationary capacities at arbitrary Level and nonspeculative costs	$O(LT^6)$	This paper
MILSP	Stationary capacities at two levels	Open	
1CLSP	Varying capacities	NP-hard	Bitran and Yanasse (1982)
1CLSP	Stationary capacities	$O(T^3)$	van Hoesel and Wagelmans (1996)
2CLSP	Capacity at first level	$O(T^8)$	Kaminsky and Simchi-Levi (2003)
MCLSP	Capacity at first level	$O(LT^8)$	Hwang et al. (2013)
MCLSP	Capacity at arbitrary level	Open	
MICLSP	One different arc capacity value	$O(L^{4L-3}T^{8L-7})$	Ahmed et al. (2016)
MULSP	Assembly structure	NP-hard	Levi and Yedidsion (2013)

We observe from the literature review that the studies on multilevel supply chains with inventory capacities are limited. Furthermore, most existing research focuses on production capacities and in particular on capacities at the first level (with an exception of He et al. (2015) and Ahmed et al. (2016)). To the best of our knowledge, this is the first paper in which a polynomial time algorithm is developed for a problem with inventory bounds in a serial multilevel setting with a general number of levels.

3. Problem Formulation and Optimality Properties

3.1. Problem Formulation

Let T denote the length of the planning horizon and L the number of levels in the supply chain. Our supply chain consists of a manufacturer at the first level, intermediaries like wholesalers and distributors at the middle levels, and a retailer at the last level. Through the operations of production at the first level and then transportation to the following levels, the items are eventually delivered to the retailer facing the customers' demand. We assume that the warehouse at level a, called the bottleneck level, has a finite storage capacity, which means that there is an upper bound on the number of items held in stock. For now, we assume that $1 \le a < L$, so that the bottleneck level is not located at the last level of the supply chain. The case with $a = L$ will be addressed later. We will use index i ($1 \le i \le L$) to denote levels and indices j, b and t (ranging from $1, \dots, T$) to denote periods. In general, index b will be used to denote a period associated with the bottleneck level a. Let U denote the warehouse storage capacity at level a. For each level $i \in \{1, 2, \dots, L\}$ and period $j \in \{1, 2, \dots, T\}$ we define the following parameters:

- d_j: demand in period j at the retailer's level (level L)
- $p_{ij}(x)$: operational cost for an amount x at level i in period j
- $h_{ij}(I)$: cost for holding I items of inventory at level i in period j

Here, we define the decision variables:

- x_{ij}: amount of *operation*, i.e., production at or transportation to level i in period j
- I_{ij}: inventory level at level i at the end of period j

For convenience, we will also use d_{t_1,t_2} to denote the sum of demands in periods $t_1, t_1+1, \dots, t_2$. Now the multilevel inventory bounded lot-sizing problem with the bottleneck at level a (abbreviated as MILSP) can be modeled as

$$\min \quad \sum_{i=1}^{L}\sum_{j=1}^{T}(p_{ij}(x_{ij}) + h_{ij}(I_{ij})) \tag{1a}$$

$$\text{s.t.} \quad I_{i,j-1} + x_{ij} = x_{i+1,j} + I_{ij}, \qquad i = 1, \dots, L-1, \quad j = 1, \dots, T, \tag{1b}$$

$$I_{L,j-1} + x_{L,j} = d_j + I_{L,j}, \qquad j = 1, \dots, T, \tag{1c}$$

$$I_{a,j} \le U, \qquad j = 1, \dots, T, \tag{1d}$$

$$I_{i,0} = I_{i,T} = 0, \qquad i = 1, \dots, L, \tag{1e}$$

$$x_{ij} \ge 0, \qquad i = 1, \dots, L, \quad j = 1, \dots, T, \tag{1f}$$

$$I_{ij} \ge 0, \qquad i = 1, \dots, L, \quad j = 1, \dots, T. \tag{1g}$$

Constraints (1b) and (1c) are the inventory balance equations, which balance the operational amounts and inventory levels through periods and levels. Constraints (1d) model the inventory bound at the warehouse at level a. We

assume that the starting inventory and the ending inventory of the planning horizon is zero at all levels, which is modeled by Constraints (1e). Finally, Constraints (1f) and (1g) show the nonnegativity of the decision variables.

We end this section by presenting the most general assumptions on the cost parameters for which our algorithm presented later will be valid. The cost function $p_{ij}(x)$ has a fixed-charge structure; that is, it consists of a setup cost $K_{ij} \geq 0$ and a per-unit operational cost $p_{ij} \geq 0$. If there are no units produced or transported (i.e., $x = 0$), then there is no operational cost. Hence, the cost for producing or transporting x units is defined as

$$p_{ij}(x) = \begin{cases} 0 & \text{for } x = 0, \\ K_{ij} + p_{ij}x & \text{for } x > 0. \end{cases}$$

The holding cost function has the same fixed-charge structure and is given by

$$h_{ij}(x) = \begin{cases} 0 & \text{for } x = 0, \\ H_{ij} + h_{ij}x & \text{for } x > 0, \end{cases}$$

where H_{ij} is the fixed component and h_{ij} is the variable component. We refer to Atamtürk and Küçükyavuz (2005) and Atamtürk and Küçükyavuz (2008) for situations where such a fixed component occurs.

It is commonly assumed (van Hoesel et al. 2005, section 2.3) that when a product unit moves from one level to the next level in the supply chain, a positive value is added. This phenomenon of added value can be modeled by the *nonspeculative motives* condition:

$$p_{ij} + h_{i,j} \geq h_{i-1,j} + p_{i,j+1} \text{ for } i = 2, \ldots, L \text{ and } j = 1, \ldots, T-1,$$

which we assume to hold throughout the paper. This also implies that when moving an item over multiple levels, waiting with transportation/operation is not more expensive in terms of unit cost. Furthermore, for mathematical tractability, we also assume nonincreasing setup costs at the bottleneck level, that is, $K_{a,b} \geq K_{a,b+1}$ for $b = 1, \ldots, T-1$. In an ordering setting, these fixed costs could consist of order forms, authorization, and inspection, whereas in a manufacturing setting, it could additionally include the wage of a mechanic to set up a machine (Silver et al. 1998). Then the assumption holds if (i) the ordering or production process and hence the costs do not change over time or (ii) if the ordering or production process becomes more efficient over time (e.g., due to learning), causing the costs to decrease over time. As will be shown later (Proposition 3), this ensures that deliveries to the bottleneck warehouse only take place when some units are immediately transported to the next level, implying that the scarce warehouse space is used efficiently. For reference purposes the assumptions are summarized here.

Assumption 1. *The cost structure satisfies the nonspeculative motives condition.*

Assumption 2. *The setup costs are nonincreasing at the bottleneck level.*

The two-level problem with nonstationary inventory bounds at the first level and a general fixed-charge cost structure is NP-hard (Phouratsamay et al. 2018). Hence, we cannot hope for a polynomial-time algorithm unless additional assumptions are posed on the problem like the ones mentioned previously

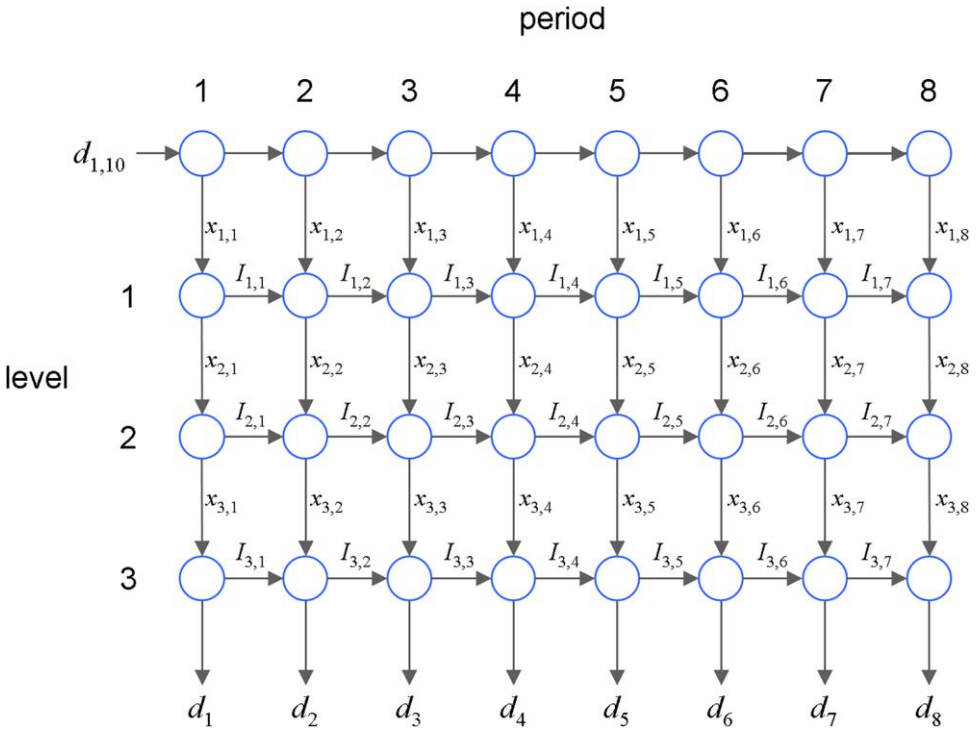
3.2. Optimality Properties and Decomposition

To analyze the model, we consider it as a network flow problem described by Constraints (1b)–(1g), where the flows correspond to the amounts of operation x_{ij} and the inventory levels I_{ij}. The total demand $d_{1,T}$ from a super source node is distributed over the network to the T retailer's nodes (the sink nodes). An example of a problem with eight periods and three levels is illustrated in Figure 1. A node (i, j) denotes level i and period j and has an incoming operational flow x_{ij} (for $i \geq 1$) and incoming inventory flow $I_{i,j-1}$ (for $j \geq 2$). Each node (a, b) at the bottleneck level is called a *bottleneck node*. Nodes on levels before the bottleneck level a are called *upstream nodes*, whereas nodes on levels after the bottleneck level are called *downstream nodes*.

As we will show in the remainder of this section, a typical extreme point solution corresponding to the network flow problem is as shown in Figure 2. To have a more compact representation, the vertical arcs leaving the retailer's nodes at level L are removed as they only indicate the demands and the flows of these arcs are fixed. Furthermore, the network does not contain the arcs entering the manufacturer's nodes at level 1, but production at level 1 is indicated by a shaded *production node*, which now serves as a source node. The arcs with *free flows*, that is, flow amounts strictly between their lower and upper bounds, are indicated by solid arrows. The flows corresponding to the inventory variables with values equal to the upper bound U (at the bottleneck level a) are depicted by dashed arrows.

Before showing the structural properties of an optimal solution, we first introduce some definitions. A node (i, j) in a solution satisfies the *single-sourcing* (respectively, *double-sourcing*. property if there is not (respectively, is) both

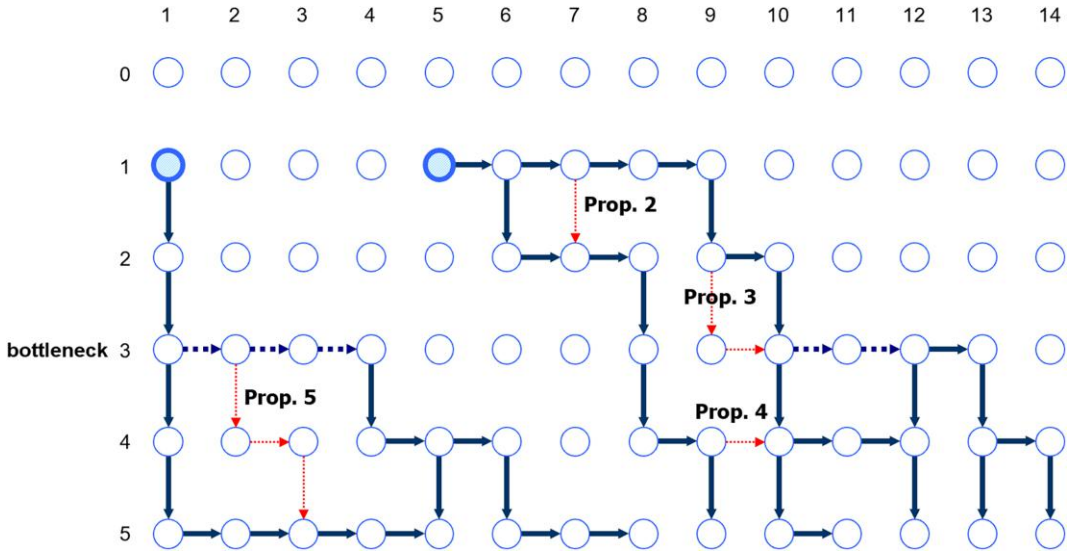
Figure 1. (Color online) Production and Transportation Network with Three Levels and Eight Periods



positive operational inflow and positive inventory inflow, that is, $I_{i,j-1}x_{ij} = 0$ (respectively, $I_{i,j-1}x_{ij} > 0$). A bottleneck node (a, b) with operational flow $x_{a,b} > 0$ is called a *spring node*. A *downstream path* is an undirected path of which each arc has positive flow and each arc is not at the upstream or bottleneck level. For instance, in Figure 2, we have the downstream path $(3,10) - (4,10) - (4,11) - (4,12) - (3,12)$.

