Eötvös Loránd University of Science
Department of Operational Research

Inventory optimization with guaranteed service time

Mátyás Korom

Supervisor: Alpár Jüttner
Department of Operational Research
Eötvös Loránd University of Science

Contents

Chapter 1

Introduction

Inventory management plays a fundamental role in the life of manufacturing organizations. Not only does its efficiency highly affect the profitability, but also determines quality of service and the overall adaptability of the whole company.

There are three main purposes why a company holds an inventory[2]. The first one is that materials has to be transported between different locations of the supply chain. Therefore, every stage has to hold certain amount of inventory to cover its needs until the next transport arrives. The second one is to deal with the uncertainty of the external demand. Finally, maintaining an inventory may cut down costs in terms of logistic.

An inventory can either be a single entity or — in case of large production and distribution organizations — a complex multi-stage and multi-location system. Operation of a single stage inventory is already a difficult but well studied topic, interested readers are referred to e.g. [1] for a detailed introduction to known approaches. Optimal running of multi-stage is even more difficult.

There are two basic approaches to minimize the costs in a multi-stage inventory. The first one tries to plan an optimal supply network and reduce the cost of manufacturing by placing the factories and warehouses at the right places. The second approach applies various inventory holding policies to minimize the costs for an already given network. In fact, these are complementary approaches. Obviously, the first one may result in higher savings for a green field supply network establishment. However, in many cases there is an already established supply chain or there may be external constraints determining the main structure and parameters of the network. Then, the second approach has a significant role to play in reducing the operational cost of a supply chain.

A prominent work by Stephen C. Graves and Sean P. Willems in 1998 [3] (later revised in 2005 [4]) introduces a flexible model to the second approach, i.e.

it assumes an already existing supply network and strictly bounded demands, then computes the optimal internal service times minimizing the overall operational cost of the full system. They and later S. Humair and Willems[13] gave efficient algorithms for finding the optimal solutions for special supply network, while E. Lesnaia, I. Vasilescu, and Graves[14] proved that the problem is NP-hard for general directed acyclic graphs. On the positive side, T. L. Magnanti et al.[16] and Graves and Lesnaia[17] gave practically efficient approximating solutions for this general case. A further generalization assuming capacity constraints is discussed by T. Schoenmeyr and Graves[19].

This thesis gives an overview of the works above and — as new results — discusses the analogous questions results in the case when non-uniform service times are allowed. Finally, the appendix shortly summarizes notations most frequently used throughout the thesis.

Acknowledgment

I would like to acknowledge my supervisor Alpár Jüttner for introducing me to the theory of safety stocks and guided me through the writing of the thesis. I am also grateful for the support of IBM-ELTE Center of Applied Mathematics during my work on this topic.

Chapter 2

Supply chains with guaranteed service times

This chapter introduces the basic model and the different algorithms developed by Stephen C. Graves, Sean P. Willems[3] and others.

2.1 The model and assumptions

The supply chain is modelled as a directed acyclic graph, where a node represent a stage at the chain and an edge induce a customer - supplier relationship. A stage is a part of the chain where some relevant process happens. Practically that can almost be anything including assembly, distribution or row material pre-processing. We assume that external demand occurs only at nodes have zero outdegree. We call this kind of nodes demand nodes.

We are given a scalar ϕ_{uv} associated to every arc (uv), which means node v needs ϕ_{uv} unit of component from node u to produce its own product. Another two scalars are associated to nodes. The first is T_u, which denotes production time. If node u got all the subcomponents from its suppliers, it takes T_u time to manufacture its product. The second is h_u, which is the holding cost per unit at node u.

So far, we have a directed acyclic graph with a scalar field ϕ_{uv} on the arcs, and two scalar fields, namely T_u and h_u on the nodes. Let us introduce the basic assumptions.

- Bounded External Demand: We assume there are given a $D_u(\tau)$ function, which is the upper bound for demand for any τ long period for every demand node u. So a demand sequence of length τ, namely $d_u(t)$, $d_u(t-1)$,

$\dots d_u(t-\tau+1)$ at node u is valid if and only if

$$d_u(t) + d_u(t-1) + \dots d_u(t-\tau+1) \leq D_u(\tau) \tag{2.1}$$

holds.

- Guaranteed Service Time: One of the most important assumptions. Each node (both demand nodes and internal nodes) quotes a services time (outbound service time) S_u to their customers, and guarantees it will fulfil every valid demand occurred at time t at time $t + S_u$. The task is to determine these outbound service times. In this chapter, we assume each node guarantees the same service time to all of its customers.

- Maximized service time for demand nodes: There is a given $s_u \geq 0$, which maximizes the service time of node u for every demand node.

- Periodic-Review Base-Stock Replenishment Policy: The last assumption says all of the stages operate with periodic-review policy with a common review period. The stages also places the same demand (multiplied by the scalar ϕ_{uv}) on their suppliers than they observes at the period. So if an external demand occurs at a demand node u at time t all of the supplier sub-chain see the demand.

The first assumption do not say anything about internal demands, but according to the last one

$$d_u(t) = \sum_{(uv)\in A} (\phi_{uv} \cdot d_v(t)), \tag{2.2}$$

therefore

$$D_u(\tau) = \sum_{(uv)\in A} (\phi_{uv} \cdot D_v(\tau)) \tag{2.3}$$

is a demand bound for internal node u.

Remark Graves and Willems also assume $D_u(\tau)$ to be concave and monotonically increasing with τ, but some of the following algorithms work perfectly well without these assumptions. However some of them uses them, therefore we will note every time these are relevant.

Observation If the demand bounds of demand nodes are concave and monotonically increasing, $D_u(\tau)$ is concave and monotonically increasing with τ for all nodes, because it is the sum of some concave and monotonically increasing functions.

2.1.1 Single stage model

This and the next subsection shows how to build the model step by step.

Consider a single stage (node u) with some inbound and some outbound arcs, and also consider this stage quotes service time S_u to its customers. Let

$$IS_u = \max_{(vu)\in E}\{S_v\}$$

be the inbound service time of node u. If the stage observes demand $d_u(t)$, it also places the equivalent demand on its suppliers at time t. So it will get all the parts necessary to begin the production at time $t + IS_u$, thus complete the production at time $t + IS_u + T_u$. Meanwhile it needs to fulfil demand $d_u(t)$ at time $t + S_u$. So if $IS_u + T_u > S_u$, stage u needs to store certain amount of already processed stock to be able to satisfy the guaranteed service time it has quoted.

Let $I_u(t)$ denote the inventory held at sage u at time t. Because orders observed at time $(t - S_u)$ need to be fulfilled while orders placed on suppliers at time $(t - IS_u)$ arrived and are available T_u periods later, this value can easily be determined by the following recursive form.

$$I_u(t) = I_u(t-1) - d_u(t - S_u) + d_u(t - IS_u - T_u) \tag{2.4}$$

By definition $I_u(t) \geq 0$ at any time. Using Equation (2.4) recursively

$$0 \leq I_u(t) = I_u(0) - \sum_{i=t-IS_u-T_u}^{t-S_u} d_u(i) \tag{2.5}$$

must hold for every valid demand sequence $d_u(1)$, $d_u(2)$, ..., $d_u(t - S_u)$. Rearranging Inequality (2.5) and using the fact

$$\sum_{i=t-IS_u-T_u}^{t-S_u} d_u(i) \leq D_u(IS_u + T_u - S_u) \qquad \forall t$$

we get the following

$$I_u(0) \geq D_u(IS_u + T_u - S_u). \tag{2.6}$$

This is a necessary and sufficient condition to satisfy the guaranteed service time model.

2.1.2 Multi stage model

At this general case we are given a directed acyclic graph $D = (V, A)$ with node set V and arc set A. If we associate a service time S_u to every node and define

IS_u as the single stage model, then using Inequality (2.6), we can calculate the minimal inventory that is sufficient to satisfy the assumptions.

$$I_u(0) \geq D_u(IS_u + T_u - S_u) \qquad \forall u \in V$$

As h_u is the holding cost per unit, the minimal inventory value held at node u is

$$\sum_{u \in V} (h_u \cdot D_u(IS_u + T_u - S_u)) \tag{2.7}$$

Now, we can formulate the optimization problem as a NLP.

Problem 1.

$$\min \sum_{u \in V} (h_u \cdot D_u(IS_u + T_u - S_u))$$

$s.t.:$

$$IS_u + T_u \geq S_u \qquad \forall u \in V$$
$$IS_u \geq S_v \qquad \forall v \in V\colon (vu) \in A$$
$$S_u \in \mathbb{Z},\ \geq 0 \qquad \forall u \in V$$
$$S_u \leq s_u \qquad \forall u \in V,\ \textit{demand node}$$

Observation One may assume $IS = 0$ for all nodes with zero indegree.

