Decomposition Approach to Serial Inventory Systems under Uncertainties

Jennifer Kusuma

Honours Thesis, 2005

Supervisors: Prof. Peter Taylor,
Dr. Andre Costa
Acknowledgements

I would like to thank my supervisors, Peter Taylor and Andre Costa. It has been a wonderful experience to work under your guidance. Peter, for giving me the support and care that went well beyond this project. Andre, for the commitment and enthusiasm you put into this and for always pushing me to question the things I would otherwise take for granted.

Thanks to my family and friends. My father especially, for always making mathematics seem exciting and simple; you never stop inspiring me. Jayne Lau, who never fails me in difficult times. Your enthusiasm and energy have influenced me in more ways that you could imagine.

Finally I would like to extend my gratitude to the lecturers, the graduate and other honours students, whose friendship and support made this experience as rewarding and enjoyable as it has been.
Contents

1 Introduction 4

2 Literature Review 7

3 Preliminaries 9

3.1 Dynamic Programming 9

4 Single-Installation Inventory System 12

4.1 Newsvendor Problem 12

4.2 Dynamic Inventory Management 14

4.2.1 The Case With No Time Lag 15

4.2.2 The Case With Time Lag 24

5 Multi-echelon Inventory System 33

5.1 Classical Decomposition of Multi-Echelon Systems 34

5.1.1 Introduction 34
Chapter 1

Introduction

We study the optimization problem for serial multi-echelon inventory systems under uncertainties. Figure 1.1 illustrates an example of such problems, with three installations, each receiving stock from the installation above it, and installation 3 receiving stock from an outside supplier. Demand originates only in the lowest installation and unfulfilled demands are backlogged. The controller is subjected to holding, backlogging and ordering costs and wishes to minimize the total cost over the time horizon.

![Figure 1.1: serial inventory system](image)

The inventory problems described above - and their variations - have been studied in the last 40 years. Clark and Scarf in 1960 [8] first introduced an efficient method to solve such systems, by decomposing them into single-installations problems and solving each subproblem individually. The method has since been improved and adapted to solve variations of the problem. In this thesis we look at a different approach to solving multi-echelon systems, as proposed by Muharremoglu and Tsitkilis in 2001 [18]. The new method involves decomposing the multi-echelon system into a family of subproblems as
well, but with each subproblem consisting of a customer-unit pair.

The important contribution of Muharremoglu and Tsitsiklis can be summarized as follows. It is a generalization of the Clark and Scarf problem formulation. The new method allows the mathematical model to incorporate more random factors by allowing the existence of a Markov chain that affects the dynamic of the system. The demand as well as the leadtimes can now be modelled by random variables that depends on the state of the Markov chain. Moreover, the new method gives rise to the formulation of an efficient algorithm with polynomial complexity.

We assess the new decomposition method by checking its results against the Clark and Scarf method. The thesis is set out as follows. We begin by reviewing the literature leading up to the development of this new method. In Chapter 3 we summarize the basic ideas of dynamic programming. Dynamic programming is a common thread of the algorithms used in the literature, since stochastic inventory systems are often formulated as recursive equations.

In Chapter 4 we review three single-installation systems. We start with the single time period case, which is the Newsvendor problem, and extend it to the case with \( n \) time periods remaining, often referred to as the problem of dynamic inventory management. Two situations are considered for the latter: when there is no time lag in the delivery and when there is. We prove the optimality of a certain class of policies, \((S, s)\) policies, under K-convexity and implement Matlab codes for case with time lag. The results and methods studied for each model help us understand the dynamic of the next model, which incorporates more complexities.

Chapter 5 is divided into three sections. First we review Clark and Scarf decomposition method, which makes use of the results presented in Chapter 4. We then present Muharremoglu and Tsitksiklis new approach and implement the algorithm in Matlab. A numerical comparison from the two methods is provided.

Chapter 6 looks at another paper written by Tsitsiklis and Muharremoglu in 2003 [19]. In their previous paper [18], the leadtime is considered as an external parameter that is
not within our control. In [19], the single-unit single-customer decomposition approach is used to allow dynamic leadtime management; we are allowed to control the leadtimes in supply chain management. With this relaxation, many real life inventory systems can be modelled and solved efficiently.
Chapter 2

Literature Review

To fully appreciate the contribution of Muharremoglu and Tsitsiklis [18], we need to consider the development of three broad classes of inventory systems and show how the limitations were overcome by the new decomposition approach.