To show why a typical extreme point solution is characterized by a structure as shown in Figure 2, we also present arcs which violate the structural properties, indicated by the dotted arcs. In summary, we will prove that there exists an optimal solution where (i) there are no double sourcing nodes at the upstream and bottleneck level

Figure 2. (Color online) Network Flows of a Typical Extreme Point Solution



Note. Dotted arcs show the violation of some property.

(Proposition 2), (ii) any spring node (a, b) also has operational flow at $(a, b+1)$ (Proposition 3), (iii) two spring nodes cannot be connected by a downstream path (Proposition 4), and (iv) any spring node is connected to at most one double sourcing node (Proposition 5).

To prove the structural properties in our propositions, we mainly use the so-called "cycle free" property that holds for network flow problems with concave costs (Ahuja et al. 1993) in combination with the network structure and cost assumptions of our particular problem.

Proposition 1 (Ahuja et al. 1993). *The subnetwork of arcs with free flows corresponding to an extreme point solution of MILSP does not contain an (undirected) cycle.*

As we can see in Figure 2, the subnetwork of arcs with free flows (of solid arcs) does not contain any cycle. The next proposition states that the single-sourcing property holds for upstream and bottleneck nodes.

Proposition 2. *Under Assumption 1, there exists an optimal solution such that the single-sourcing property holds for each node (i, j) with $i = 1, \ldots, a$ and $j = 1, \ldots, T$.*

Proof. Consider an extreme point solution and its corresponding subnetwork of arcs with positive flow. Suppose that $I_{i,j-1}x_{ij} > 0$ and hence $I_{i,j-1} > 0$ and $x_{ij} > 0$. Clearly, there is a path $\mathcal{P}_1$ (respectively, $\mathcal{P}_2$) consisting of arcs with positive flow from the super source node to (i, j) through node $(i, j-1)$ (respectively, node $(i-1, j)$), and these paths comprise a cycle. If (i, j) is an upstream node (i.e., $i < a$), then all arcs in this cycle are free flow arcs, and we have a contradiction with Proposition 1. Therefore, we assume that (i, j) is a bottleneck node (i.e., $i = a$) and hence $\mathcal{P}_1$ contains at least one bottleneck arc. Because of Assumption 1, sending one unit of flow over path $\mathcal{P}_1$ is at least as expensive in terms of variable costs (i.e., per unit operational and holding costs) as sending a unit over path $\mathcal{P}_2$. Hence, redirecting all flow from path $\mathcal{P}_1$ to $\mathcal{P}_2$ does not increase the total costs. Finally, feasibility is guaranteed, because the inventory levels at the bottleneck level will not increase. □

As we can see in Figure 2, each upstream or bottleneck node has indeed the single-sourcing property. The single-sourcing property holds for all nodes in the classical uncapacitated lot-sizing model, in which each connected component is a so-called arborescent tree (called a Zangwill tree in this paper). This structure is exploited in the decomposition approach of Zangwill (1969). Because an optimal solution may have double-sourcing nodes in MILSP, this approach is inadequate and hence another approach should be applied.

It will turn out that spring nodes play a key role in designing an efficient solution approach. For such a node, Proposition 2 implies that the beginning inventory can be assumed to be zero, that is, $I_{a,b-1} = 0$. If we have no flow from the bottleneck level to the next level ($x_{a+1,b} = 0$) and if $K_{a,b} \geq K_{a,b+1}$, then we can postpone the delivery of period b to period $b+1$ at no additional cost. Thus, we can see that the inventory capacity is used efficiently under Assumption 2. This is formalized in the following proposition.

Proposition 3. *Under Assumptions 1 and 2, there exists an optimal solution such that $x_{a,b} > 0$ implies $x_{a+1,b} > 0$ for $b = 1, \ldots, T$.*

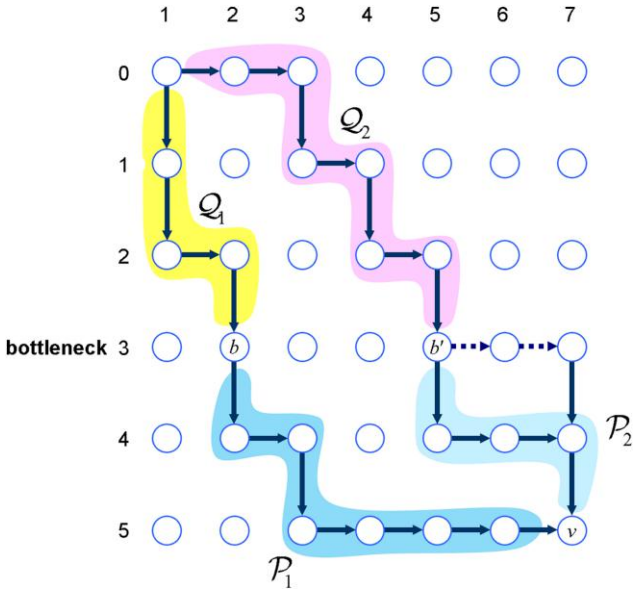
Next to spring nodes, downstream paths connecting two bottleneck nodes, being referred to as basis paths (see Section 4), will also play an important role in the solution approach.

Proposition 4. *There exists an optimal solution such that two distinct spring nodes (a, b) and (a, b') are not connected by a downstream path.*

Proof. Consider an optimal solution that satisfies all properties stated in the preceding propositions and assume that it has a downstream path consisting of arcs with positive flows between (a, b) and (a, b'). Without loss of generality, assume that $b < b'$ and that there are no spring nodes in between b and b'. Because $x_{a,b'} > 0$ it follows from Proposition 2 that $I_{a,b'-1} = 0$. Hence, the path cannot consist of only bottleneck nodes. This means there should be a double-sourcing node v such that there is positive flow along a (directed) path $\mathcal{P}_1$ from (a, b) to v and along a path $\mathcal{P}_2$ from (a, b') to v (see Figure 3 for a visual illustration). In case there are several of such double sourcing nodes, we assume that $v = (i, j)$, with i minimal, and j minimal for given i.

Clearly, path $\mathcal{P}_2$ contains either the arc from (a, b') to $(a+1, b')$ or the bottleneck arc from (a, b') to $(a, b'+1)$. We will argue that if there exists a path with positive flow from (a, b') to v that uses the bottleneck arc, then there must also exist a path from (a, b') to v that uses the arc from (a, b') to $(a+1, b')$. Because (a, b') is a spring node, Proposition 3 implies that $x_{a+1,b'} > 0$. Because this flow will eventually reach level L, there must be a path starting at $(a+1, b')$ and consisting of arcs with positive flows that either intersects with the path from (a, b') to v that uses the bottleneck arc, or it intersects path $\mathcal{P}_1$. Let v' be the first node where the intersection occurs. Then either

Figure 3. (Color online) Cycle $\mathcal{C}$ in the Proof of Proposition 4



$v' = v$ and we have a path through $(a+1, b')$, or we have a contradiction with the minimality of v (because of the orientation of the arcs).

Furthermore, both spring nodes (a, b) and (a, b') receive positive flow from the super source node via paths $\mathcal{Q}_1$ and $\mathcal{Q}_2$, respectively. Hence, the component

$$\mathcal{C} = \mathcal{P}_1 \cup \mathcal{P}_2 \cup \mathcal{Q}_1 \cup \mathcal{Q}_2$$

contains an (undirected) cycle of arcs with positive flows as illustrated in Figure 3. This is a contradiction with Proposition 1, which completes the proof. □

Observe that the network of free flow arcs in Figure 2 has two connected components. We call such a connected component a *regeneration network*. As we see from this figure, a regeneration network is characterized by a first bottleneck period b_1 and a last bottleneck period b_2, and a first demand period t_1 and a last demand period t_2 satisfied by the regeneration network. We use a tuple $N = (b_1, b_2, t_1, t_2) \in \mathcal{N}$ to represent a regeneration network where

$$\mathcal{N} = \{(b_1, b_2, t_1, t_2) \in \mathbb{N}^4 : 1 \le b_1 \le b_2 \le t_2 \le T, b_1 \le t_1 \le t_2\}. \tag{2}$$

is the set of all regeneration networks. When referring to N, we mean either a tuple in the index set $\mathcal{N}$ or a whole regeneration network (consisting of arcs), which should be clear from the context. As an example, in Figure 2 we have two regeneration networks $N = (1, 4, 1, 8)$ and $N = (8, 13, 9, 14)$.

Because of Proposition 1, we can assume that each regeneration network is acyclic. Now consider a node $v = (i, j)$ in a regeneration network. Let us remove the inventory inflow arc from node $(i, j-1)$ (if it exists) and the operational inflow arc from $(i-1, j)$ (if it exists). The component containing node v is called a *v-rooted tree*. In particular, if $i < a$ (respectively, $i > a$) the v-rooted tree is called an *upstream tree* (respectively, *downstream tree*). Moreover, when (a, b) is a spring node, its tree is called *(a, b)-rooted spring tree*. These trees differ in the levels they run from: an upstream (respectively, spring, downstream) tree represents a subnetwork of free flow arcs running from an upstream level $i < a$ (respectively, bottleneck level $i = a$, downstream level $i > a$) down to the last level L. The following important property is a direct consequence of Proposition 4.

Corollary 1. *Each regeneration network has exactly one production node.*

Proof. Proposition 4 implies that each (a, b)-rooted spring tree has exactly one spring node, namely (a, b). Furthermore, by Proposition 2 each upstream node satisfies the single-sourcing property and has at most one incoming arc with positive flow. Combining this with the fact that the regeneration network is connected, gives the result. □

As another consequence of Proposition 4, we can assume that the root node (a, b) of a (a, b)-rooted spring tree $\mathcal{T}$ is the left- and top-most node in $\mathcal{T}$, i.e., each node (i, j) in $\mathcal{T}$ satisfies $i \ge a$ and $j \ge b$. This is because any node (i, j) in $\mathcal{T}$ with $j < b$ would imply the existence of a spring node (a, b') with $b' < b$ in $\mathcal{T}$, whereas any node (i, j) in $\mathcal{T}$ with $j > b$

and $i<a$ would imply the existence of a spring node (a,b') with $b'>b$ in $\mathcal{T}$. In both cases, there would be a downstream path between (a,b) and (a,b'), which we can assume does not exist. It follows from Propositions 2 and 4 that we can also assume this property to hold for upstream trees: the root node (i,j) of any (i,j)-rooted tree with $i<a$ is the left- and top-most node. However, this property does not hold in general for (i,j)-rooted downstream trees with $i>a$. For instance, node (4, 4) in the (5, 3)-rooted tree in Figure 2 is above the root node (5, 3).