Observation Nodes with zero indegree can not quote higher service times to their customers than their production times. Since for these kind of nodes $IS = 0$, they have to hold $D_u(T_u - S_u)$ processed stock, which is zero if $T_u = S_u$, and is out of range $D_v(\tau)$ if $T_u < S_u$. Along this line we can define a maximal possible service time for every node as follows. Let $M_u = T_u$ for every node with zero indegree, and

$$M_u = \max_{(vu) \in A} (M_v) + T_u. \tag{2.8}$$

Repeating the previous argument no node can quote a service time, which is higher than M_u to its customers in an optimal solution to Problem 1. This observation will be used in the next section.

If we suppose the demand bound functions to be concave, Problem 1 is a concave optimization problem over linear constraints which is known to be NP-hard in the general case. Graves and Willems developed a polynomial algorithm for trees[3], Humair and Willems introduced a little bit more general algorithm for graphs with Clusters of Commonality[13] which is exponential in terms of the number of vertices in a cluster, while Lesnaia, Vasilescu and Graves proved the general problem is NP-hard[14]. The next sections of this chapter introduce these results, while the next chapter introduces two approximating algorithms developed by Magnanti, Zuo-Jun Max Shen, Jia Shu, David Simchi-Levi, Chung-Piaw Teo [16] and Graves and Lesnaia[17].

Figure 2.1: Labelling algorithm for trees.

$i = 1$
while $V \neq \emptyset$ **do**
  Label an arbitrary leaf node with i, and remove it from graph D.
  $i := i + 1$.
end while

2.2 Algorithm for trees

Graves and Willems developed a dynamic programming method to solve Problem 1 if the underlying directed graph D is a tree. First they label the nodes with integers such a way that all nodes has at most one neighbour with higher label. An easy way to do that to use the algorithm presented in Figure 2.1.

Definition Such a labelling of nodes is called *leaf ordering*.

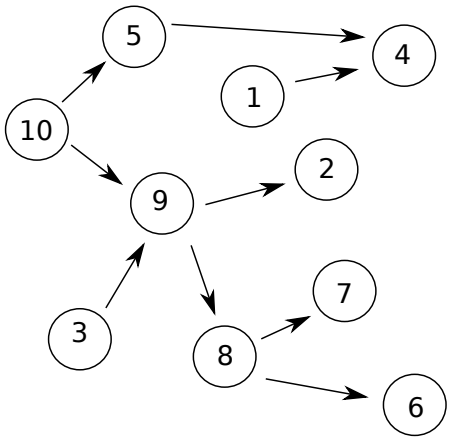

Figure 2.2: A proper leaf ordering of a tree

This leaf ordering is used to decompose the problem into sub-problems in the following way.

Let $|V| = N$, and $l(v)$ be the label of node v. Thus $1 \leq l(v) \leq N \ \forall v \in V$. Call node v the parent of node u if $l(v) > l(u)$ and $(uv) \in A$ or $(vu) \in A$. Similarly, v is child of u if $l(v) < l(u)$ and $(uv) \in A$ or $(vu) \in A$. So every node apart from the one with $l(v) = N$ has exactly one parent but can have several children. Denote the set of children of u by $ch(u)$ and the parent by $p(u)$. Define sub-graph N_v of D the following way:

- $V(N_v) := \{v\} \cup_u V(N_u)$, where u is a child of v.

- $A(N_v) := I_G(V(N_v))$.

Definition $f_u(x)$ is the optimal inventory holding cost for N_u supposing $S_u \leq x$.

Similarly

Definition $g_u(x)$ is the optimal inventory holding cost for N_u supposing $IS_u \geq x$.

Let $c_v(S_v, IS_v)$ be the following.

$$c_v(S_v, IS_v) = h_v \cdot D_v(IS_v + T_v - S_v) + \sum_{\substack{u \in ch(v) \\ (vu) \in A}} g_u(S_v) + \\ + \sum_{\substack{w \in ch(v) \\ (wv) \in A}} f_w(IS_v) \quad (2.9)$$

It is easy to see $c_v(S_v, IS_v)$ is the optimal inventory holding cost for N_v supposing the inbound service time of v is IS_v and outbound service time is S_v.

Lemma 1. *If* $(p(v)v) \in A$, *then*

$$g_v(x) = \min_{x \leq IS_v \leq M_v - T_v} \min_{0 \leq S_v \leq M_v} c_v(S_v, IS_v) \quad (2.10)$$

If $(vp(v)) \in A$, *then*

$$f_v(x) = \min_{0 \leq S_v \leq x} \min_{0 \leq IS_v \leq M_v - T_v} c_v(S_v, IS_v) \quad (2.11)$$

Proof. It is straightforward from the definition of $c_v(S_v, IS_v)$. □

With Lemma 1, one can calculate the function f or g for every node one-by-one according to the leaf ordering.

Since there exists no node guaranteeing service time higher than its M, and gets service time higher than $M - T$ it is enough to minimize the expressions over the $[0, M_v]$ interval. Hence the minimization can be done in $(M_v - T_v - x) \cdot (M_v + 1)$ steps by simply computing all the possible values. Since $M_v \leq \max_{v \in V}(T_v) \cdot |V|$ the enumeration of $g_v(x)$ takes $O(|V|^2)$ steps. The same is true for $f_v(x)$ as well.

Figure 2.3 summarizes the algorithm. The variable *opt* contains the optimal solution to Problem 1 at the end of the algorithm.

Figure 2.3: The dynamic programming method for trees

for $i = 1 \to N - 1$ **do**

if $l(v) = i$ **and** $(p(v)v) \in A$ **then**
calculate $g_v(x)$ for $x = 0,\ 1 \ldots (M_v - T_v)$
end if

if $l(v) = i$ **and** $(vp(v)) \in A$ **then**
calculate $f_v(x)$ for $x = 0,\ 1 \ldots M_v$
end if

end for

$$opt := \min_{\substack{0 \le IS_v \le M_v - T_v \\ 0 \le S_v \le M_v}} (c_v(IS_v, S_v))$$

2.3 Algorithm for graphs with Clusters of Commonality

The tree structure is often insufficient to accurately model more complex supply chains. This chapter relaxes the tree assumption and introduce an algorithm, that optimizes the safety stock in more complex networks, namely networks with Clusters of Commonality.

Theorem 1. *The vertex set of a graph $G = (V, E)$ has a unique partition V_1, $V_2 \ldots$, V_k such that for every $i \in \{1, 2 \ldots, k\}$ the induced graph of V_i is 2-edge-connected. Furthermore if the vertexes in these sets are pinched the resultant graph is a tree.* □

Definition The $D = (V, A)$ directed graph is a *graph with Clusters of Commonality (CoC)* if its 2-edge-connected components are directed bipartite graphs, and all the edges inside a 2-edge-connected component are directed towards the same partition.

Remark Humair and Willems define the Clusters of Commonality in a somewhat different but more complicated way.

Observation A graph with Clusters of Commonality is acyclic. □

The algorithm developed by Humair and Willems[13] for solving Problem 1 for graphs with Clusters of Commonality is an extension of the one introduced in the previous section. Similarly to the tree case this also has two main phase

with an additional phase 0 which verify if the graph is a CoC network. The first one labels the nodes, the second calculates the optimal safety stock for different sub-graphs at the order of labels.

Phase 0 - verifying if the network is CoC:

- Start a depth first search with arbitrary root node.
- If a circle is found contract it. (These nodes belong to the same 2-edge-connected component.)

With appropriate data structure the depth-first search does not need to be restarted after a circle is contracted, so the identification of the 2 edge-connected components can be done in linear time.

- Check if the 2-edge-connected components are bipartite.
- Check if every 2-edge-connected component has one node partition with $\rho(v) = 0$ in the induced graph.

Phase 1 - labelling the nodes:

- Contract every 2-edge-connected component. (This results a tree: D'.)
- Number the nodes in the graph D' the same way as in the tree case.

Suppose a 2-edge-connected component has label k and n nodes.

- Add $n-1$ to every label which is greater than k.
- Assign number $k+n-1$ to the node adjacent to node (or component) labelled with $k+n$ (originally $k+1$). We call this node the root of the component.
- Assign arbitrary numbers from interval $[k, k+n-2]$ to other nodes in the component.

Observation For every cluster the cluster root has the greatest label.