Multi-echelon Inventory Systems. Relatively little analytical work had been done on multi-echelon systems before Clark and Scarf put forward their idea of decomposition in 1960 [14]. The results from solving each sub-system gave the optimal policies for the original serial inventory system for a finite horizon problem. Their method much guided the literature in the next 40 years to come. Federgruen and Zipkin [12], for example, extended this result to the stationary infinite horizon model while Chen and Zheng [6] considered the continuous time model. In these traditional serial systems, we had independent and identically distributed (i.i.d.) demand and deterministic leadtime assumptions and a unit must pass through all installations in the system.

The relaxation of i.i.d. demand. One major theme in the continuing development of inventory theory was to incorporate more realistic assumptions about product demand and leadtimes into inventory models. The following papers considered inventory problems with demand distribution that depended on the state of a modulating Markov chain. For the case of single-installation system with such a demand model, [27], [5], [24] proved the
optimality of state dependent \((S, s)\) policies under different time horizon and backlogging assumptions. Muharremoglu and Tsitsiklis [18] noted that Chen and Song (2001) wrote the only paper that characterized the optimal policies for serial inventory systems under a Markov modulated demand assumption. All of the above papers, however, assumed deterministic leadtimes in their formulations.

Relaxation of the deterministic leadtimes. There had been a considerable amount of study of stochastic leadtimes in inventory control. The cases under consideration, however, were variations of single-installation systems ([15], [29], [28]).

By adopting a single-unit, single customer decomposition method, Muharremoglu and Tsitkilis [18], were able to characterize and provide an algorithm for a serial multi-echelon inventory system with the leadtimes and demands both dependent on a modulating Markov chain.

Muharremoglu and Tsitsiklis further extended their decomposition idea, and applied it to supply chain management [19]. They incorporated multiple suppliers with different lead times and costs as control variables. Previous attempts were restricted to cases where the costs were additive, [17], or when there were only two suppliers, and the difference between the lead times of different suppliers is one - see for example [3], [9], [13], [1]. Muharremoglu and Tsitsiklis [19] relaxed this additive assumption on the costs and the restriction on the time lag. Units were allowed to jump installations, and the controller decided its path. As mentioned, the model was proposed to be able to represent more complex inventory systems.
Chapter 3

Preliminaries

3.1 Dynamic Programming

Dynamic programming was born of the realization that certain features recurred again and again. Although some have considered Dynamic Programming (DP) as an optimization tool ([20], [11]), we would refer to Denardo’s [10] more general definition: dynamic programming is the collection of mathematical tools used to analyze sequential decision processes. As such, DP can be used for a variety of problems, well beyond optimization ones. In this thesis however, we are concerned with using DP to optimize inventory systems.

The two important insights in DP:

• The original problem is decomposed into a family of subproblems.

• We are only required to solve the relatively easier subproblems. The result for the original problem is recovered from the ‘combination’ of the results from the subproblems.

The two insights above come from Bellman’s principle of optimality. Consider the prob-
lem of determining the shortest path to go from point A- C- E- F. The principle of optimality says that the optimal path from A to F can be recovered by finding the optimal path from the subproblems A-C and C - F. The latter can in turn be decomposed into finding the optimal paths from C-E and E-F and so the recursion goes on.

![Figure 3.1](image)

We now formulate the generic problem.

\[ x_{k+1} = f_k(x_k, u_k, w_k), k = 0, 1, \ldots, N - 1 \]

where

- **k**: discrete time,
- **x_k**: state of the system, which summarizes past information that is relevant for future optimization,
- **u_k**: the control variable (the decision to be selected at time \( k \) from a given set),
- **w_k**: the random parameter or disturbance,
- \( N \): the horizon.

An example of a basic stochastic optimization problem is

\[ \min_{u \in U} E_w \{ g(u, w) \}, \]

where \( g(x) \) is the cost function to be optimized. From this equation we derive the
The first term on the right refers to the one period cost in period $N$, and the second term is the recursion call to the function $g_k$ for the period $N - 1$. The algorithm used to solve such functional equations is not unique. Some of the common approaches are as follows [4].