In contrast to the upstream and bottleneck nodes, in general there exist downstream nodes (i,j) for which the double-sourcing property holds. Although this breaks down the tree structure when considering the network of positive flows, the next proposition shows that this gives us information about the inventory levels at the bottleneck nodes.

Proposition 5. *Consider an (a,b)-rooted spring tree with a double-sourcing node (i,j). Then* (i) *there is a bottleneck node (a,b^*), $b^*>b$ to which (i,j) is connected by arcs with positive flow such that $I_{a,b'}=U$ for $b'=b,\dots,b^*-1$, and* (ii) *there is no other double-sourcing node.*

Proof. Suppose that node (i,j) is contained in an (a,b)-rooted spring tree $\mathcal{T}$ where $i>a$ and $j\geq b$. Let $\mathcal{P}_1$ be the path from (a,b) to (i,j). Because $I_{i,j-1}x_{ij}>0$, it holds that $j>b$ and hence either $(i,j-1)$ or $(i-1,j)$ is contained in $\mathcal{P}_1$ but not both. If $(i-1,j)$ is on the path, then node $(i,j-1)$ must be connected to another bottleneck node (a,b'), which precedes the spring node (a,b). However, this contradicts the fact that (a,b) is the left- and top-most node of $\mathcal{T}$. Hence, the path $\mathcal{P}_1$ is of the form $\{(a,b),\dots,(i,j-1),(i,j)\}$.

Now consider the flow $x_{i,j}$. Clearly, there must also be a directed path $\mathcal{P}_2$ from a bottleneck node, say (a,b^*), to $(i-1,j)$. Let $\mathcal{P}_2$ be the path such that (a,b^*) is the only bottleneck node of $\mathcal{P}_2$. The cycle free property implies that (a,b^*) cannot be equal to (a,b). Thus we have a downstream path $\mathcal{P}$ between the bottleneck nodes (a,b) and (a,b^*) given by

$$\mathcal{P}=\mathcal{P}_1\cup\mathcal{P}_2.$$

Now consider the bottleneck nodes (a,b) and (a,b^*). Because (a,b) is the only spring node and node (a,b^*) has an outflow of at least $x_{i,j}$ units, we know that there is flow from (a,b) to (a,b^*), that is, $I_{a,\ell}>0$ for $b\leq\ell<b^*$. We now show by contradiction that these flow amounts equal U. If $I_{a,b}<U$, then clearly $I_{a,\ell}<U$ for $b\leq\ell<b^*$ and we have a cycle of free flows, which is a contradiction.

Consider the other case in which there must be a node (a,ℓ), $b<\ell<b^*$, such that $0<I_{a,\ell-1}=U$ and $0<I_{a,\ell}<U$, which further implies that there is an outflow from node (a,ℓ) with $x_{a+1,\ell}>0$ and a directed path of free flow arcs from (a,ℓ) to (a,b^*). Because the flow $x_{a+1,\ell}$ must eventually reach retailer level L by some path of free flows, say path $\mathcal{Q}$, this path $\mathcal{Q}$ must intersect path $\mathcal{P}$ at some node, say (i',j'), which also must be a double sourcing node. Then the path from (a,ℓ) to (i',j'), from (i',j') to (a,b^*) and then back to (a,ℓ) yields a cycle of free arcs, which is a contradiction. Therefore, we conclude that (i) $I_{a,b}=I_{a,b^*-1}=U$ and hence $I_{a,b}=I_{a,b+1}=\cdots=I_{a,b^*-1}=U$, and (ii) there cannot be another double sourcing node (i',j') on path $\mathcal{P}$, and hence in the (a,b)-spring tree. □

4. Dynamic Programming Solution Approach

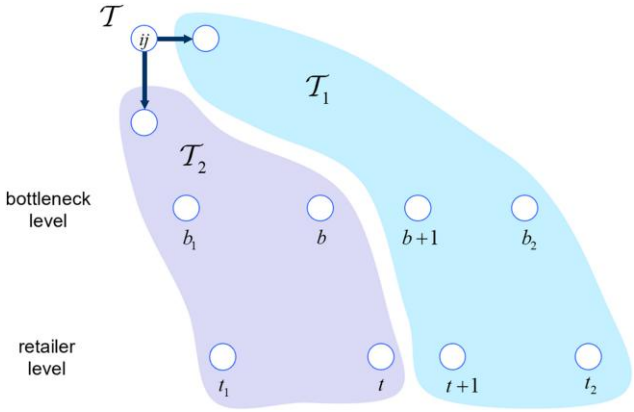
4.1. Overview of the Dynamic Programming Approach

The algorithm consists of two main parts, one is for computing the costs of the upstream trees and the other one for the *downstream networks*. Once the costs of the downstream networks are determined by a dynamic programming (DP) approach (Section 4.3), the costs of the upstream trees can be computed also by a DP approach. With the procedures, we ultimately obtain the optimal value and an optimal sequence of consecutive regeneration networks (Section 4.2).

To compute the costs of the upstream trees and finally the regeneration networks, we exploit the single-sourcing property (see Proposition 2). For uncapacitated multilevel lot-sizing problems, this allows for a decomposition in which a component is characterized by which retailer demands are satisfied from a particular level and period (Zangwill 1969). However, we need to generalize this decomposition approach by additionally keeping track of the bottleneck periods in order not to violate the inventory capacities. In Figure 4, we show schematically how an upstream tree is decomposed in two smaller upstream trees. Ultimately, the optimal overall cost will be obtained by observing that an optimal solution consists of a sequence of consecutive regeneration networks.

The main challenge is to find the cost of each downstream network, which consists of a spring tree expanded with the *saturated* arcs connecting bottleneck nodes (see Proposition 5). The structure of a downstream network with a double-sourcing node is schematically illustrated in Figure 5. A downstream network consists of a so-called basis path (downstream path) connecting the first and last bottleneck nodes (see Section 4.3.1), where

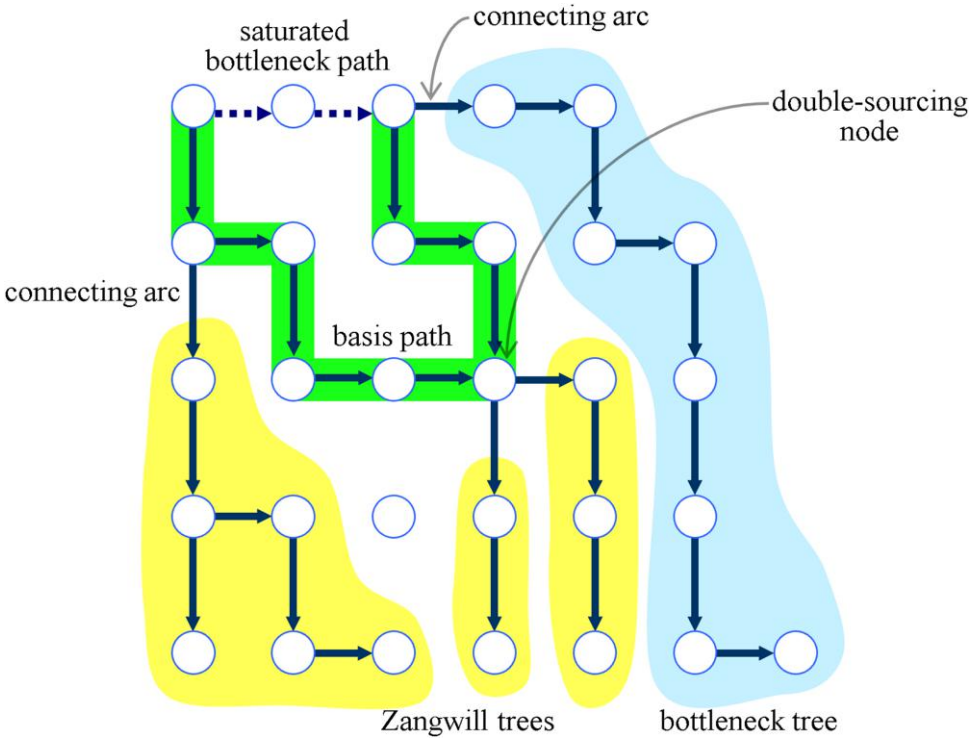
Figure 4. (Color online) Decomposition of Upstream Tree



both Zangwill trees and a bottleneck tree (to be formally defined later) are connected to this path (see Section 4.3.2 for the cost computation). The costs of a downstream network are computed by moving along the basis path and adding the relevant components and their costs. Because the number of possible basis paths is exponential, we will not enumerate them, but instead we use the basis path approach of Hwang et al. (2013) where the optimal path is computed recursively by keeping track of the last two nodes of the path (see Section 4.3.4). The costs of the arcs connecting the components with the basis path (called the transition costs) depend on (i) the position in the path (before or after the source node) and (ii) the configuration of the two connected arcs (see Section 4.3.3).

In summary, our solution approach can be viewed as a combination of Zangwill's decomposition method and the basis-path approach. However, a straightforward combination of the two methods leads to a high-complexity algorithm. It turns out that the running time is dominated by the determination of upstream-tree costs. Exploiting the nonspeculative motives condition, we show that one can ignore the possible overlap when combining two upstream trees in a bigger one. This property leads to a reduction of a factor T in running time (see Section 4.4).

Figure 5. (Color online) Decomposition of a Downstream Network



4.2. Outer DP for Upstream Trees

In this section, we will show how to solve the overall problem assuming that the costs of the (a, b)-rooted spring trees are known.

4.2.1. DP for Upstream Trees. As follows from the previous section, each (i, j)-rooted upstream tree satisfies a number of consecutive demands using bottleneck periods in some interval. Let $c_N(i,j)$ be the minimal cost of the (i, j)-rooted upstream tree associated with $N = (b_1, b_2, t_1, t_2)$ (so $i < a$ and $j \le b_1$) and hence satisfying demands in the interval $[t_1, t_2]$ using bottleneck periods in the interval $[b_1, b_2]$. The cost $c_N(i,j)$ can be recursively computed by realizing that the (i, j)-rooted upstream tree can be decomposed in two smaller rooted upstream trees (Figure 4). To see this, let $v = (i,j)$ and consider the v-rooted tree $\mathcal{T}$ and its smaller upstream trees $\mathcal{T}_1$ with root $v_1 = (i, j+1)$ and $\mathcal{T}_2$ with root $v_2 = (i+1, j)$. Any positive flow leaving an upstream node reaches a bottleneck node, which must be a spring node. Suppose that the flow from v_1 (respectively, v_2) runs through a spring node s_1 in $\mathcal{T}_1$ (respectively, s_2 in $\mathcal{T}_2$). We know that $\mathcal{T}_1$ and $\mathcal{T}_2$ are disjoint because no path of upstream nodes connects v_1 and v_2 due to the single-sourcing property (Proposition 1), which means $s_1 \neq s_2$. Moreover, any two spring nodes are not connected by a downstream path of positive flows (Proposition 4). Hence, the upstream tree $\mathcal{T}$ decomposes into the disjoint upstream trees $\mathcal{T}_1$ and $\mathcal{T}_2$. As a result, the optimal cost of an (i, j)-rooted upstream tree is given by the recursion

$$c_N(i,j) = \min_{\substack{N'=(b_1,b,t_1,t),N''=(b+1,b_2,t+1,t_2):\\ b_1\le b<b_2,\, t_1-1\le t\le t_2}} \{p_{i+1,j}(d_{t_1,t}) + c_{N'}(i+1,j) + h_{i,j+1}(d_{t+1,t_2}) + c_{N''}(i,j+1)\}, \tag{3}$$

where in the initialization we need the costs $c_N(a, b_1)$ which are computed in Section 4.3.4. Because of Proposition 1, the optimal cost $c(N)$ of a regeneration network $N = (b_1, b_2, t_1, t_2)$ can be found by optimizing over the possible production nodes at level 1:

$$c(N) = \min_{1\le j\le b_1} \{c_N(1,j)\}. \tag{4}$$

4.2.2. DP for Overall Problem. Given the optimal cost $c(N)$ of each regeneration network $N = (b_1, b_2, t_1, t_2)$ and using the fact that an optimal solution consists of a sequence of regeneration networks, we can use a straightforward DP to find the optimal objective value of the overall problem. To that end, we define the DP variable $F(b_2, t_2)$ as the minimum cost to satisfy demands in $[1, t_2]$ using bottleneck periods $[1, b_2]$ by using a number of consecutive regeneration networks. With the initial states $F(0,0) = 0$ and $F(0,t) = \infty$ for $t \in [1, T]$, the DP variables $F(b_2, t_2)$ can be computed by the recursion

$$F(b_2,t_2) = \min\left\{ F(b_2-1,t_2),\ \min_{\substack{N=(b_1,b_2,t_1,t_2):\\ 1\le b_1\le b_2,\,1\le t_1\le t_2}} \{F(b_1-1,t_1-1) + c(N)\} \right\}. \tag{5}$$

The first term in the right-hand side of (5) is needed because there may be more than one unused bottleneck arc between two consecutive regeneration networks (e.g., the bottleneck arcs between periods 4 and 7 in Figure 2). Clearly, the optimal objective value is given by $F(T, T)$.