Let us denote the clusters by $V_1, V_2 \dots V_q$, the head of cluster V_j by V_j^+, the source by V_j^- and define the children function:

- For cluster root v from cluster V_j

$$ch(v) = \{u : \ (uv) \text{ or } (vu) \in A \text{ and } l(u) < l(v)\} \cup V_j$$

- For a non cluster root w in cluster V_j :

$$ch(w) \;=\; \{u \;:\; (uw) \text{ or } (wu) \;\in\; A,\, l(u) \;<\; l(v) \text{ and } u \;\notin\; V_j\}$$

Phase 2 - calculate optimal safety stock:

- If node v is not a cluster root, let $c_v(S_v, IS_v)$ be

$$c_v(S_v, IS_v) = D_v(IS_v + T_v - S_v) + \\ + \sum_{\substack{u \in ch(v) \\ (vu) \in A}} g_u(S_v) + \sum_{\substack{w \in ch(v) \\ (wv) \in A}} f_w(IS_v) \quad (2.12)$$

and evaluate $f_v(x)$ or $g_v(x)$ as in the previous section.

- If node v is a cluster root of V_j we have to keep an eye on the constraints belong to the arcs in the cluster (the f and g functions of the children are not independent), therefore suppose $v \in V_j^-$ and let us define

$$h_v^-(S_v, IS_v) := \min(\sum_{u \in V_j^-} f_u(x_u) + \sum_{w \in V_j^+} g_w(y_w)) \quad (2.13)$$

s.t.:

$$y_w \geq S_v \qquad \forall (vw) \in A,\, v, w \in V_j$$
$$y_w \geq x_u \qquad \forall (uw) \in A,\, u, w \in V_j$$

and

$$c_v(S_v, IS_v) := D_v(IS_v + T_v - S_v) + \sum_{\substack{u \in ch(v) \setminus V_j \\ (vu) \in A}} g_u(S_v) + \\ + \sum_{\substack{w \in ch(v) \setminus V_j \\ (wv) \in A}} f_w(IS_v) + h_v^-(S_v, IS_v). \quad (2.14)$$

If $v \in V_j^+$ let us define

$$h_v^+(S_v, IS_v) := \min(\sum_{u \in V_j^-} f_u(x_u) + \sum_{w \in V_j^+} g_w(y_w)) \tag{2.15}$$

s.t.:

$$IS_v \geq x_u \qquad \forall (uv) \in A,\ u, v \in V_j$$
$$y_w \geq x_u \qquad \forall (uw) \in A,\ u, w \in V_j$$

and

$$c_v(S_v, IS_v) := D_v(IS_v + T_v - S_v) + \sum_{\substack{u \in ch(v) \setminus V_j \\ (vu) \in A}} g_u(S_v) + \\ + \sum_{\substack{w \in ch(v) \setminus V_j \\ (wv) \in A}} f_w(IS_v) + h_v^+(S_v, IS_v). \tag{2.16}$$

After the c_v function is computed the computation of the desired f_v or g_v function can be done in linear time.

Unfortunately the computation of h_v^- or h_v^+ and therefore the computation of c_v is exponential in the terms of the size of the clusters. Note that Humair and Willems introduced two enumeration schemes[13], which improve the running time, but those algorithms are still exponential.

2.4 The complexity of the general case

This section proves that Problem 1 is NP-hard. The section is a clarified version of the original paper of Ekaterina Lesnaia, Iuliu Vasilescu and Stephen C. Graves [14]. The key idea is to convert the vertex cover problem into Problem 1.

Problem 2. *We are given a simple undirected $G = (V, E)$ graph. Search for a set of vertexes S which satisfy the following to statement:*

1. $\forall (uv) \in E : u \in S$ *or* $v \in S$
2. *If some $\widetilde{S}$ satisfies statement 1, then $|S| \leq |\widetilde{S}|$*

This problem is called the vertex cover problem.

This problem is known to be NP-complete[15], so, if for every simple undirected graph this problem is convertible into a minimal safety stock problem, that proofs Problem 1 is NP-hard.

Construct a minimal safety stock problem:

1. Consider graph $G = (V, E)$. $|V| = N$.
2. Make all of the edges directed acyclically.
3. Add a new node P and new arcs (vP) for all $v \in V$ if v is not a demand node.
4. Call the new graph $D = (V \cup \{P\}, A)$ and the set of demand nodes in D V_D
5. Let the demand bound function be

$$D_v(\tau) = \begin{cases} 0, & \text{if } \tau = 0 \\ 1, & \text{if } \tau \geq 1 \end{cases}$$

for all nodes in V_D. It is easy to see that all demand bound functions have the form:

$$D_v(\tau) = \begin{cases} 0, & \text{if } \tau = 0 \\ k_v, & \text{if } \tau \geq 1 \end{cases}$$

for some k_v.

6. Set the per unit hold cost scalar

$$h_v = \begin{cases} \frac{1}{k_v}, & \text{if } v \neq P \\ N + 1, & \text{if } v = P \end{cases}$$

7. Set the production times

$$T_v = \begin{cases} 1, & \text{if } v \neq P \\ 0, & \text{if } v = P \end{cases}$$

8. Set the upper bound of service time for demand nodes equal to 1.

Note that the largest safety stock value that a node can hold is equal to 1 if the node is not equal to P. Node P can hold safety stock for zero or $N + 1$. It is easy to give a feasible solution to the problem. For example, let $S_v = 1$ for all $v \in V \cup \{P\}$.

Lemma 2. *$S_u \leq 1$ holds for every non-demand node u in D for all optimal solution.*

Proof. Consider a non-demand node v quotes service time greater than 1, namely $S_v \geq 2$. Hence $(vP) \in A$, $IS_P \geq 2$. The amount of safety stock node P have to hold is $D_P(IS_P + T_P - S_P) \geq D_P(2 + 0 - 1) = 1$. So the value of the safety stock held at node P is $N + 1$.

On the other hand, if every non-demand node quotes service time less than 2, $IS_v \leq 1$ and $h_v \cdot D_v(IS_v + T_v - S_v) \leq h_v \cdot D_v(1) = 1$ for each node $v \neq P$, while $h_P \cdot D_P(IS_P + T_P - S_P) \leq D_P(0) = 0$ holds for node P. The total value of safety stock in this case is at most N. □

Lemma 3. *If $(uv) \in A$, at least one in the two nodes holds safety stock in every optimal solution.*

Proof. By contradiction, suppose there is a node pair u and v such that $(uv) \in A$ and none of them holds safety stock. If node u does not hold safety stock, then $h_u \cdot D_u(IS_u + T_u - S_u) = 0$. Therefore $D_u(IS_u + T_u - S_u) = 0$, which is true only if $IS_u + T_u - S_u = 0$. The same holds for node v. Hence

$$\begin{aligned} &T_u = T_v = 1, \\ &0 \leq IS_u,\ IS_v,\ S_u,\ S_v \leq 1, \end{aligned}$$

$$\begin{aligned} &S_u,\ S_v = 1, \\ &IS_u,\ IS_v = 0 \end{aligned}$$

must hold.

But since $(uv) \in A$ $IS_u \geq S_v$, which is a contradiction. □

From Lemma 3 it follows if we solve the optimal safety stock problem for D, and choose the nodes which hold safety stock, we get a vertex cover for graph G. Furthermore, the cardinality of such a vertex covering set is equal to the optimal safety stock value. To complete the proof, we only need to verify that we can assign a solution to Problem 1 with safety stock value equal to the cardinality of the vertex covering set to every vertex cover problem.

Lemma 4. *If $S \subseteq V$ is a vertex covering set, there exist a solution $\{S_v : v \in V\}$ to Problem 1 where each node v holds safety stock if and only if $v \in S$.*

Proof. Set $S_v = 0$, $IS_v = 1$ for all $v \in S$, $S_P = 1$, $IS_P = 1$ for node P and $S_v = 1$, $IS_v = 0$ for all $v \notin S \cup \{P\}$.

Check if this solution is feasible:

- $S_v \leq 1$ for all demand nodes.
- No service time is greater than 1 so $IS_v \geq S_{\text{neighbour of } v}$ holds for $v \in S$ or $v = P$.

- If $v \notin S \cup \{P\}$, all of the neighbours of v are in S therefore all of its neighbours service time is equal to 0, so $IS_v \geq S_{\text{neighbour of } v}$ holds for $v \in S$

Hence $IS_v + T_v - S_v > 0$ if and only if $v \in S$ the total safety stock value is equal to $|S|$. □

Lemma 3 and Lemma 4 together proof the safety stock problem is NP-hard.