- **Top-down approach**: the problem is broken into subproblems, and these subproblems are solved and the solutions remembered, in case they need to be solved again. In the top-down model an overview of the system is formulated, without going into detail for any part of it. Each part of the system is then refined by designing it in more detail. Each new part may then be refined again, defining it in yet more detail until the entire specification is detailed enough to validate the model.

- **Bottom-up approach**: All subproblems that might be needed are solved in advance and then used to build up solutions to larger problems. In bottom-up design, individual parts of the system are specified in detail, and then linked together to form larger components, which are in turn linked until a complete system is formed. Strategies based on the bottom-up information flow seem necessary and sufficient because they are based on the knowledge of all variables that may affect the elements of the system.

There also exist algorithms which combine the two approaches to get more optimal performance.

As mentioned, inventory problems are often expressed as functional (hence recursive) equations. For example, to find the optimal policy for $N$ time horizon, we must first find the optimal policy for $N - 1$ time horizon.
Chapter 4

Single-Installation Inventory System

In this chapter we review three single-installation inventory systems, characterize and then prove their optimal policies. We begin with some of the most basic inventory problems, the Newsvendor Problem and the Dynamic Inventory System with no time lag. The latter can be generalized to the case with positive time lag, as studied and solved by Scarf [22]. These results serve as the building blocks of the multi-installation decomposition approach in Chapter 5.

4.1 Newsvendor Problem

Consider the following problem. A vendor who is operating for a single trading period, is selling a product that becomes obsolete at the end of the period. The demand is modelled by a non-negative absolutely continuous random variable $\xi$ with probability density function $\varphi$ and cumulative distribution function $F$,

$$ P[\xi \leq x] = \int_0^x \varphi(y) dy = F(x), \quad x \geq 0. $$
The vendor is exposed to three kinds of costs. There is an ordering cost \( c \) per unit and a holding cost \( h \) per unit, charged for the stocks on hand at the beginning of the period. The penalty cost \( p \) per unit, is charged at the end of the period for any lost sales. The penalty cost is zero when there is enough stock to satisfy the demand during the course of the trading period.

Let \( z \) be the number of units ordered at the beginning of the period. We wish to minimize the expected total cost \( C(z) \), where

\[
C(z) = E[cz + hz + p \max\{0, \xi - z\}] .
\tag{4.1}
\]

Expressing \( E[\max\{0, \xi - z\}] \) as

\[
\int_0^z 0 \varphi(\xi) d\xi + \int_z^\infty (\xi - z) \varphi(\xi) d\xi ,
\]

we have

\[
C(z) = cz + hz + p \left[ \int_z^\infty \xi \varphi(\xi) d\xi - \int_z^\infty z \varphi(\xi) d\xi \right] = cz + hz + p \left[ \int_z^\infty \int_0^\xi d\xi \varphi(\xi) d\xi - \int_z^\infty \varphi(\xi) d\xi \right].
\]

Changing the integration order in the third term, we obtain

\[
C(z) = cz + hz + p \left[ \int_0^\infty \int_{\max(z,v)}^\infty \varphi(\xi) d\xi d\xi - \int_0^z (1 - F(z)) \right]
\]

To get the optimal value for \( z \), we evaluate the derivative \( C'(z) \) and solve the equation
\[ C'(z) = 0, \]
\[ c + h + p(-1 + F(z)) = 0. \]

The optimal policy is to order an amount \( z^* \) such that \( F(z^*) = \frac{p-c-h}{p} \). Since \( F \) is invertible, the optimal policy is thus unique. Also, a check on the second derivative \( C'' \),

\[ C''(z^*) = p(\varphi(z^*)) \geq 0, \quad (4.2) \]

shows that it is a minimum (since \( \varphi(z) \) is a probability density function).

The method and the result discussed can be modified to accommodate certain changes in the problem formulation. The Newsvendor problem is commonly formulated as a maximization problem; rather than minimizing his or her costs, the vendor is concerned with the optimal profit instead. As such, we are concerned with the trade off between the sales price \( s \) per unit and the product cost \( b \) per unit, where \( b < s \). A discussion on the method, results and the extensions to the cases with discrete demands and/or supply is presented in [21].

Intuitively, the two formulations are similar; both consider the trade off between the costs associated with over- or under- stock. The holding cost \( h \), used in our formulation, is not usually considered in a single-period inventory model, but it is not impractical in real life. Barancsi et al. [2] gives an example where this is applicable: products being held in the venue use up space that can otherwise be used to stock up other (profitable) items. In this situation, we might consider the inconvenience caused as a ‘holding cost’.