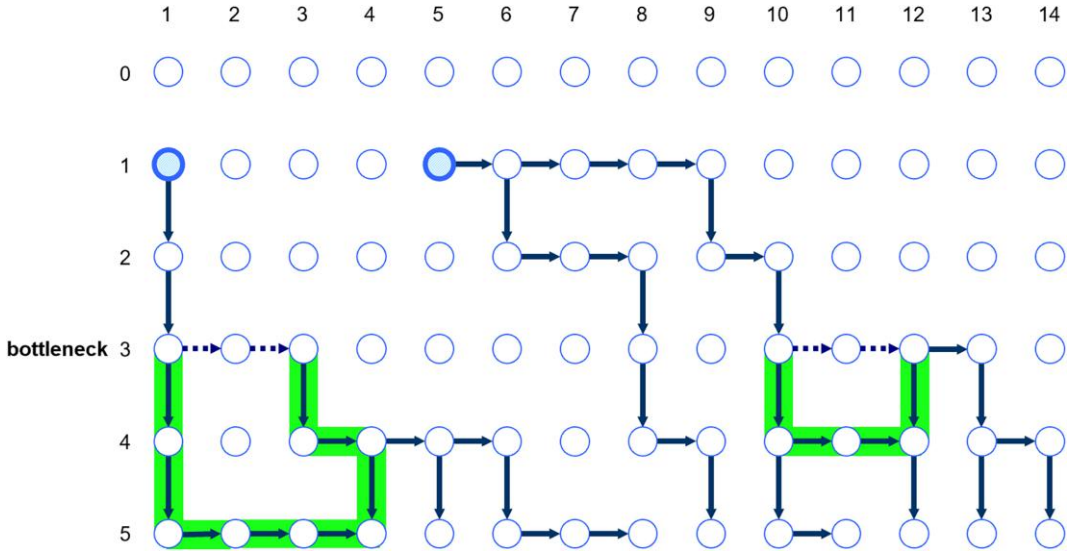
4.2.3. Running Time. The computational complexity to compute all values of $c_N(i,j)$ and $c(N)$ by (3) and (4) is $\mathcal{O}(LT^7)$ and $\mathcal{O}(T^5)$, respectively. Because the computational complexity to compute all $F(b_2, t_2)$ by (5) is $\mathcal{O}(T^4)$, the overall complexity to find the optimal solution is $\mathcal{O}(LT^7)$ assuming that the optimal costs of all rooted spring trees have been precomputed. In Section 4.4, we show how to reduce the time complexity by a factor of T to $\mathcal{O}(LT^6)$.

4.3. Inner DP for Downstream Networks

4.3.1. Basis Path Concept. In this section, we will focus on the structure of an (a, b)-rooted spring tree $\mathcal{T}$ and show how to decompose it further to determine the optimal costs. By definition, it holds that $x_{a,b} > 0$. We have seen that (a, b) can be assumed to be the left- and top-most node in $\mathcal{T}$. Furthermore, it follows from Proposition 5, that we can assume that $\mathcal{T}$ contains at most one double-sourcing node.

First, we examine in more detail an (a, b)-rooted spring tree containing a double-sourcing node. Using Proposition 5, we can assume that there is a bottleneck node (a, b^*) with $b^* > b$, such that $I_{a,b'} = U$ for $b' = b, \ldots, b^* - 1$. Moreover, because $\mathcal{T}$ is a tree, there is a unique path $\mathcal{P}$ between the node (a, b) and the node (a, b^*) not containing any other bottleneck node. We call this the *basis path* of the (a, b)-rooted spring tree (Hwang et al. 2013), and

Figure 6. (Color online) Upstream Trees and Downstream Basis Paths



node (a, b) (respectively, (a, b^*)) the *opening node* (respectively, *closing node*) of the basis path. For example, the path $\mathcal{P} = \{(3,10),(4,10),(4,11),(4,12),(3,12)\}$ in Figure 6 is a basis path where (4, 12) is the double-sourcing node, whereas (3, 10) and (3, 12) are the opening and closing nodes of the path.

Because we are interested in the total cost and hence the arcs with positive flow, it is convenient to expand the (a, b)-rooted spring tree $\mathcal{T}$ with the nodes and arcs in between the opening node (a, b) and the closing node (a, b^*) at the bottleneck level. All arcs have flow at their upper bounds and, for this reason, we call the path from (a, b) to (a, b^*) a *saturated bottleneck path*. Furthermore, we call the extension of the (a, b)-rooted spring tree with the saturated bottleneck path a *downstream network*.

Besides the basis path $\mathcal{P}$ and the saturated bottleneck path, the downstream network consists of other components. First of all, consider a nonbottleneck node $u \in \mathcal{P}$ and arc $e = (u, v)$ with $e \in \mathcal{T}$ and $e \notin \mathcal{P}$. Then the v-rooted tree, denoted by $\mathcal{T}(v)$, must satisfy a consecutive number of demands (this is because all nodes in $\mathcal{T}(v)$ satisfy the single-sourcing property as the arcs in this part of the network have no upper bound). We call such a $\mathcal{T}(v)$ a *Zangwill tree*, because, as we shall see later, the optimal costs can be determined by a DP as in Zangwill (1969).

Furthermore, consider the closing node $u = (a, b^*)$ and an arc $e = (u, v) \in \mathcal{T}$ with $v = (a, b^* + 1)$ (if any). Again the v-rooted tree $\mathcal{T}(v)$ satisfies a consecutive number of demands. We call such a component a *bottleneck tree*. Because the bottleneck arcs have an upper bound on the flow, bottleneck trees are slightly different from Zangwill trees in terms of computing the minimal costs. Finally, we call the arcs e that connect the Zangwill trees or bottleneck tree to the basis path *connecting arcs*.

In conclusion, each downstream network can be decomposed in (i) a basis path, (ii) a saturated bottleneck path, (iii) Zangwill trees, (iv) a bottleneck tree, and (v) connecting arcs. Figure 5 illustrates the different components of a downstream network. Not all components need to be present. For instance, in case the downstream network does not contain a double-sourcing node, there is no basis path containing arcs, but it consists of a single node. In this degenerate situation, the downstream network consists of a single Zangwill tree or a single bottleneck tree. In the next sections, we will use the decomposition to find the optimal cost of downstream networks.

4.3.2. Cost Computation of Zangwill and Bottleneck Trees.

As follows from the previous section, Zangwill trees and bottleneck trees are important components of a downstream network. As they will be used in the DP to compute the minimal cost of a downstream network (see Section 4.3.4), we show in this section how to compute their minimal costs.

4.3.2.1. Zangwill Trees.

We start with showing how to compute the minimal cost of Zangwill trees. First, because there are no upper bounds on the arc flows below the bottleneck level, there exists an optimal solution such that the single-sourcing property is satisfied. Because of this, we can apply a recursive equation that is very similar to the algorithm in Zangwill (1969). To formalize the DP algorithm, let $\phi^{i,j}_{t_1,t_2}$ be the minimum cost to satisfy demands $d_{t_1}, \ldots, d_{t_2}$ having d_{t_1,t_2} units available at level i in period j, where $i = a+1, \ldots, L$ and $j = 1, \ldots, T$. By realizing that

each Zangwill tree consists of two smaller Zangwill tree s, all values $\phi^{i,j}_{t_1,t_2}$ can be computed by the following recursive equation:

$$\phi^{i,j}_{t_1,t_2} = \min_{t_1-1\le t\le t_2} \left\{ p_{i+1,j}(d_{t_1,t}) + \phi^{i+1,j}_{t_1,t} + h_{i,j+1}(d_{t+1,t_2}) + \phi^{i,j+1}_{t+1,t_2} \right\}, \tag{6}$$

with the initialization $\phi^{L,t}_{t,t} = 0$ for $t \in [1,T]$ and $\phi^{i,j}_{t_1,t_2} = 0$ if $t_1 > t_2$. Further details can be found in Zangwill (1969) (see also van Hoesel et al. (2005) and Hwang et al. (2013)). We finally mention that by using (6), all values $\phi^{i,j}_{t_1,t_2}$ can be obtained in $\mathcal{O}(LT^4)$ time.

4.3.2.2. Bottleneck Trees. The computation of bottleneck trees is slightly more involved than the computation of Zangwill trees because we should take care of the inventory bound at the bottleneck level. Furthermore, we have to keep track of the bottleneck periods used because we need these for Recursion (3). To that end, we let $\psi^{b_1,b_2}_{t_1,t_2}$ be the minimum cost to satisfy demands $d_{t_1},\dots,d_{t_2}$ using bottleneck periods $b_1,\dots,b_2$ having d_{t_1,t_2} units available at level a in period b_1. It is not difficult to see that again the single-sourcing property holds. This implies that each bottleneck tree can be decomposed in a smaller bottleneck tree and a Zangwill tree, leading to the following recursion:

$$\psi^{b_1,b_2}_{t_1,t_2} = \min_{\substack{t_1-1\le t\le t_2:\\ d_{t+1,t_2}\le U}} \left\{ p_{a+1,b_1}(d_{t_1,t}) + \phi^{a+1,b_1}_{t_1,t} + h_{a,b_1}(d_{t+1,t_2}) + \psi^{b_1+1,b_2}_{t+1,t_2} \right\}, \tag{7}$$

with the initialization $\psi^{b,b}_{t_1,t_2} = \phi^{a,b}_{t_1,t_2}$. The main differences with (6) are that we need to check the feasibility condition $d_{t+1,t_2} \le U$, and that we keep track of the bottleneck periods that are used. At this point it is also worth mentioning that if the downstream network that uses bottleneck periods $b_1,\dots,b_2$ to satisfy demands $d_{t_1},\dots,d_{t_2}$ does not contain a double sourcing node (and hence no basis path), the minimum cost is given by $\psi^{b_1,b_2}_{t_1,t_2}$. Finally, we note the time complexity of computing all values $\psi^{b_1,b_2}_{t_1,t_2}$ by (7) is $\mathcal{O}(T^5)$.

4.3.3. Configuration of Basis Nodes and Transition Cost. As we will see in Section 4.3.4, in the DP to compute the optimal cost of a downstream network we will iterate over the nodes on a basis path, to which we will refer as *basis nodes*. When moving from one basis node to the next basis node, the cost of the connected components will be accounted for. We will show in this section that the type of component will depend on the configuration of three consecutive nodes (or two consecutive arcs) on the basis path. Therefore, we examine the possible configurations of three consecutive nodes. Furthermore, we show how to compute the relevant costs when iterating over the basis path.