Chapter 3

Two approximating algorithms

After proving the optimal safety stock problem is NP-hard a natural question arises. What should we do if we have to optimize inventory for a company. A lot of algorithms were developed that solves NP-hard problems by relaxing either the constraint on the running time or the constraint of optimality. That means, they are either not polynomial at worst case, but solves some types of the given problem fast, or they are polynomial but usually can not find the optimal solution, just a solution which is relatively close to the optimal one. These are the approximating algorithms. This chapter introduce two of them to solve the safety stock problem. The first was developed by Magnanti et al.[16] while the second by Stephen C. Graves, and Ekaterina Lesnaia.[17]

3.1 Generate strong flow cover cuts with CPLEX

This section presents the approximating algorithm developed by Magnanti et al.[16] This algorithm supposes the demand bound functions to be concave and monotonically increasing. They approximate the optimal solution with two steps. At first, they suppose the demand bound function $D_v(\tau)$ is piecewise linear for every $v \in V$. They add redundant inequalities to the problem to make the CPLEX[20] optimization tool generate strong flow cuts to improve on the run time. Afterwards applying the previous results they approximate the original concave demand bound function with piecewise linear functions, and iterate this step until the gap between the approximated and optimal solution is sufficiently small.

Consider a piecewise linear concave function $\phi(x)$, $x \geq 0$. Denote the number of breakpoints by R, the n^{th} breakpoint by N_n, the gradient of the n^{th} piece by

α_n and the intersection of the extension of the n^{th} piece with the y coordinate line by f_n, see Figure 3.1. Using the multiple-choice approach we can calculate the value of $\phi(x)$ with the following formula:

$$\begin{aligned}
\phi(x) = \min & \sum_{i=1}^{R} (f_i u_i + \alpha_i z_i) \\
\text{s.t.} & \\
& \sum_{i=1}^{R} u_i \leq 1 \\
& \sum_{i=1}^{R} z_i = x \\
& N_{i-1} u_i \leq z_i \leq N_i u_i \\
& u_i \in \{0,\ 1\}\ \forall i.
\end{aligned} \tag{3.1}$$

Supposing all of the demand bound functions are piecewise linear Problem 1 can be written as

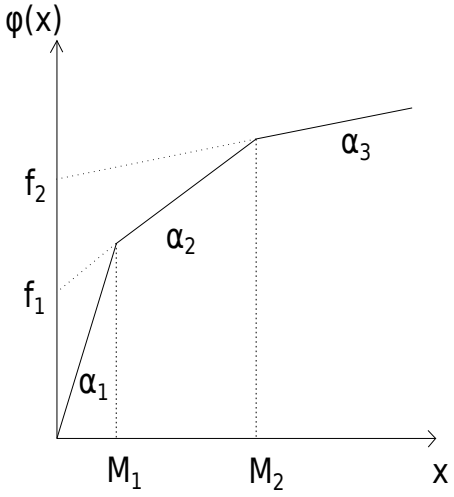

Figure 3.1: Piecewise linear function

Problem 3.

$$\min \sum_{v \in V} (h_v \cdot \sum_{i=1}^{R^v} (f_i^v u_i^v + \alpha_i^v z_i^v))$$

s.t.

$$\sum_{i=1}^{R^v} u_i^v \leq 1 \qquad \forall v \in V$$

$$\sum_{i=1}^{R^v} z_i^v = X^v \qquad \forall v \in V$$

$$N_{i-1}^v u_i^v \leq z_i^v \leq N_i^v u_i^v \qquad \forall i,\ \forall v \in V$$

$$u_i^v \in \{0,\ 1\} \qquad \forall i,\ \forall v \in V$$

$$X^v = IS_v + T_v - S_v \qquad \forall v \in V$$

$$IS_v + T_v \geq S_v \qquad \forall v \in V$$

$$IS_v \geq S_w \qquad \forall v, w \in V\text{: } (wv) \in E$$

$$S_v \in \mathbb{Z},\ \geq 0 \qquad \forall v \in V$$

$$S_v \geq s_v \qquad \forall v \in V,\ \textit{demand node}$$

Lemma 5.

$$\sum_{v \in P} X^v \geq \sum_{v \in P} (T_v) - s_{v_j}$$

holds for all path $P = \{v_1,\ v_2\ \ldots v_j\}$ *where* v_1 *is a source and* v_j *is a sink.*

Proof. By substituting X^{v_i} with $IS_{v_i} + T_{v_i} - S_{v_i}$ and using the following inequalities to give an upper estimation for the left hand side

$$S_{v_i} \leq IS_{v_{i+1}} \qquad \forall v_i \in P$$
$$S_{v_j} \geq s_{v_j}$$

we get the following inequality

$$\sum_{v \in P} X^v \geq IS_{v_1} + \sum_{v \in P} (T_v) - s_{v_j}.$$

Since v_1 is a source, $IS_{v_1} = 0$ stands which gives the desired result:

$$\sum_{v \in P} X^v \geq \sum_{v \in P} (T_v) - s_{v_j}$$

□

Denote Problem 3 with all the redundant inequalities Lemma 5 induce by $\widetilde{P}$. The CPLEX optimization tool uses the embedded inequalities to generate

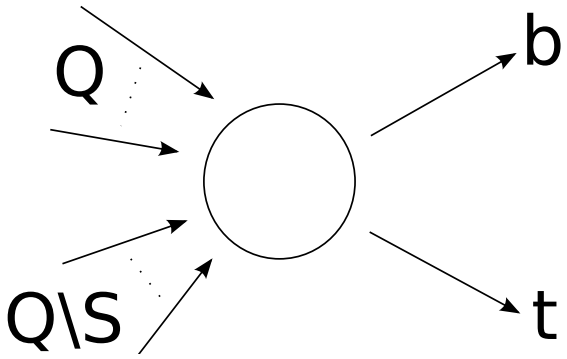

Figure 3.2: Flow cover

flow cover cuts[18] and improves the running time.

Flow cover cut inequality: Consider a node with some inbound arcs Q with capacities m_i and two outbound arcs with capacities t and b. We can formulate a single node flow inequality with these parameters.

$$\begin{aligned} \sum_i x_i - t &= b, \\ 0 \leq x_i &\leq m_i y_i \\ t &\geq 0 \\ y_i &\in \{0,\ 1\} \end{aligned} \tag{3.2}$$

$y_i = 1$ means the i^{th} inbound arc is open.

Definition Consider a subset of inbound arcs $J \subset Q$. We call J a *flow cover* if

$$\sum_{j \in J} m_j = b + \lambda$$

for some $\lambda > 0$.

Consider a flow cover J. Let $S \subset J$ and close arcs in S.

$$\begin{aligned}\sum_{j\in J\setminus S} x_j - t &\le \min\{b,\ \sum_{i\in J} m_i - \sum_{s\in S} m_s\}\\ &\le \min\{b,\ b - (\sum_{s\in S} m_s - \lambda)\}\\ &\le b - (\sum_{s\in S} m_s - \lambda)^+\\ &\le b - \sum_{s\in S}(m_s - \lambda)^+ \end{aligned} \tag{3.3}$$

holds. This yields to the so-called flow cover inequality.

$$\sum_{j\in J} x_j - t \le b - \sum_{j\in J}(m_j - \lambda)^+(1 - y_j). \tag{3.4}$$

If we look back at problem $\widetilde{P}$ we can see the following embedded inequalities.

$$\begin{aligned}\sum_{v\in P}\sum_{r=1}^{R^v} z_r^v &= \sum_{v\in P} T_v - s_{\mathrm{sink}(P)} && \text{for all source-sink path P}\\ z_r^v &\le N_r^v u_r^v && \forall r, j\\ z_r^v &\ge 0 && \forall r, j\\ u_r^v &\in \{0,\ 1\} && \forall r, j\end{aligned}$$

These are the same as in the flow cover cut problem so inequality (3.4) also holds (with the appropriate notations).

Now, one is able to apply this result and approximate the general demand bound functions with piecewise linear ones.

1. The first step is to replace every demand bound function $D_v(\tau)$ by

$$\phi_v^1(\tau) = \frac{D_v(M_v) - D_v(0)}{M_v} \cdot \tau + D_v(0).$$

2. Solve the i^{th} iteration of problem $\widetilde{P}$ with the demand bound functions $\phi_v^i(\tau)$. Suppose the optimal values for the decision variables are X_i^v and $N_n^v \le X_i^v < N_{n+1}^v$.