### 4.2 Dynamic Inventory Management

One of the defining properties of the Newsvendor problem is that the product becomes obsolete at the end of each order period. There are, however, situations in which the product retains its value for some periods of time.

We now consider a single-installation inventory system with finite time horizon \( N \in \mathbb{Z}^+ \).
There are variations in the formulation of this problem and extensive discussions on the results and features ([21], [2]). In this thesis, we base our formulation and assumptions on Scarf’s model [22].

4.2.1 The Case With No Time Lag

We consider a system with periodic reviewing. The decisions to purchase units are made at the end of each time period. There is no time lag in the delivery of purchases. Purchases arrive at the beginning of the next period and contribute to the build-up of inventories, which are depleted by demands during various periods. If demand exceeds stock, it must be satisfied in subsequent periods. The vendor wishes to minimize the expected discounted cost over \( N \) periods.

At the end of the time horizon, any inventory left is valueless. Moreover, we assume that the costs are zero. This assumption can be easily changed, to accommodate for situations where all backlogged demands must be satisfied.

**Mathematical Formulation.** We adopt the convention that variables with time index \( n \) relate to the period \( n \) time steps before the horizon \( N \). For example, if \( N = 5 \) days, then costs occurring on day 1 will have time index 5, and so on. We will use the following notation.

Demands during different periods are independent copies of the random variable \( \xi \) with pdf \( \varphi \) and cdf \( F \).

The inventory at the beginning of period \( n \) will be denoted by \( y_n \), and at the end, by \( x_n \). Hence we have the following relationship between \( x_n, y_n, \xi_n \):

\[
x_n = y_n - \xi_n, n = 1, 2, \ldots, N.
\]

(4.3)

Note that negative values of \( x_n \) signify a backlog of unfulfilled demand.
The purchasing cost of \( z > 0 \) units of inventory is \( c(z) \) for all periods.

The holding cost is \( h \) per unit for all periods. We adopt the convention that holding cost is realized at the start of each period, and hence is evaluated by multiplying \( h \) with the starting inventory \( y \).

The penalty cost is \( p \) per unit, incurred at the end of the period that ends with accumulated demand exceeding accumulated inventory. Note that the penalty cost is charged for backlogged demands, rather than for lost sales as in the Newsvendor problem.

Future costs are discounted at rate \( \alpha \in (0, 1] \); that is, a discount factor \( \alpha^m \) is applied to costs \( m \) time steps in the future.

Thus, the holding cost associated with a period which starts with inventory level \( y \) is given by

\[
h \cdot y^+ = h \left[ \max \{y, 0\} \right]
\]

and the penalty cost associated with a period that started with inventory level \( y \) and suffers demand \( \xi \) is given by

\[
p \left( (\xi - y)^+ \right) = p \left[ \max(\xi - y, 0) \right].
\]

The expected sum of holding and penalty costs associated with such a period, denoted by \( L(y) \), is

\[
L(y) = E \left[ h \left[ \max \{y, 0\} \right] + p \left[ \max(\xi - y, 0) \right] \right] = \begin{cases} hy + p \int_y^\infty (\xi - y) \varphi(\xi) \, d\xi & y > 0, \\ p \int_0^\infty (\xi - y) \varphi(\xi) \, d\xi & y \leq 0. \end{cases}
\]

(4.4)

The purchasing cost \( c(z) \) can take the form

\[
c(z) = \begin{cases} K + cz & z > 0, \\ 0 & z = 0, \end{cases}
\]

(4.5)
where $K > 0$ is the set up cost. For $K = 0$ we have the purchasing cost

$$c(z) = cz. \quad (4.6)$$

Assuming (4.5), the expected value of the discounted costs $C_n(x)$ will satisfy the functional equation

$$C_n(x_n) = \min_{z \geq 0} \{ c(z) + L(x_n) + \alpha \int_{0}^{\infty} C_{n-1}(x_n - \xi) \varphi(\xi) d\xi \}, \quad (4.7)$$

$$C_n(U_n) = \min_{z \geq 0} \{ c(z) + L(x_n) + \alpha \int_{0}^{\infty} C_{n-1}(U_n - \xi) \varphi(\xi) d\xi \}, \quad (4.8)$$

with $C_0(x_0) = 0$. Considering only the decision variable $y$, we get a new functional equation

$$G_n(y) = c(y) + L(y) + \alpha \int_{0}^{\infty} C_{n-1}(y - \xi) \varphi(\xi) d\xi. \quad (4.9)$$

At this point, the optimal decision is found by minimizing $G_n(y)$ over the set $y_n \in [x_{n-1}, \infty)$.