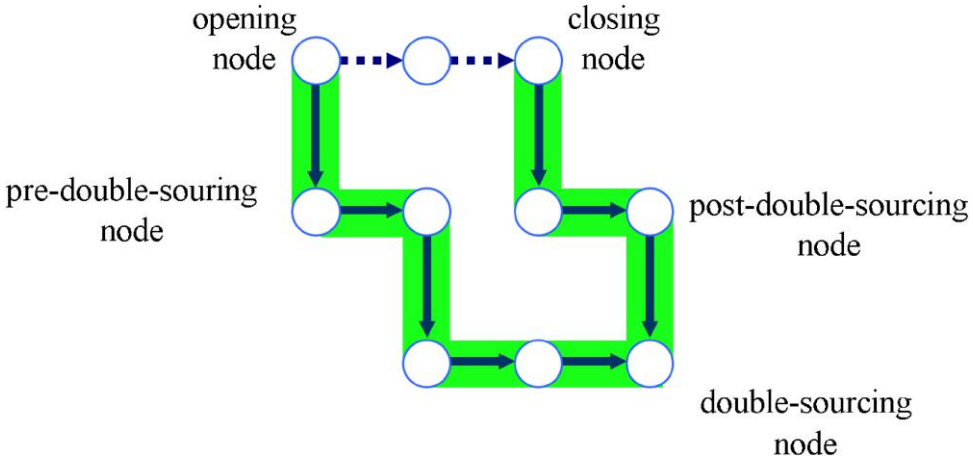
For the moment, assume that the basis path is given and consider three consecutive nodes u, v, and w on the basis path. We distinguish between three cases dependent on the location of node v relative to the double-sourcing node. This means that node v is either (i) a double-sourcing node, (ii) a *pre–double-sourcing* node, or (iii) a *post–double-sourcing* node, where in case (ii) (respectively, (iii)) node v precedes (respectively, succeeds) the double-sourcing node. Figure 7 illustrates the three possible cases.

We first deal with the case that basis node $v = (i,j)$ is a e-sourcing node. Because of this property, it is not difficult to see (e.g., from Figure 7) that node u is either $(i-1,j)$ or $(i,j-1)$, and node w is either $(i+1,j)$ or $(i,j+1)$, resulting in four possible configurations. In case node v is a post–double-sourcing node, again we have four possible configurations, whereas there is only one possible configuration when v is a double-sourcing node. The nine possible configurations are illustrated in Figure 8.

Before we provide the details, we start with explaining the general idea on how we do the cost accounting when moving over the basis path. As follows from Section 4.3.1, each component connected to a node v on the basis path consists of either a Zangwill tree, or a bottleneck tree. To be precise, this is the component that remains after removing the arcs on the basis path connected to v and is denoted by $\hat{\mathcal{T}}(v)$ (Figure 9). Such a component is slightly different from a rooted tree $\mathcal{T}(v)$ (and hence the new notation), as for a rooted tree $\mathcal{T}(v)$ we removed the incoming arcs, whereas for $\hat{\mathcal{T}}(v)$, we removed the arcs on the basis path (which may be outgoing arcs). We know that $\hat{\mathcal{T}}(v)$ covers exactly the demands of the retailer's nodes in the tree (i.e., the interval $[t'+1,t]$ in Figure 9). It follows that if $\mathcal{P}' = \{(a,b_1),(a+1,b_1),\dots,v\}$ is a partial basis path in some downstream network $N = (b_1,b_2,t_1,t_2)$, then the nodes in $\mathcal{P}'$ must cover exactly the demands in some interval $[t_1,t]$. This results in the following state definition.

Definition 1. The tuple (N,v,w,t) denotes the state in which v and w are consecutive basis nodes in a downstream network $N = (b_1,b_2,t_1,t_2)$, where demands $d_{t_1,t}$ are satisfied from basis nodes (a,b_1) up to v.

Figure 7. (Color online) Pre–Double-Sourcing, Double-Sourcing, and Post–Double-Sourcing Nodes



We are now ready to specify the cost incurred when moving from state (N,u,v,t') to state (N,v,w,t), which we denote by the *transition cost* $g_N(u,v,w)_{t',t}$. In particular, $g_N(u,v,w)_{t',t}$ includes

1. The cost associated with tree $\hat{\mathcal{T}}(v)$ to cover the demands in $[t'+1,t]$
2. The cost of the flow of the arc between u and v (dependent on the configuration it is arc (u,v) or (v,u))

These costs depend on the configuration of nodes u, v and w, and this is the reason that the node w is needed in Definition 1. Instead of considering the nine cases corresponding to all possible configurations, we choose three typical cases (e-sourcing, double-sourcing, and post–double-sourcing). The other cases can be handled in a similar way.

Case 1 **(e-sourcing):** $u=(i-1,j), v=(i,j), w=(i,j+1)$.

In this case (Figure 9), the tree $\hat{\mathcal{T}}(v)$ consists of a Zangwill tree satisfying demands $d_{t'+1,t}$, where the connecting arc in this tree must be an operational one because of the configuration of u, v and w. This means that the cost of $\hat{\mathcal{T}}(v)$ consists of $p_{i+1,j}(d_{t'+1,t})$ (the connecting arc) plus $\phi^{i+1,j}_{t'+1,t}$ (the remaining Zangwill tree). Furthermore, we need to compute the flow cost of arc (u,v). Because the flow on the first arc of the basis path equals $d_{t_1,t_2}-U$ (because demands d_{t_1,t_2} are satisfied from the total outflow of the first bottleneck node, and the inventory flow at the bottleneck level is at capacity) and because demands $d_{t_1,t'}$ are satisfied by the nodes prior to node v, the flow on arc (u,v) equals $(d_{t_1,t_2}-U)-d_{t_1,t'}=d_{t'+1,t_2}-U$, having an associated cost of $p_{i,j}(d_{t'+1,t_2}-U)$. Combining the previous statements leads to the transition cost

$$g_N(u,v,w)_{t',t}=p_{i,j}(d_{t'+1,t_2}-U)+p_{i+1,j}(d_{t'+1,t})+\phi^{i+1,j}_{t'+1,t}.$$

Case 2 **(double-sourcing):** $u=(i,j-1), v=(i,j), w=(i-1,j)$.

We determine the cost associated with the downstream tree $\hat{\mathcal{T}}(v)$ that covers demands $d_{t'+1},\ldots,d_t$. This is exactly the cost of satisfying demands $d_{t'+1},\ldots,d_t$ from node $v=(i,j)$, which is the cost $\phi^{i,j}_{t'+1,t}$ of a Zangwill tree. As in the e-sourcing case, the flow on arc (u,v) is $d_{t'+1,t_2}-U$, incurring holding cost $h_{i,j-1}(d_{t'+1,t_2}-U)$. This leads to the transition cost

$$g_N(u,v,w)_{t',t}=h_{i,j-1}(d_{t'+1,t_2}-U)+\phi^{i,j}_{t'+1,t}.$$

Case 3 **(post–double-sourcing):** $u=(i,j+1), v=(i,j), w=(i-1,j)$.

In this configuration tree, $\hat{\mathcal{T}}(v)$ consists of only node v (because otherwise there would be a second double-sourcing node) and hence $t'=t$. Therefore, it is enough to deal with the cost for the amount of flow between nodes

Figure 8. (Color online) Possible Configurations of Three Consecutive Basis Nodes in a Downstream Basis Path

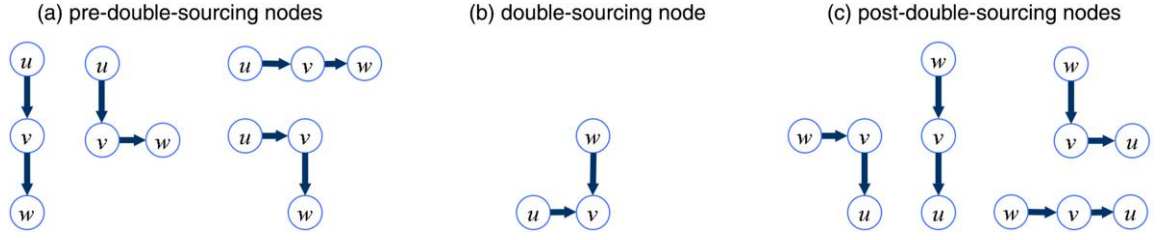
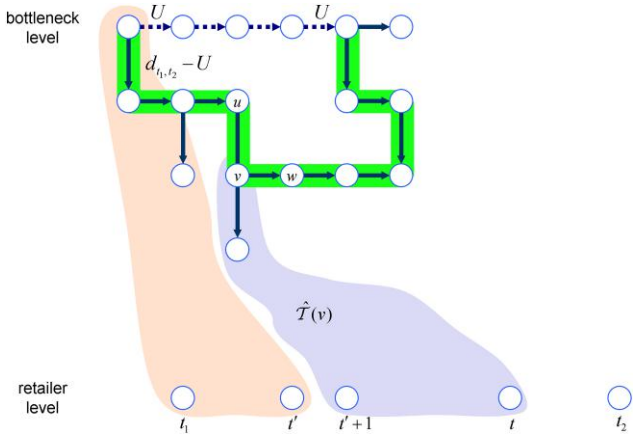


Figure 9. (Color online) Example of Immediate Cost in Pre–Double-Sourcing Case



v and u. Note that d_{t+1,t_2} of the U units available at the beginning of the closing node are allocated to satisfy future demands. Therefore, the remaining $U - d_{t+1,t_2}$ units flow into node u. Hence, the transition cost is

$$g_N(u,v,w)_{t',t} = h_{ij}(U - d_{t+1,t_2}).$$

An overview of the transition cost for the nine downstream configurations is given in Table 2. Assuming that the cost of the Zangwill trees have been precomputed, it follows from this table that a single transition cost $g_N(u,v,w)_{t',t}$ can be computed in constant time whenever needed.

4.3.4. DP Algorithm for Basis Paths and Downstream Networks. Instead of assuming a given basis path, we now determine the optimal basis path in a downstream network. Any path is implicitly specified once the predecessor (u, v) of each (v, w) arc in the path is given. As the orientation of the arcs in the path depends on the configuration, we use the term (u, v) arc to denote either the arc (u, v) or the arc (v, u). To develop the DP algorithm, we let $G_N(v,w)_t$ denote the minimal cost of state (N, v, w, t). Formally, we have the following definition.

Definition 2. The value $G_N(v,w)_t$ is the minimal cost of satisfying demands $d_{t_1}, \ldots, d_t$ in $N = (b_1, b_2, t_1, t_2)$ using a partial basis path from node (a, b_1) up till node v with the (v, w) arc being in the basis path.

4.3.4.1. Initialization. Suppose that v is the opening node of the downstream basis path, so $v = (a, b_1)$. Because spring node v implies immediate transportation, the second node w on the basis path is $w = (a + 1, b_1)$. No demands are satisfied from node u as this node is not defined (the arc (u, v) does not exist). Furthermore, we will take into account the inventory cost of the bottleneck nodes at the termination of the basis path, so they are not taken into account yet. As a result, the recursion is initialized for $v = (a, b_1)$ and $w = (a + 1, b_1)$ by

$$G_N(v,w)_t = \begin{cases} 0 & \text{if } t = t_1 - 1 \\ \infty & \text{if } t_1 \le t \le t_2. \end{cases} \tag{8}$$

Table 2. Transition Cost $g_N(u,v,w)_{t',t}$ for Downstream Basis Nodes

Type	Basis nodes u, v, w	Transition cost $g_N(u,v,w)_{t',t}$
Pre–double-sourcing	$(i,j-1),(i,j),(i+1,j)$	$h_{i,j-1}(d_{t+1,t_2} - U)$
	$(i-1,j),(i,j),(i+1,j)$	$p_{i,j}(d_{t+1,t_2} - U)$
	$(i-1,j),(i,j),(i,j+1)$	$p_{i,j}(d_{t'+1,t_2} - U) + p_{i+1,j}(d_{t'+1,t}) + \phi^{i+1,j}_{t'+1,t}$
	$(i,j-1),(i,j),(i,j+1)$	$h_{i,j-1}(d_{t'+1,t_2} - U) + p_{i+1,j}(d_{t'+1,t}) + \phi^{i+1,j}_{t'+1,t}$
Double-sourcing	$(i,j-1),(i,j),(i-1,j)$	$h_{i,j-1}(d_{t'+1,t_2} - U) + \phi^{i,j}_{t'+1,t}$
Post–double-sourcing	$(i+1,j),(i,j),(i-1,j)$	$p_{i+1,j}(U - d_{t'+1,t_2}) + h_{ij}(d_{t'+1,t}) + \phi^{i,j+1}_{t'+1,t}$
	$(i,j+1),(i,j),(i-1,j)$	$h_{ij}(U - d_{t+1,t_2})$
	$(i,j+1),(i,j),(i,j-1)$	$h_{ij}(U - d_{t+1,t_2})$
	$(i+1,j),(i,j),(i,j-1)$	$p_{i+1,j}(U - d_{t'+1,t_2}) + h_{ij}(d_{t'+1,t}) + \phi^{i,j+1}_{t'+1,t}$

4.3.4.2. Recursive Equation. Now suppose that v is not the opening node and w is not the closing node of the basis path. Furthermore, suppose that tree $\hat{\mathcal{T}}(v)$ covers demands $d_{t'+1},\dots,d_t$ (Figure 9). Let $B(v,w)$ be the set of nodes such that for $u \in B(v,w)$ the nodes u, v and w form a feasible configuration (Figure 8). By optimizing over the predecessor u and the period t' and by taking into account the transition cost, we have the following recursive equation:

$$G_N(v,w)_t = \min_{\substack{t_1 \le t' \le t \\ u \in B(v,w)}} \{G_N(u,v)_{t'} + g_N(u,v,w)_{t',t}\}, \tag{9}$$

with $g_N(u,v,w)_{t',t}$ as computed in Table 2.