3. If $N_n^v = X_i^v$, let $\phi_v^{i+1}(\tau) = \phi_v^i(\tau)$, otherwise

$$\begin{aligned}
\alpha_j^{v,\,i+1} &= \alpha_j^{v,\,i} && j < n \\
f_j^{v,\,i+1} &= f_j^{v,\,i} && j < n \\
\alpha_n^{v,\,i+1} &= \frac{D_v(X_i^v) - D_v(N_n^v)}{X_i^v - N_n^v} \\
f_n^{v,\,i+1} &= D_v(X_i^v) - \alpha_n^{v,\,i+1} \cdot X_i^v \\
\alpha_{n+1}^{v,\,i+1} &= \frac{D_v(N_n^v) - D_v(X_i^v)}{N_n^v - X_i^v} \\
f_{n+1}^{v,\,i+1} &= D_v(X_i^v) - \alpha_{n+1}^{v,\,i+1} \cdot X_i^v \\
\alpha_j^{v,\,i+1} &= \alpha_j^{v,\,i} && n+1 < j \\
f_j^{v,\,i+1} &= f_j^{v,\,i} && n+1 < j.
\end{aligned}$$

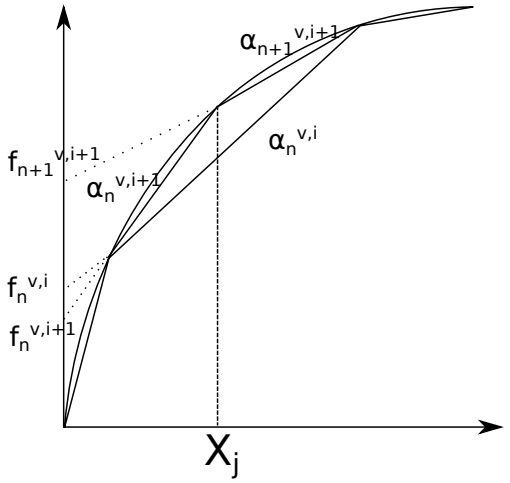

Figure 3.3: Approximate with piecewise linear function

4. Terminate if

$$h_v \cdot |\phi_v^i(X_i^v) - D_v(X_i^v)| \leq \frac{\epsilon}{|V|} \qquad \forall v \in V \tag{3.5}$$

Since the value of $\phi_v^i(\tau) \leq D_v(\tau)$, the optimal solution to the original problem (let us denote it by Opt_o) is always greater or equal to the approximated optimum (Opt_a). Namely,

$$Opt_a \leq Opt_o \tag{3.6}$$

On the other hand, using inequality (3.5)

$$\begin{aligned} Opt_o &\leq \sum_{v \in V} (h_v \cdot D_v(X_i^v)) \\ &\leq \sum_{v \in V} (h_v \cdot \phi_v^i(X_i^v)) + |V| \cdot \frac{\epsilon}{|V|} \\ &\leq Opt_a + \epsilon \end{aligned} \tag{3.7}$$

Inequality (3.6) and (3.7) together induce

$$Opt_a \leq Opt_o \leq Opt_a + \epsilon \tag{3.8}$$

So, choosing the terminating criterion appropriately, the algorithm provides a solution to problem $\widetilde{P}$ arbitrary close to the optimal one. Magnanti at al. showed that this method can significantly improve the computational efficiency. While for a randomly generated graph with 100 nodes and 142 arcs and also randomly generated parameters, it took more than one hour to solve the problem without the redundant inequalities, meanwhile took only 181 seconds with them. For more computational experiences see[16].

3.2 Branch and bound

The approximation algorithm developed by Stephen C. Graves and Ekaterina Lesnaia[17] has a branch and bound structure. The branch and bound scheme requires two main decisions. The first is to determine how to branch, while the second is how to bound. If this two is determined, a branch and bound algorithm has the following structure.

Consider we know a feasible solution $\widetilde{x}$.
repeat
  Make a branching step according to the branching rules.
  Give a lower (or upper, according to the task) bound x_1, $x_2 \ldots$, x_k for all the branches.
  for $i = 1 \rightarrow k$ **do**
    if $x_i \geq \widetilde{x}$ **then**
      Do not search for solution in that branch any more.
    end if
  end for
  Refresh $\widetilde{x}$
until $\widetilde{x}$ is the optimal solution

Of course, the branch and bound scheme does not guarantee that the algorithm is of polynomial running time, but for different problems e.g. Integer

Programming it can provide a generally quick and efficient solving method[18]. This section introduce a branch and bound algorithm for Problem 1 developed by Stephen C. Graves and Ekaterina Lesnaia[17].

In Problem 1, there are two different kinds of decision variables, namely the service times quoted by nodes S and the inbound service times IS. If the variables of one type are known for an optimal solution, the variable of the other type can easily be obtained. The following two lemmas show how to do that.

Lemma 6. *If there exists an optimal solution to Problem 1 with outbound service times $\{S_v : v \in V\}$, there is an optimal solution with the same outbound service times and with inbound services times*

$$IS_v = \max_{(uv)\in A}\{S_u\}$$

Proof. $IS_v \geq \max_{(uv)\in A}\{S_u\}$ holds for the given optimal solution, because this is a constraint of Problem 1

On the other hand, if $IS_v > \max_{(uv)\in A}\{S_u\}$ for a node v, $IS_v - 1$ also satisfies all of the constraints and since $D_v(\tau)$ the demand bound function is monotonically increasing, therefore

$$h_v \cdot D_v(IS_v + T_v - S_v) \geq h_v \cdot D_v((IS_v - 1) + T_v - S_v).$$

So, with $IS_v := IS_v - 1$ the new feasible solution has not greater objective value than the original one. □

Lemma 7. *If there exists an optimal solution to Problem 1 with inbound service times $\{IS_v : v \in V\}$, there is an optimal solution with the same inbound service times and with outbound services times*

$$S_v = \min_{(vu)\in A}\{IS_u\}$$

Proof. The proof of lemma 6 can be modified easily to prove Lemma 7. □

Lemma 8. *There always exists an optimal solution to Problem 1 which satisfies the following claims.*

- $IS_v = 0 \ \forall$ *source node* v.
- $S_v = s_v \ \forall$ *demand node* v.

Proof. The two claims are straightforward, as all the demand bound functions are monotonically increasing. □

These results can be used to construct a branch and bound algorithm.

At first, let us define layers. Let layer L_0 be the set of source nodes, i.e.

$$L_0 = \{v \mid v \in V,\ \rho(v) = 0\}$$

Define the other layers in the following way

$$L_i = \{v \in V \mid \text{the longest directed path between } L_0 \text{ and } v \text{ has lenght of } i\}$$

Label the nodes with integers in such a way that all nodes in L_i have larger numbers than the nodes in $L_j\ \forall j < i$, and there are not two nodes with the same number. Practically, we label an arbitrary node in L_0 with 1, an other with 2 and so on until all nodes in L_0 are labelled, then we move on and continue with nodes in L_1 e.t.c. This way, if $|V| = N$, the nodes are labelled with $1, 2 \ldots N$. Let $r_i := |L_i|$.

The next step is to construct the branching tree. If there are outbound service times S_v assigned to all nodes in layers L_i, $i < j$, all the inbound service times of the nodes in L_j can be defined. This immediately follows from Lemma 6 and the definition of L_j. So, supposing the service times of layers L_i $i < j$ are fixed, the resultant sub-problem is also an inventory optimization problem (Problem 1) with some additional constraints on some inbound service time — these arise because of arcs between L_i $i < j$ and L_k $k > j$ — but this additional constraints can be easily treated. Therefore a branching step would be to assign a different service time values to node v supposing all nodes with smaller labels has a fixed service time.

Now we have to compute a lower bound for the sub-problems. After a branching step the nodes with given in and outbound service times can be removed as no decision variables are associated with them. Denote the remainder acyclic graph by $\widetilde{D}$. To set the lower bound, remove some arcs from $\widetilde{D}$ until the remaining graph is a tree. Compute the optimal service times for the tree according to the algorithm presented in Section 2.2. Hence removing an arc corresponds to removing a constraint of type

$$IS_v \leq S_u,\ (uv) \in A.$$

Therefore, this optimum is a suitable lower bound for Problem 1 with graph $\widetilde{D}$.

The last step is to compute a solution to every sub-problem. The previously described method results a solution to a tree. But increasing some inbound service times result in a feasible solution to the original problem.

Chapter 4

Supply chains with capacity constraints

Tor Schoenmeyr and Stephen C. Graves[19] extended the original model by adding capacity constraints to the stages of the supply chain. This chapter presents their results.

4.1 Base-Stock Replenishment Policy

Let us formulate a very similar problem to the original one described by Chapter 2, but add a c_v capacity constraint to every node in V. Suppose node v can manufacture product more than c_v at no time interval of length one.

Observation If for some node v

$$c_v \cdot \tau < D_v(\tau) \qquad \forall \tau : 0 \leq \tau \leq M_v,$$

then the guaranteed service time assumption can not be satisfied.

Therefore we suppose:

$$\forall v \in V \ \exists \tau : \ 0 \leq \tau \leq M_v \text{ and } c_v \cdot \tau \geq D_v(\tau)$$

Since the demand — and therefore the incoming material — can exceed the production for some short interval, certain amount of unprocessed stock must be stored (we suppose there is no maintenance cost of these type of stock). This arrived but due to the capacity constraint unprocessed material is called the internal queue and denoted at time t by $IQ_v(t)$. We also need another function. Let $R_v(t)$ be the stock, that node v started to process at time t. $R_v(t)$ can be computed as

$$R_v(t) = \min\{IQ_v(t-1) + d_v(t - IS_v);\ c_v\} \tag{4.1}$$

while the internal queue is

$$IQ_v(t) = IQ_v(t-1) + d_v(t - IS_v) - R_v(t) \tag{4.2}$$

The total inventory can be computed in the following way

$$I_v(t) = I_v(t-1) + R_v(t - T_v) - d_v(t - S_v) \tag{4.3}$$

This equation can be reformulated using Equation (4.2) as

$$I_v(t)+IQ_v(t-T_v) = I_v(t-1)+IQ_v(t-T_v-1)+d_v(t-IS_v-T_v)-d_v(t-S_v) \tag{4.4}$$

Now we can recursively replace $I_v(t-1) + IQ_v(t - T_v - 1)$ with $I_v(t-2) + IQ_v(t-T_v-2)+d_v(t-IS_v-T_v-1)-d_v(t-S_v-1)$, $I_v(t-2)+IQ_v(t-T_v-2)$ with $I_v(t-3) + IQ_v(t - T_v - 3) + d_v(t - IS_v - T_v - 2) - d_v(t - S_v - 2)$ and so for, until we get the equation

$$I_v(t) = I_v(0) - d_v(t - IS_v - T_v) - \cdots - d_v(t - S_v) - IQ_v(t - T_v) \tag{4.5}$$

For the sake of convenience let us introduce the following notation

$$\sum_{i=x}^{y} d(x) = d(x, y)$$

Using this notation, we get

Lemma 9.

$$IQ_v(t) = \max_{n \in \mathbb{Z}_+}(d_v(t - IS_v - n + 1, t - IS_v) - cn) \tag{4.6}$$

Proof. Replacing $R_v(t)$ with Equation (4.1) in Equation (4.2) we get

$$\begin{aligned} IQ_v(t) &= IQ_v(t-1) + d_v(t - IS_v) - \min\{IQ_v(t-1) + d_v(t - IS_v);\ c_v\} \\ &= \max\{0;\ IQ_v(t-1) + d_v(t - IS_v) - c_v\} \end{aligned}$$

Using this recursively

$$
\begin{aligned}
IQ_v(t) &= \max\{0;\ IQ_v(t-1) + d_v(t - IS_v) - c_v\} \\
&= \max\{0;\ \max\{0;\ IQ_v(t-2) + d_v(t - IS_v - 1) - c_v\} \\
&+ d_v(t - IS_v) - c_v\} \\
&= \max\{0;\ d_v(t - IS_v) - c_v;\ IQ_v(t-2) + d_v(t - IS_v) \\
&+ d_v(t - IS_v - 1) - 2c_v\} \\
&\vdots \\
&= \max_{n \in \mathbb{Z}_+}\{d_v(t - IS_v - n + 1,\ t - IS_v) - nc_v;\ 0\} \\
&= \max_{n \in \mathbb{Z}}\{d_v(t - IS_v - n + 1,\ t - IS_v) - nc_v\}
\end{aligned}
$$

which gives the desired result.

□

With the previous lemma, Equation (4.5) can be written as

$$
\begin{aligned}
I_v(t) = I_v(0) - d_v(t - IS_v - T_v, t - S_v) - \\
- \max_{n \in \mathbb{Z}}(d_v(t - IS_v - T_v - n + 1, t - IS_v - T_v) - c_v n)
\end{aligned} \tag{4.7}
$$

$I_v(t) \geq 0$ by definition, therefore

$$
\begin{aligned}
I_v(0) \geq d_v(t - IS_v - T_v, t - S_v) + \\
+ \max_{n \in \mathbb{Z}}(d_v(t - IS_v - T_v - n + 1, t - IS_v - T_v) - c_v n)
\end{aligned}
$$

for every valid demand and t, so

$$
I_v(0) \geq D_v(IS_v + T_v - S_v) + \max_{n \in \mathbb{Z}}(D_v(n) - c_v n) \tag{4.8}
$$

must hold for every v in V. This term is similar to the one introduced in Chapter 2. We can formulate an optimization problem with these cost functions similar to Problem 1. The only difference is the additional term occurred due to the capacity constraints.

As we mentioned above for the algorithm for trees and clusters of commonality the assumption of concavity is not necessary so they work with these cost functions as well. For a general network the branch and bound algorithm introduced in Section 3.2 uses only the tree algorithm, so it works with capacity constraints as well.

4.2 Censored Order Policy

One may note the material hold at the internal queues is a dead load. The supplier has to ship this mater but the customer does not use it. In fact, it would be better for the supplier to ship this mater at a later time while from the point of view of the customer it does not make any difference. Inspired by this observation this section introduces a new policy which results lower holding cost for the network.

Suppose demand node v has nothing in its internal queue at time t and observes $d_v(t)$ demand where $d_v(t) > c_v$ the capacity constraint. If this node places $d_v(t)$ demand on its suppliers, at time $t + IS_v$, it will have more unprocessed material than c_v, so it will have to place $d_v(t) - c_v > 0$ material into the internal queue.

Suppose this node places only c_v order onto its suppliers at time t and $d_v(t+1) + (d_v(t) - c_v)$ at time $t + 1$. At this case node v will not have any unprocessed material in the internal queue at time t, its suppliers have to ship less product at time t and the same amount at the $[t,\ t+1]$ interval than before. On the other hand node v will have the same amount of finished product at time $t+T_v$, $t+T_v+1$ e. t. c. So, if node v "censors" its order and never order more than the capacity constraint, the guaranteed service times can be hold with the same S_v and the safety stock of the suppliers can be reduced.

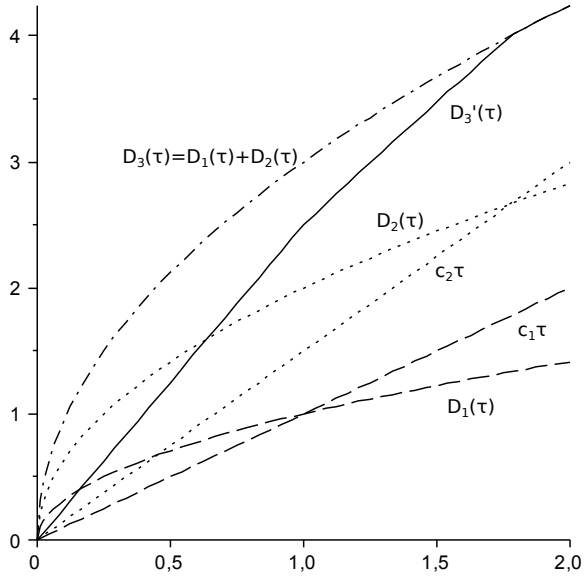

Figure 4.1: The new demand bound

Since with this policy the nodes will not place the same demand on their suppliers as they observe, it is convenient to make difference between the two kinds of demand. Let us denote the observed demand by $d_v^o(t)$ and the placed by $d_v^p(t)$. With these notation, the censored order policy means

$$d_v^p(t) = \min\{d_v^o(0,t) - d_v^p(0,t-1);\ c_v\} \tag{4.9}$$

for every node in V. With this order policy, the observed maximal demand $D_u^o(\tau)$ for a demand node

$$D_v^o(\tau) = D_v(\tau) \tag{4.10}$$

and for a non-demand node u

$$D_u^o(\tau) = \sum_{(uv)\in A} \min(D_v^o(\tau),\ c_v\tau). \tag{4.11}$$

Observation 1. The new $d_v^p(t)$ demands are valid if the original ones are valid. Namely

$$\sum_{i=1}^{\tau} d_v^p(t+i) \le D_v(\tau)$$

2. $D_v^o(\tau)$ is concave and monotone if the originals are concave and monotone.
3. With the new demands $IQ_v(t) = 0$ for $\forall n \in V$, $t \ge 0$.
4. And the most valuable one

$$\sum_{i=1}^{\tau} d_v^o(t+i) \le D_v^o(\tau) \tag{4.12}$$

Inequality (4.12) induces no node has to hold safety stock more than

$$D^o(IS + T - S) + \max_{n\in\mathbb{Z}}(D^o(n) - cn) \tag{4.13}$$

if it has inbound service time IS, outbound service time S, production lead time T and capacity constraint c. Since $D_v^o(\tau) \le D_v(\tau)$ every node has to hold no more safety stock than they hold without the Censored Order Policy.

This problem can be solved like the Base-Stock Replenishment Policy case, since the only difference are the demand bound functions. Therefore the dynamical programming method for trees, the one for graphs with Clusters of Commonality, and the branch and bound algorithm is applicable at this case.