4.3.4.3. Completion of the Downstream Network. To obtain the optimal cost of the downstream network N, we consider the case where $w=(a,b)$ is the closing node of the basis path. Assume that the basis nodes until $v=(a+1,b)$ satisfy demands $d_{t_1},\dots,d_t$. There are three types of cost that need to be taken into account: (i) the cost of carrying U units in inventory at the bottleneck level in periods nodes $b_1,\dots,b$, (ii) the cost of the flow on the (v,w) arc, and (iii) the cost of the bottleneck tree (if any).

For the first type of cost, we use the notation $h(j_1,j_2)$ to denote the inventory holding cost of keeping U units from period j_1 to j_2 at the bottleneck level, that is, $h(j_1,j_2)=\sum_{j=j_1}^{j_2-1} h_{a,j}(U)$. To evaluate the second type of cost, note that the flow on the (v,w) arc equals $U-d_{t+1,t_2}$. Finally, in case $b<b_2$, there must be a bottleneck tree $\mathcal{T}(u)$ satisfying demands $d_{t+1},\dots,d_{t_2}$ with a flow of d_{t+1,t_2} units on the connecting arc. This flow is less than the inventory capacity and hence $d_{t+1,t_2}<U$. Combining all the cost components, the cost $c_N(a,b_1)$ of downstream network N becomes

$$c_N(a,b_1) = \min_{\substack{b_1+1 \le b \le b_2 \\ t_1 \le t \le t_2 : d_{t+1,t_2} < U}} \left\{ G_N((a+1,b),(a,b))_t + h(b_1,b) + p_{a,b}(U-d_{t+1,t_2}) + h_{a,b}(d_{t+1,t_2}) + \psi_{t+1,t_2}^{b+1,b_2} \right\}. \tag{10}$$

Note that (10) is valid if the downstream network contains a double sourcing node. As mentioned before, if this is not the case the minimum cost is given by $\psi_{t_1,t_2}^{b_1,b_2}$. Hence, $c_N(a,b_1)$ should be taken equal to the minimum of the latter value and the right-hand side of (10). Recall that the cost $c_N(a,b_1)$ is used in the main DP of Section 4, and therefore this completes the DP approach.

4.3.4.4. Running Time. To determine the running time, there are $\mathcal{O}(LT^6)$ states (N,v,w,t) and hence values $G_N(v,w)_t$. Because $|B(v,w)| \le 3$, finding all values $G_N(v,w)_t$ by (9) takes $\mathcal{O}(LT^7)$ time at first sight. However, the cost of state (N,v,w,t) with $N=(b_1,b_2,t_1,t_2)$ is the same as the cost of state (N',v,w,t) with $N'=(b_1,b_2+1,t_1,t_2)$. In other words, when computing the optimal costs of the partial basis paths in N', we automatically obtain the optimal costs of all partial basis paths in N, which means that all values $G_N(v,w)_t$ in (9) can be computed in $\mathcal{O}(LT^6)$ time. Furthermore, because there are $\mathcal{O}(T^4)$ downstream networks N, the computation of (10) takes $\mathcal{O}(T^6)$ time. We conclude that the optimal costs of all downstream networks can be computed in $\mathcal{O}(LT^6)$ time.

4.4. Running Time Improvement

The bottleneck in the running time is the determination of all $c_N(i,j)$ by (3) in $\mathcal{O}(LT^7)$ time. In this section, we will show that this running time can be improved to $\mathcal{O}(LT^6)$. To do this, we introduce some extra notation. We define (i,j,N) as a minimum cost (i,j)-rooted upstream tree with underlying regeneration network $N=(b_1,b_2,t_1,t_2)$, so (i,j,N) is the upstream tree with cost equal to $c_N(i,j)$ (see Section 4). Now consider the set of upstream trees (i,j,N) where the period b_2 is not fixed but a free parameter, so $N=(b_1,b,t_1,t_2)$ with $b_1 \le b \le t_2$. Let b_2^* be an ending bottleneck period for which the cost of these upstream trees is minimal, that is,

$$b_2^* = \arg\min_{b_1 \le b \le t_2} \{c_N(i,j) : N=(b_1,b,t_1,t_2)\}.$$

For ease of notation, we do not show the dependency of b_2^* on the tuple (i,j,b_1,t_1,t_2). In a similar way we define

$$b_1^* = \arg\min_{i \le b \le \min\{b_2,t_1\}} \{c_N(i,j) : N=(b,b_2,t_1,t_2)\}.$$

From Theorem 1, it will follow that we can replace Recursion (3) by the following recursion:

$$c_N(i,j) = \min_{\substack{N'=(b_1,b_2^*,t_1,t),N''=(b_1^*,b_2,t+1,t_2): \\ t_1-1 \le t \le t_2}} \{p_{i+1,j}(d_{t_1,t}) + c_{N'}(i+1,j) + h_{i,j+1}(d_{t+1,t_2}) + c_{N''}(i,j+1)\}, \tag{11}$$

with b_2^* (respectively, b_1^*) depending on $(i+1,j,b_1,t_1,t))$ (respectively, $(i,j+1,b_2,t+1,t_2)$). At first sight, this recursion seems invalid as the upstream trees $(i+1,j,N')$ and $(i,j+1,N'')$ may have arcs in common at the bottleneck level (in case $b_2^* > b_1^*$) leading to an infeasible solution (the capacity constraint is violated). However, in the proof of Theorem 1, we show that such a solution can be transformed into a feasible solution without increasing the cost.

Theorem 1. *When computing the cost of the upstream trees in Recursion (3), the capacity constraint at the bottleneck level can be ignored under Assumption 1.*

Proof. We will prove that under Assumption 1, Recursion (3) can be replaced by (11). Let b^* and t^* be indices for which the minimum takes place in (3). That is, the upstream tree (i, j, N) with $N = (b_1, b_2, t_1, t_2)$ consists of the upstream tree $(i+1,j,N')$ with $N' = (b_1, b^*, t_1, t^*)$ and the upstream tree $(i,j+1,N'')$ with $N'' = (b^*+1, b_2, t^*+1, t_2)$ next to the connecting arcs from (i, j) to $(i+1,j)$ and (i, j) to $(i,j+1)$. Now consider the alternative tuples $\hat{N}' = (b_1, b_2^*, t_1, t^*)$ and $\hat{N}'' = (b_1^*+1, b_2, t^*+1, t_2)$ with b_2^* (respectively, b_1^*) depending on the tuple $(i+1,j,b_1,t_1,t^*)$ (respectively, $(i,j+1,b_2,t^*,t_2)$). By definition of b_1^* and b_2^*, we have $c_{\hat{N}'}(i+1,j) \le c_{N'}(i+1,j)$ and $c_{\hat{N}''}(i,j) \le c_{N''}(i,j)$. Therefore, the value of $c_N(i,j)$ obtained by (11) is not larger than the one obtained by Recursion (3). It is now sufficient to show that the flows of the upstream tree $(i+1,j,\hat{N}')$ and the upstream tree $(i,j+1,\hat{N}'')$ can be transformed into a feasible solution (if not feasible yet) without increasing the cost.

Let $\mathcal{S}_1$ (respectively, $\mathcal{S}_2$) be the arcs of upstream tree $(i+1,j,\hat{N}')$ (respectively, $(i,j+1,\hat{N}'')$) with positive flow (so including the saturated bottleneck arcs) and the connecting arc from (i, j) to $(i+1,j)$ (respectively, (i, j) to $(i,j+1)$). Furthermore, let x^1 (respectively, x^2) be the flows of $\mathcal{S}_1$ (respectively, $\mathcal{S}_2$) and $x = x^1 + x^2$ be the sum of the flows. Note that x is feasible if $b_2^* \le b_1^*$, because then $\mathcal{S}_1$ and $\mathcal{S}_2$ do not have any bottleneck arcs in common, which are the only arcs with capacities. Therefore, consider the case $b_2^* > b_1^*$ and let $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$ be the union of the upstream trees. As an example, Figure 10 illustrates the networks $\mathcal{S}_1$, $\mathcal{S}_2$ and $\mathcal{S}$.

We will show that x can be transformed into a feasible solution by reallocating flow of x^2 in two steps. First, consider the path $\mathcal{P}_1$ in $\mathcal{S}_1$ from (a,b_2^*) to (L,t^*). The arc from (a,b_2^*) to $(a+1,b_2^*)$ must be on the path (because $x_{a,b_2^*} > 0$) and such a path exists because demand in t^* is satisfied. In Figure 10(c), we have $\mathcal{P}_1 = \{(3,3),(4,3),(5,3)\}$. Furthermore, consider a path $\mathcal{P}_2$ in $\mathcal{S}_2$ from bottleneck node (a, b) to (L, t) for some $b < b_2^*$ and $t > t^*$, for which the arc from (a, b) to $(a+1,b)$ is on the path. There is at least one such path since $x^2_{a,b_1^*} > 0$ and an example in Figure 10(c) is the path $\{(3,2),(4,2),(4,3),(4,4),(5,4)\}$. By definition of the paths, $\mathcal{P}_1$ and $\mathcal{P}_2$ intersect and let (i',j') be the most upstream intersection node (node (4, 3) in Figure 10(c)). We now reallocate all flow on the subpath from (a, b) to (i',j') in $\mathcal{P}_2$ over the subpath from (a, b) to (i',j') in $\mathcal{P}_1$. Because of Assumption 1, the cost will not increase. After having done the reallocation for all paths $\mathcal{P}_2$, the modified flow, say $\hat{x}^2$, has the property that there is no flow $\hat{x}^2_{a+1,b} > 0$ for $b < b_2^*$ (Figure 11(a)).