Chapter 5

Supply chains with different outbound service times

The original model of Humair and Willems[3] suppose that a node guarantees the same service time to all of its customers. In this chapter, we construct the model that allows non-uniform service times and make the original algorithm for trees applicable at this case. This is an extension of the original model and a more accurate model of the real problem. It is also useful to the companies, because this model does not result higher total inventory, and sometimes allows a lower one.

Section 5.2 introduces an algorithm which solves the non-uniform optimization problem for a bipartite graph $D = (S,\ T,\ A)$ where all the arc has the source in the same partition (namely $(uv) \in A \Rightarrow u \in S$ and $v \in T$).

The last section shows the non-uniform case is also NP-hard if we allow arbitrary cost functions.

Suppose node v has costumers $u_1 \dots u_k$, its inbound service time is IS and has production time T. Furthermore, suppose u_i has demand bound $D_i(\tau)$ and needs ϕ_{vi} unit material from v to produce one unit of product. As we have seen in Chapter 2, if v quotes S_{vi} service time to u_i, it has to hold safety stock equal to $\phi_{vi} \cdot D_i(IS + T - S_{vi})$. If the holding cost of x unit of safety stock at node v is $C_v(x)$, the total holding cost is

$$C_v\left(\sum_{i=1}^{k} \phi_{vi} \cdot D_i(IS + T - S_{vi})\right) \tag{5.1}$$

Let us denote $\sum_{i=1}^{k} \phi_{vi} \cdot D_i(IS + T - S_{vi})$ by $\widetilde{D_v}(IS,\ S_{v1}, \dots,\ S_{vk})$ and the number of customers of v by $\delta(v)$. With this notation, the safety stock problem

with non-uniform service times can be written for graph $D = (V, E)$ as

Problem 4.

$$\min \sum_{v \in V} (C_v(\widetilde{D_v}(IS_v,\ S_{vu_1}, \ldots,\ S_{vu_{\delta(v)}})))$$

s.t.

$$\begin{aligned}
&IS_v + T_v \geq S_{vu} && \forall v, u \in V\colon (vu) \in E \\
&IS_v \geq S_{uv} && \forall v, u \in V\colon (uv) \in E \\
&S_{vu} \in \mathbb{Z},\ \geq 0 && \forall v, u \in V\colon (vu) \in E \\
&S_v \geq s_v && \forall v \in V,\ \textit{demand node}
\end{aligned}$$

Observation For an optimal solution to Problem 4 and for every service time

$$S_{uv} = \min\{IS_v,\ IS_u + T_u\}$$

5.1 Algorithm for trees

Let us suppose the graph is a tree and the cost function is linear just like at the original uniform case. Namely, $C_v(x) = h_v \cdot x$.

Label the nodes and define the $f_v(x)$ and $g_v(x)$ functions as in the uniform case. Namely, let $f_v(x)$ be the optimal inventory holding cost for N_v supposing $S_{vp(v)} \leq x$ if $(vp(v)) \in A$ and $g_u(x)$ be the optimal inventory holding cost for N_u supposing $IS_u \geq x$ if $(p(v)v) \in A$.

The efficiency of the algorithm depends on how fast we can calculate the $g_v(x)$ and $f_u(x)$ functions. If we can do it at polynomial time, the resulting algorithm is also of polynomial running time, because we iterate on the nodes only once. To calculate the desired functions we use the following lemma.

Lemma 10. *If IS_v is given for node v, the optimal safety stock is the sum of the individual optima of the branches.*

Proof.

$$\begin{aligned}
C_v(\widetilde{D_v}(IS_v,\ S_{vu_1}, \ldots,\ S_{vu_{\delta(v)}})) &= h_v \cdot \widetilde{D_v}(IS_v,\ S_{vu_1}, \ldots,\ S_{vu_{\delta(v)}}) \\
&= h_v \cdot \sum_{i=1}^{\delta(v)} (D_{u_i}(IS_v + T_v - S_{vu_i})) \\
&= \sum_{i=1}^{\delta(v)} (h_v \cdot D_{u_i}(IS_v + T_v - S_{vu_i}))
\end{aligned}$$

Therefore

$$\min_{S_{vu_1},\ldots,\ S_{vu_{\delta(v)}}} C_v(\widetilde{D_v}(IS_v,\ S_{vu_1},\ldots,\ S_{vu_{\delta(v)}})) =$$

$$= \sum_{i=1}^{\delta(v)} (\min_{S_{vu_i}}(h_v \cdot D_{u_i}(IS_v + T_v - S_{vu_i})))$$

□

This result can be applied to calculate $g_v(x)$.

$$c_v(S_{vu_1},\ S_{vu_2},\ \ldots,\ S_{vu_{\delta(v)}}, IS_v) = h_v \cdot \widetilde{D_v}(IS_v,\ S_{vu_1},\ldots,\ S_{vu_{\delta(v)}}) +$$
$$+ \sum_{\substack{u_i \in ch(v) \\ (vu_i) \in A}} g_{u_i}(S_{vu_i}) + \sum_{\substack{w \in ch(v) \\ (wv) \in A}} f_w(IS_v)$$

and

$$g_v(x) = \min_{x \le IS_v \le M_v - T_v} \min_{0 \le S_v \le M_v} c_v(S_v, IS_v)$$
$$= \min_{x \le IS_v \le M_v - T_v} \left(\sum_{i=1}^{\delta(v)} (\min_{S_{vu_i}}(h_v \cdot D_{u_i}(IS_v + T_v - S_{vu_i})) + g_{u_i}(S_{vu_i})) + \right.$$
$$\left. + \sum_{\substack{w \in ch(v) \\ (wv) \in A}} f_w(IS_v) \right)$$

For simplicity we can assume $p(v) = u_{\delta(v)}$. Function $f_v(x)$ can be obtained similarly

$$f_v(x) = \min_{0 \le S_{vp(v)} \le x} \min_{0 \le IS_v \le M_v - T_v} c_v(S_v, IS_v)$$
$$= \min_{0 \le IS_v \le M_v - T_v} \left(\sum_{i=1}^{\delta(v)-1} (\min_{S_{vu_i}}(h_v \cdot D_{u_i}(IS_v + T_v - S_{vu_i})) + g_{u_i}(S_{vu_i})) + \right.$$
$$\left. + \min_{0 \le S_{vp(v)} \le x} (h_v \cdot D_{u_{\delta(v)}}(IS_v + T_v - S_{vu_{c(c)}})) + \min_{0 \le y \le IS_v} \left(\sum_{(wv) \in A} f_w(y) \right) \right)$$

Although these expressions may seems very complicated, but they can be enumerated at polynomial time, therefore

Claim 1. *The optimal solution can be obtained by enumerating the functions $f_v(x)$ and $g_v(x)$ step-by-step at polynomial time.*

5.2 Algorithm for bipartite graphs with head and target partition

Let us investigate the case when the underlying graph is bipartite, $D = (S, H, A)$ where S and H are the sets of nodes and A is the set of arcs. Moreover suppose if $(xy) \in A$, then $x \in S$ and $y \in H$. Also suppose the cost function is linear for every node.

Observation The set of the demand nodes is equal to H.

Observation If $s \in S$, then $IS_s = 0$.

Let the $c_h(x)$ $(h \in H)$ function be the following

$$c_h(x) = h_h \cdot D_h(x + T_h - s_h) + \sum_{(sh)\in A} (h_s \cdot \phi_{sh} \cdot D_h(\max\{T_s - x, 0\})) \quad (5.2)$$

We can suppose $IS_h = \max_{(sh)\in A}\{S_{sh}\}$ and $S_{sh} = \min\{IS_h, T_s\}$ for every $h \in H$, $s \in S$. Since the cost function is linear,

$$\begin{aligned}
&\sum_{v\in S\cup H} (C_v(\widetilde{D_v}(IS_v,\ S_{vu_1}, \ldots,\ S_{vu_{\delta(v)}}))) = \\
&\quad = \sum_{s\in S}(h_s \cdot \widetilde{D_s}(IS_v,\ S_{su_1}, \ldots,\ S_{su_{\delta(s)}})) + \sum_{h\in H}(h_h \cdot D_h(IS_h + T_h - s_h) \\
&\quad = \sum_{s\in S}\sum_{(sh)\in A} h_s \cdot \phi_{sh} \cdot D_h(T_s - S_{sh}) + \sum_{h\in H}(h_h \cdot D_h(IS_h + T_h - s_h) \\
&\quad = \sum_{h\in H} c_h(IS_h) \quad (5.3)
\end{aligned}$$

Therefore the goal is to minimize $\sum_{h\in H} c_h(IS_h)$. If we take a closer look at the definition of $c_h(x)$, the value of the inbound service time of another $h' \in H$ does not have any impact on the value of $c_h(x)$. Therefore the minimization can be done separately.