Figure 10. (Color online) Subnetworks $\mathcal{S}_1$ (Left), $\mathcal{S}_2$ (Middle), and $\mathcal{S}$ (Right)

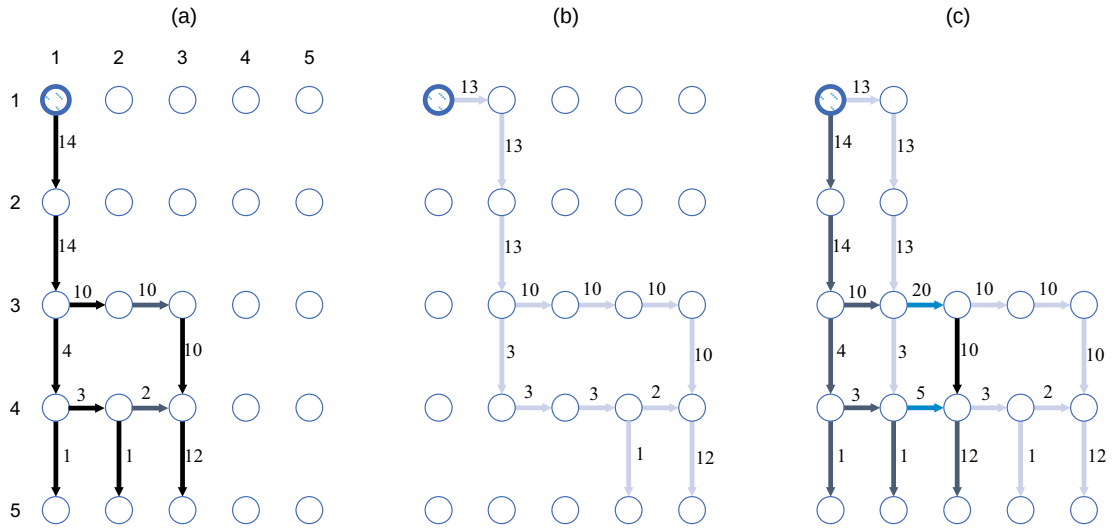
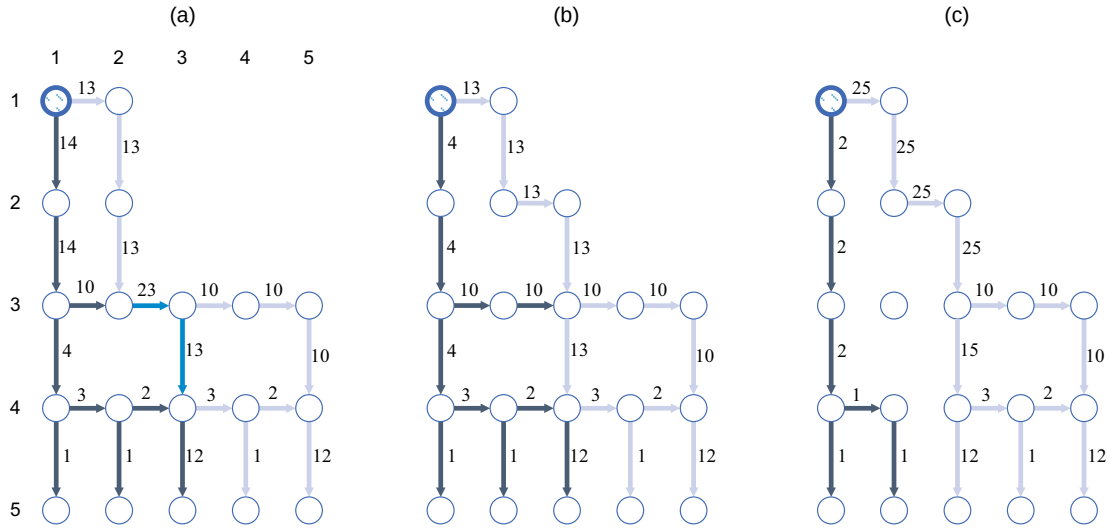


Figure 11. (Color online) Reallocations and Subnetworks $\mathcal{S}'$, $\mathcal{S}''$ and $\mathcal{S}'''$



In the second step, we reallocate all flows $\hat{x}^2_{a,b} > 0$ with $b < b^*_2$ and postpone it to period b^*_2. That is, the flow $\hat{x}^2_{a,b}$ over the path $\mathcal{Q}_1 = \{(a-1,b),(a,b),(a,b+1),\ldots,(a,b^*_2)\}$ is now sent over the path $\mathcal{Q}_2 = \{(a-1,b),(a-1,b+1),\ldots,(a-1,b^*_2),(a,b^*_2)\}$. Again, the cost will not increase because of Assumptions 1 and 2. After the reallocation of all such flows, the capacity constraint is not violated anymore (Figure 11(b)). In conclusion, by the two-step reallocation of the flows, we have found a feasible solution without increasing the costs. We could further remove cycles, as has been done in Figure 11(c), but this is not needed to get a feasible solution. □

We now analyze the running time of the alternative approach. First, the running time of (11) compared with (3) reduces from $\mathcal{O}(LT^7)$ to $\mathcal{O}(LT^6)$, because we minimize only over t (instead of over b and t). Second, we need to compute b^*_2 for each tuple (i,j,b_1,t_1,t_2), which takes $\mathcal{O}(LT^5)$ time in total. As the same holds for the computation of all b^*_1, the total running time for evaluating (3) becomes $\mathcal{O}(LT^6)$. Finally, we conclude that an optimal solution of MILSP can be found in $\mathcal{O}(LT^6)$ time because this is the highest running time over all recursions.

Theorem 2. *MILSP can be solved in $\mathcal{O}(LT^6)$ time.*

4.5. Special Cases

In some special cases, the running time can be reduced. In particular, we will show that this is possible when the bottleneck is at the last level and when there are only two levels and the bottleneck is at the first level. These special cases with corresponding running times are summarized in Table 3. Furthermore, there is a practical improvement in case of nonspeculative motives at the downstream level.

4.5.1. Bottleneck at Last Level. Because we have $a = L$, we know by Proposition 2 that the single-sourcing property holds at all nodes, that is, $I_{i,j-1}x_{ij} = 0$ for $i = 1,\ldots,L$ and $j = 1,\ldots,T$. Hence, each downstream network is a path, and we do not need a basis path approach (i.e., Equations (8)–(10)) to compute the cost of downstream networks. Moreover, it follows that $b_1 = t_1$ and $b_2 = t_2$ in downstream network $N = (b_1,b_2,t_1,t_2)$. This clearly reduces the time complexity considerably and it turns out that (3) is the bottleneck in terms of running time with a complexity of $\mathcal{O}(LT^4)$. This is the same as the running time for solving the uncapacitated version of the problem (Zangwill 1969).

4.5.2. Two Levels with Bottleneck at First Level. For each downstream network $N = (b_1,b_2,t_1,t_2)$, we know there is positive flow on the operational arc (u, v) with $u = (1,b_1)$ and $v = (2,b_1)$. This means that demand in period t_1 is

Table 3. Running Time of Some Special Cases

Case	Running time
$a = L$	$\mathcal{O}(LT^4)$
$a = 1$ and $L = 2$	$\mathcal{O}(T^5)$

Table 4. Generation of the Parameters

Parameter	d_t	K_{ij}	p_{ij}	H_{ij}	h_{ij}
Value	IU[1,20]	$\begin{cases}\overline{K} & \text{if } i=a\\ \text{U}[0,2\overline{K}] & \text{otherwise}\end{cases}$	1	0	$\begin{cases}0.01i & \text{if } i\le a\\ (i-a)/(L-a) & \text{otherwise}\end{cases}$

satisfied by this downstream network and hence $t_1 = b_1$. In turn, this means that we only need to consider downstream network $N = (b_1, b_2, b_1, t_2)$. Because a downstream network is now specified by the tuple (b_1, b_2, t_2), the total running time decreases by a factor T. Because L is a constant now, the resulting running time is $\mathcal{O}(T^5)$.

4.5.3. Nonspeculative Motives at the Downstream Level. In none of the propositions do we use the fact that there are nonspeculative motives at the downstream level. In fact, in this case one can show that the double-sourcing period can only be one level below the bottleneck level, that is, at level $a+1$. If it is further down, one can reroute the flow using the same operational arcs (hence occurring no additional setup cost) obtaining a solution with lower costs. This leads to a practical improvement as the basis paths can be restricted to the ones that have a double sourcing period at level $a+1$.

5. Computational Results

5.1. Design of Experiments

To test the suitability of our DP algorithm, we have conducted computational experiments. Table 4 shows how the demand and cost parameters are generated. The value $\overline{K}$ is considered as the base setup level and U$[a,b]$ (respectively, IU$[a,b]$) represents an (respectively, integer) uniform distribution on the interval $[a,b]$. Furthermore, the inventory capacity is set at $U = 10$. It turns out from some preliminary tests that under these parameter settings we frequently find instances for which the optimal solutions have double-sourcing periods, which complicates the problem and means no simpler DP algorithm can be applied such as the approach of Zangwill (1969).

We have tested multiple scenarios where each scenario differs in terms of time horizon, number of levels, and base setup level as shown in Table 5. We have performed a full combinatorial experiment except that we have only selected a few bottleneck levels a for each value of L. For each scenario, we have generated 10 instances, leading to 600 instances in total. Finally, all tests were performed on a laptop with 10 cores (Intel i7-1265U), 12 logical processors, and 32 GB RAM. The DP algorithm is implemented in C++ (code and instances are available on GitHub: github.com/wvdheuvel/SerialLotSizingWithBoundedInventory), whereas Cplex 12.10 is used as Mixed Integer Linear Programming (MIP) solver.

5.2. Results

The results of the experiments are shown in Table 6. We compare the running times of our DP algorithm with the runtime of the MIP solver Cplex 12.10 on the formulation of Section 3. We report the average and standard deviation over the 10 instances of each scenario. Furthermore, in the column DS, we report the number of instances (out of 10) that have at least one double-sourcing node implying that the inventory bound is active.

Table 6 shows that instances of 24 periods can be solved quickly by both methods: The DP needs at most 1 second, whereas the MIP needs at most 21 seconds on average. The instances of 32 periods can still be solved to optimality within the 1-hour time limit, but the MIP is considerable slower than the DP (more than a factor 30) in case of 12 levels. For $T = 40$ and $T = 48$, most of these instances cannot be solved to optimality anymore by the MIP (the superscript indicates the number not solved to optimality), whereas the DP takes about 2 minutes on average in the worst case. We have tested whether the MIP run times could be improved by using several types of cuts more aggressively, but this did not lead to a systematic improvement. (Also note that Cplex uses all 10 cores, whereas the DP runs on a single core.) Furthermore, we see that the MIP has a larger standard deviation than the DP, which is expected since the steps in the DP are much less sensitive to the parameter values of the instances. It interesting to see that the MIP approach is especially sensitive to the number of levels and not so much the number of periods, while the DP is sensitive to number of periods (as expected given the $\mathcal{O}(LT^6)$ time complexity). We can

Table 5. Generation of the Scenarios

Parameter	T	L	$\overline{K}$
Values	{24, 32, 40, 48}	{3, 6, 12}	{4, 8}

Table 6. Runtimes of the Algorithms (10 Instances for Each Scenario)

			$T = 24$				$T = 32$				$T = 40$				$T = 48$			
			MIP		DP		MIP		DP		MIP		DP		MIP		DP	
$\overline{K}$	L	a	DS	Average	Standard deviation	Average	Standard deviation	DS	Average	Standard deviation	Average	Standard deviation	DS	Average	Standard deviation	Average	Standard deviation	DS	Average	Standard deviation	Average	Standard deviation
4	3	2	5	0.1	0.0	0.43	0.02	8	0.1	0.0	1.8	0.5	10	0.2	0.1	6.5	1.1	10	0.4	0.2	36.0	9.3
	6	3	9	1.1	0.9	0.75	0.11	9	4.5	0.5	3.8	0.2	10	6.4	0.8	20.5	2.1	10	8.5	1.8	64.8	6.3
	6	5	9	0.6	0.6	0.56	0.05	10	2.7	1.2	3.6	0.1	10	5.4	1.2	20.2	1.2	10	9.3	3.6	72.2	3.2
	12	4	4	11.2	2.6	0.92	0.01	5	168.6	252.0	4.9	0.1	4	2,204[4]	1,439.3	24.3	0.4	6	3,248[8]	949.4	100.4	8.3
	12	7	7	20.7	21.6	0.89	0.07	8	207.5	250.6	5.1	0.6	8	3,422[8]	609.4	25.0	0.6	10	3,616[10]	3.7	123.7	2.5
	12	10	10	14.1	12.5	0.88	0.18	10	180.8	202.0	5.6	1.4	10	3,326[7]	459.5	16.9	5.2	10	3,613[10]	3.1	84.7	19.0
8	3	2	8	0.13	0.02	0.43	0.03	10	0.2	0.0	1.9	0.5	10	0.3	0.1	9.0	2.0	10	0.5	0.2	43.2	13.5
	6	3	8	2.16	1.40	0.83	0.10	7	5.7	0.7	4.3	0.5	9	9.0	1.5	26.6	4.9	10	12.0	2.5	80.8	11.8
	6	5	8	0.53	0.20	0.73	0.16	10	2.4	1.4	5.2	1.4	10	5.5	1.3	24.7	3.5	9	7.1	1.1	86.5	9.4
	12	4	5	11.44	2.83	0.95	0.01	6	173.2	278.3	5.5	1.2	7	2,900[7]	1,220.3	28.7	4.4	6	3,613[10]	2.8	118.1	11.7
	12	7	5	9.34	2.38	0.95	0.21	8	182.0	174.8	6.2	1.5	10	3,489[8]	367.7	27.6	3.8	9	3,298[9]	1,001.1	109.9	7.4
	12	10	10	6.29	2.21	0.82	0.02	10	114.2	161.4	5.1	0.7	10	1,736[3]	1,390.8	25.8	0.6	10	3,617[10]	2.0	111.9	4.8

conclude that, although the $\mathcal{O}(LT^6)$ time complexity of the DP seems high (but polynomial), we can still solve instances of reasonable size within a few minutes. For larger instances, the disadvantage of the DP is rather the memory use than the running time. For example, the instances of $T=48$ periods cannot be solved with 16 GB of memory but can with 32 GB.