Lemma 11.

$$\min_{\{IS_h:\ h\in H\}} \sum_{h\in H} c_h(IS_h) = \sum_{h\in H} \min_{IS_h} c_h(IS_h)$$

□

To get an algorithm that solves Problem 4 in this case we only have to add that the minimization of $c_h(x)$ is over the $[0, \max_{(sh)\in A}\{T_s\}]$ interval, so the minima can be enumerated.

5.3 The complexity of the general case

This section proves the following theorem.

Theorem 2. *Problem 4 is NP-hard for a general acyclic digraph.*

Proof. At fist we choose a special cost function.

$$C_v(x) = \begin{cases} h_v & \text{if } x > 0 \\ 0 & \text{if } x = 0 \end{cases} \qquad \forall v \in V \tag{5.4}$$

Lemma 12. *Consider an optimal solution to Problem 4 with cost functions defined above and with a node v that guarantees service times $S_{vu_1}, \dots\ S_{vu_{\delta(v)}}$ to its costumers. Then the solution with the same service times, except $S_{vu_i} = \min\{S_{vu_1}, \dots\ S_{vu_{\delta(v)}}\}$ is also optimal.*

Proof. The value of the safety stock at nodes other than v can not increase, because their outbound service times are not effected and their inbound service times are not increased.

For the sake of convenience denote $\min\{S_{vu_1}, \dots\ S_{vu_{\delta(v)}}\}$ by $\widetilde{S_v}$. If $h_v = 0$ we are ready. Otherwise

$$\sum_{i=1}^{\delta(v)} D_{u_i}(IS_v + T_v - S_{vu_i}) > 0 \Leftrightarrow C_v(\widetilde{D_v}(IS_v,\ S_{vu_1}, \dots,\ S_{vu_{\delta(v)}})) = h_v$$

Moreover

$$\sum_{i=1}^{\delta(v)} D_{u_i}(IS_v + T_v - S_{vu_i}) = 0 \Leftrightarrow IS_v + T_v - S_{vu_i} = 0 \qquad \forall i$$

Therefore

$$C_v(\widetilde{D_v}(IS_v,\ S_{vu_1}, \dots,\ S_{vu_{\delta(v)}})) = 0 \Leftrightarrow S_{vu_i} = IS_v + T_v \qquad \forall i$$

and at that case

$$S_{vu_i} = \widetilde{S_v} \qquad \forall i$$

which induce

$$C_v(\widetilde{D_v}(IS_v,\ \widetilde{S_v}, \ldots,\ \widetilde{S_v})) = h_v \Leftrightarrow C_v(\widetilde{D_v}(IS_v,\ S_{vu_1}, \ldots,\ S_{vu_{\delta(v)}})) = h_v$$

and that proves the lemma.

□

Lemma 13. *There exists an optimal solution to Problem 4 with uniform service times.*

Proof. This is a straightforward consequence of lemma 12. □

Let us consider a safety stock optimization problem. Denote the optimum of the total safety stock by opt_u at the uniform case and by opt_{nu} at the non-uniform case. At the general case

$$opt_u \geq opt_{nu}$$

holds. On the other hand Lemma 13 shows at this special case, when the cost functions have the form of Equation (5.4)

$$opt_u = opt_{nu} \tag{5.5}$$

also holds.

Let us specialise the problem a little more. First of all add an extra P node to the supply chain and arcs (vP) for every non-demand node. Let $D' = (V', A')$ the resulting graph. Let the demand bound function be

$$D_v(\tau) = \begin{cases} 0, & \text{if } \tau = 0 \\ 1, & \text{if } \tau \geq 1 \end{cases}$$

for all demand nodes including P, and

$$h_v = \begin{cases} 1 & \text{if } v \neq P \\ |V'| & \text{if } v = P \end{cases}$$

In Section 2.4, we had shown the uniform case for this problem is equivalent to the vertex cover problem. Moreover with Lemma 13

$$\tau = opt_u = opt_{nu} \tag{5.6}$$

where τ is the optimal vertex cover of the original non directed graph. This proves the non-uniform case is also NP-hard. □

Conclusion

At first, we analysed the original model of Graves and Willems[3], and the related algorithms for special supply chain networks. Then, we proved that Problem 1 is NP-hard. In Chapter 3, we presented two approximating algorithms. The first one by Magnanti at al.[16] which uses strong flow cover cuts and the CPLEX optimization tool. The second one by Graves and Lesnaia[17] which was a branch and bound algorithm. In Chapter 4, we extended the original model, introducing the concept of capacity constraints. Finally, Chapter 5 examined a modified version of the original model where the nodes are permitted to guarantee different service times to their customers. We also gave a polynomial time algorithm for trees and another one for bipartite graphs with head and target partition. We also showed this problem is NP-hard for general graphs.

We finish this thesis with mentioning some future research areas. It would be interesting to develop different approximating algorithms for the non-uniform case, and algorithms that can handle non-uniform service times and capacity constraints simultaneously. Another area is to extend the model. The first way is to include shipment times. The second way is to let the stages choose supplier from stages that manufacture the same product.

Appendix A

Common notations

- $V(G)$ is the node set of graph G.
- $A(G)$ is the arc set of graph G.
- $I_G(V')$ is the induced arcs of V' in the graph G.
- $\delta(v)$ is the outdegree of node v.
- $\rho(v)$ is the indegree of node v.
- S_v is the service time node v guarantees.
- IS_v is the inbound service time.
- $D_v(\tau)$ is the demand bound function.
- T_v is the production time of node v.
- $l(v)$ is the label of node v.
- M_v is the maximal lead time of node v.
- h_v is the holding cost per unit for node v.
- N_v is the child sub-graph of node v.
- $p(v)$ is the parent function.
- $ch(v)$ is the set of children of node v.

Bibliography

[1] Sven Axs ater: "Inventory Control", Springer Science+Business Media, LLC, 2006.

[2] http://en.wikipedia.org/wiki/Inventory

[3] Stephen C. Graves, Sean P. Willems: "Optimizing Strategic Safety Stock Placement in Supply Chains" 1998

[4] Stephen C. Graves, Sean P. Willems: "Optimizing the Supply Chain Configuration for New Products" 2005

[5] Simpson, K. F.: "In-process Inventories" *Operations Research* 863-873, 1958.

[6] Inderfurth, K.: "Safety Stock Optimization in Multi-stage Inventory Systems" *International Journal of Production Economics* Vol. 24 103-113, 1991.

[7] Inderfurth, K.: "Valuation of Leadtime Reduction in Multi-Stage Production Systems" in: G. Fandel, T. Gulledge and A. Jones (eds.) *Operations Research in Production Planning and Inventory Control* Springer, Berlin, 413-427, 1993.

[8] Inderfurth, K., and S. Minner: "Safety Stocks in Multi-Stage Inventory Systems under Different Service Levels" 1995.

[9] Minner, S.: "Dynamic Programming Algorithms for Multi-Stage Safety Stock Optimization" 1995.

[10] Lee, H. L. and C. Billington: "Material Management in Decentralized Supply Chains" *Operations Research* 41, 835-847, 1993.

[11] Glasserman, P. and S. Tayur: "Sensitivity Analysis for Base-stock levels in Multiechelon Production-inventory Systems" *Management Science* 41, 263-281, 1995.

[12] Ettl, M., G. E. Feigin, G. Y. Lin and D. D. Yao: "A Supply Network Model with Base-Stock Control and Service Requirements" IBM Technical Report (RC 20473), 1996.

[13] Salal Humair, Sean P. Willems: "Optimizing Strategic Safety Stock Placement in Supply Chains with Clusters of Commonality" *Operations Research* Vol. 54, No. 4, July–August 2006, pp. 725–742

[14] Ekaterina Lesnaia, Iuliu Vasilescu, and Stephen C. Graves: "The Complexity of Safety Stock Placement in General-Network Supply Chains" Working Paper, Massachusetts Institute of Technology, USA, 2004.

[15] Garey, Michael R., Johnson, D. S.: "Computers and Intractability: A Guide to the Theory of NP-Completeness" 1979

[16] Thomas L. Magnanti, Zuo-Jun Max Shen, Jia Shu, David Simchi-Levi and Chung-Piaw Teo: "Inventory Placement in Acyclic Supply Chain Networks" 2004

[17] Stephen C. Graves and Ekaterina Lesnaia: "Optimizing Safety Stock Placement in General Network Supply Chains"

[18] G. L. Nemhauser and L. A. Wosley: "Integer and Combinatorial Optimization"

[19] Tor Schoenmeyr and Stephen C. Graves: "Strategic safety stocks in supply chains with capacity constraints" 2009

[20] http://en.wikipedia.org/wiki/CPLEX