6. Conclusion

In this paper, we deal with a supply chain problem with a bottleneck capacity on inventory at a single level, which is modeled as a serial multilevel lot-sizing problem with bounded inventory. Under the assumptions of nonspeculative motives and nonincreasing setup cost at the bottleneck level, we present a polynomial time algorithm for this problem independent of the level at which the bottleneck capacity occurs.

We provide a novel procedure combining the approach of Zangwill (1969) and the basis-path approach of Hwang et al. (2013). The effectiveness of the algorithm is verified by a set of computational experiments. It turns out that our algorithm is highly efficient compared with a state-of-the-art commercial solver. Our algorithm can solve problems until 48 periods and 12 levels in a few minutes, whereas the commercial solver cannot solve them to optimality in an hour, which proves the practical use of our algorithm.

As a direction for future research, one may relax some assumptions. For example, one may relax the assumption of nonspeculative motives at the upstream level, implying that Proposition 2 does not hold anymore. This means that the single sourcing property will not hold anymore at the bottleneck level, which clearly makes the problem harder to solve. If we relax too many assumptions, the problem turns NP-hard (Phouratsamay et al. 2018). Moreover, it would be interesting to investigate whether our approach could help in providing faster algorithms for other lot-sizing problems like the ones considered in He et al. (2015) and Ahmed et al. (2016) or be used in the development of (mathematical programming) heuristics for more complicated structures such as an assembly structure.

Acknowledgments

The authors thank three anonymous reviewers and an associate editor for providing constructive comments, which helped to improve the exposition of the paper, and Stéphane Dauzère-Pérès for motivating the problem of this paper.

Appendix A. Determining the Transition Costs

In this section, we present how to determine the transition costs in Table 2.

A.1. Case 1: Pre–Double-Sourcing Nodes

In this case, the amount flow from nodes u to v is $d_{t'+1,t_2} - U$ as demonstrated in Case 1 of Section 4.3.3.

A.1.1. Subcase 1.1. Let $u=(i,j-1)$, $v=(i,j)$, $w=(i+1,j)$. In this case, we see from the configuration of the nodes u, v, and w that no Zangwill tree is attached to node v, which means that $t'=t$. It suffices to consider the cost associated with the flow from u to v for the transition from $N(u,v,t')$ to $N(v,w,t)$. Hence, we have

$$g_N(u,v,w)_{t',t} = h_{i,j-1}(d_{t+1,t_2} - U).$$

A.1.2. Subcase 1.2. Let $u=(i-1,j)$, $v=(i,j)$, $w=(i+1,j)$. Similar to Subcase 1.1, we have no Zangwill tree attached to node v, and thus we have

$$g_N(u,v,w)_{t',t} = p_{i,j}(d_{t+1,t_2} - U).$$

A.1.3. Subcase 1.3. Let $u=(i-1,j)$, $v=(i,j)$, $w=(i,j+1)$. See Case 1 of Section 4.3.3.

A.1.4. Subcase 1.4. Let $u=(i,j-1)$, $v=(i,j)$, $w=(i,j+1)$. In this case, we may have a Zangwill tree satisfying demands $d_{t'+1,t}$. The cost of $\hat{\mathcal{T}}(v)$ consists of the connecting arc cost $p_{i+1,j}(d_{t'+1,t})$ and the Zangwill-tree cost $\phi^{i+1,j}_{t'+1,t}$. Combining this with the cost of the flow (u,v), we have

$$g_N(u,v,w)_{t',t} = h_{i,j-1}(d_{t'+1,t_2} - U) + p_{i+1,j}(d_{t'+1,t}) + \phi^{i+1,j}_{t'+1,t}.$$

A.2. Case 2: Double-Sourcing Node

See Case 2 of Section 4.2.3.

A.3. Case 3: Post–Double-Sourcing Nodes

In this case, the amount flow between nodes u and v is $U - d_{t'+1,t_2}$ as demonstrated in Case 3 of Section 4.3.3.

A.3.1. Subcase 3.1. Let $u=(i+1,j)$, $v=(i,j)$, $w=(i-1,j)$. From the configuration of u, v, and w, node v, we might have a Zangwill tree containing node $(i,j+1)$, satisfying demand $d_{t'+1,t}$. Hence, the cost $\hat{\mathcal{T}}(v)$ is $h_{i,j}(d_{t'+1,t})+\phi^{i,j+1}_{t'+1,t}$. Considering the cost $p_{i+1,j}(U-d_{t'+1,t_2})$ of the flow between u and v, we see that

$$g_N(u,v,w)_{t',t}=p_{i+1,j}(U-d_{t'+1,t_2})+h_{i,j}(d_{t'+1,t})+\phi^{i,j+1}_{t'+1,t}.$$

A.3.2. Subcase 3.2. Let $u=(i,j+1)$, $v=(i,j)$, $w=(i-1,j)$. See Case 3 of Section 4.3.3.

A.3.3. Subcase 3.3. Let $u=(i,j+1)$, $v=(i,j)$, $w=(i,j-1)$. Similar to Subcase 3.2, $\hat{\mathcal{T}}(v)$ contains only v and thus $t'=t$. It suffices to consider the cost on the flow between u and v. Hence,

$$g_N(u,v,w)_{t',t}=h_{i,j}(U-d_{t+1,t_2}).$$

A.3.4. Subcase 3.4. Let $u=(i+1,j)$, $v=(i,j)$, $w=(i,j-1)$. By configuration, we can see that the transition cost is the same as that of Subcase 3.1, that is,

$$g_N(u,v,w)_{t',t}=p_{i+1,j}(U-d_{t'+1,t_2})+h_{i,j}(d_{t'+1,t})+\phi^{i,j+1}_{t'+1,t}.$$

References

Aggarwal A, Park JK (1993) Improved algorithms for economic lot-size problems. *Oper. Res.* 41(3):549–571.
Ahmed S, He Q, Li S, Nemhauser GL (2016) On the computational complexity of minimum-concave-cost flow in a two-dimensional grid. *SIAM J. Optim.* 26:2059–2079.
Ahuja RK, Magnanti TL, Orlin JB (1993) *Network Flows: Theory Algorithms, and Applications* (Prentice Hall, Englewood Cliffs, NJ).
Atamtürk A, Küçükyavuz S (2005) Lot sizing with inventory bounds and fixed costs: Polyhedral study and computation. *Oper. Res.* 53:711–730.
Atamtürk A, Küçükyavuz S (2008) An $O(n^2)$ algorithm for lot sizing with inventory bounds and fixed costs. *Oper. Res. Lett.* 36(3):297–299.
Bitran GR, Yanasse HH (1982) Computational complexity of the capacitated lot size problem. *Management Sci.* 28:1174–1186.
Chung CS, Lin CHM (1988) An O(T^2) algorithm for the NI/G/NI/ND capacitated lot size problem. *Management Sci.* 34:420–426.
Dauzère-Pérès S, Nordli A, Olstad A, Haugen K, Koester U, Per Olav M, Teistklub G, et al. (2007) Omya Hustadmarmor optimizes its supply chain for delivering calcium carbonate slurry to European paper manufacturers. *Interfaces* 37(1):39–51.
Federgruen A, Tzur M (1991) A simple forward algorithm to solve general dynamic lot sizing models with n periods in $\mathcal{O}(n\log n)$ or $\mathcal{O}(n)$ time. *Management Sci.* 37:909–925.
Florian M, Klein M (1971) Deterministic procurement planning with concave costs and capacity constraints. *Management Sci.* 26:669–679.
He Q, Ahmed S, Nemhauser GL (2015) Minimum concave cost flow over a grid network. *Math. Programming* 150:79–98.
Hwang HC, van den Heuvel W (2012) Improved algorithms for lot-sizing problem with bounded inventory and backlogging. *Naval Res. Logist.* 59:244–253.
Hwang HC, Ahn H, Kaminsky P (2013) Basis paths and a polynomial algorithm for the multi-level production-capacitated lot-sizing problem. *Oper. Res.* 61(2):469–482.
Kaminsky P, Simchi-Levi D (2003) Production and distribution lot sizing in a two stage supply chain. *IIE Trans.* 35:1065–1075.
Lee CY, Çetinkaya S, Jaruphongsa W (2003) A dynamic model for inventory lot sizing and outbound shipment scheduling at a third-party warehouse. *Oper. Res.* 51:735–747.
Levi R, Yedidsion L (2013) Np-hardness proof for the assembly problem with stationary setup and additive holding costs. *Oper. Res. Lett.* 41(2):134–137.
Love SF (1972) A facilities in series inventory model with nested schedules. *Management Sci.* 18:327–338.
Love SF (1973) Bounded production and inventory models with piecewise concave costs. *Management Sci.* 20:313–318.
Melo RA, Wolsey LA (2010) Uncapacitated two-level lot-sizing. *Oper. Res. Lett.* 38:241–245.
Phouratsamay SL, Kedad-Sidhoum S, Pascual F (2018) Two-level lot-sizing with inventory bounds. *Discrete Optim.* 30:1–19.
Sargut FZ, Romeijn HE (2007) Lot-sizing with non-stationary cumulative capacities. *Oper. Res. Lett.* 35(4):549–557.
Silver EA, Pyke DF, Peterson R (1998) *Inventory Management and Production Planning and Scheduling*, vol. 3 (Wiley, New York).
van den Heuvel W, Wagelmans APM (2008) A holding cost bound for the economic lot-sizing problem with time-invariant cost parameters. Technical report EI 2008-10, Econometric Institute, Erasmus University, Rotterdam, Netherlands.
van Hoesel CPM, Wagelmans APM (1996) An O(T^3) algorithm for the economic lot-sizing problem with constant capacities. *Management Sci.* 42:142–150.
van Hoesel CPM, Romeijn HE, Morales DR, Wagelmans APM (2005) Integrated lot sizing in serial supply chains with production capacities. *Management Sci.* 51(11):1706–1719.
Van Vyve M (2007) Algorithms for single-item lot-sizing problems with constant batch size. *Math. Oper. Res.* 32:594–613.
Wagelmans APM, van Hoesel CPM, Kolen A (1992) Economic lot sizing: An O($n\log n$) algorithm that runs in linear time in the Wagner-Whitin case. *Oper. Res.* 40:S145–S156.
Wagner HM, Whitin TM (1958) Dynamic version of the economic lot size model. *Management Sci.* 5:89–96.
Wolsey LA (2006) Lot-sizing with production and delivery time windows. *Math. Programming Ser. A* 107:471–489.
Zangwill WI (1969) A backlogging model and multi-echelon model of a dynamic economic lot size production system: A network approach. *Management Sci.* 15:506–527.
Zhao M, Zhang M (2020) Multiechelon lot sizing: New complexities and inequalities. *Oper. Res.* 68(2):534–551.