**Algorithmic Issues.** We extract Algorithm 1 from [22] to solve for $y_n$. Given $C_0(x_0)$ we can recover the solutions $y_n$, $n = 1, 2, \ldots, N$ recursively.

<table>
<thead>
<tr>
<th>Algorithm 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Step 0</strong> Set $n = 1$ and $C_0(x_0) = 0$.</td>
</tr>
<tr>
<td><strong>Step 1</strong> Minimize $G_n(y)$ to get the optimal $y_n^*$, over the set $y \geq x$.</td>
</tr>
<tr>
<td><strong>Step 2</strong> Find $C_n(x)$ using $C_n(x) = G_n(y) - c(x)$.</td>
</tr>
<tr>
<td><strong>Step 3</strong> Set $n = n + 1$. Return to step 1</td>
</tr>
</tbody>
</table>

The algorithm stops when $n = N + 1$.

**Example.** Let us consider a problem with $N = 2$. The set up cost is $K = 0$, the holding cost is $h = 1$ per unit and the penalty is $p = 3$ per unit backlogged. $\alpha = 0.9$ and the demand $\xi$ has a discrete uniform distribution $(0, 1, 2, 3)$ for all time period. The full
CHAPTER 4. SINGLE-INSTALLATION INVENTORY SYSTEM

Given $C_0(x_0) = 0$, we have $EC_0(x_0) = 0$. In Step 1, we consider $y \in [-5, 5]$ to find the minimum of $G_1(y)$ in (4.9). The minimum is found at $y_1^* = 0$ giving $G_1(0) = 4.5$ (Appendix A.1).

![Graph of the function $L(y)$ with $y \in [-5, 5]$. The minimum of $(L(y) + cy)$ gives the optimal $G_1(y)$ (since $EC_0(x_0) = 0$).](image)

In Step 2, we need to find $C_1(x)$. This is a function of $x_1$, given the optimal $y_1$. The table in Appendix A.2 gives the value of this function for $x \in [-8, 5]$. We now consider $n = 2$. The above procedure is repeated. We have the minimizing $y_2^* = 2$. Suppose that our initial stock $x_2 = -5$, we have to order $z_2 = 7$ units to bring the stock to the optimal level for that day.

**Characterization of the Optimal Policies** A policy for this problem is defined as a function or decision rule whose input is end of period inventory level $x_n$, and output is the stock required at the start of the next period $y_{n+1}$.
**Definition 1** A policy $\delta(x_n)$ is called a base-stock policy if, for the pair $(S_{n+1}, s_{n+1})$,

$$
\delta(x_n) = \begin{cases} 
S_{n+1} & \text{if, } x_n \leq s_{n+1}, \\
x_n & \text{otherwise.}
\end{cases}
$$

It is shown in [22] that the optimal policy of a dynamic inventory system is always of the $(S_n, s_n)$ type, provided the following sufficient conditions hold:

- the function $L(x)$ is convex,
- the ordering cost is of the form (4.5).

When the set-up cost $K$ is zero (equation 4.6), the two levels $s_n$ and $S_n$ collapse into a single threshold value $\bar{x}_n$ (Figure 3.1b).

![Figure 4.2: a) Under a base-stock policy, we stock up to $S$ when $x$ is on the shaded region. b) There is no set-up cost $K$ incurred when orders are placed. We order whenever the stock level is less than $X = S_n = s_n$](image)

Observe that we have not recovered $(S_n, s_n)$ from Algorithm 1. Recovering the base-stock levels from our output is not straightforward. Consider the following cases.

$x_n < y^*_{n+1}$. In this case we would order, and thus $S_n = y_{n+1}$.

$x_n = y^*_{n+1}$. We do not order, and so the result is inconclusive. All we can tell is that $S_{n+1} \leq x_n$. 


One way to recover the $S_n$ level is to consider the unconstrained minimization of equation (??) instead. Notice that in this case, $y_{n+1}$ is allowed to be less than $x_n$. An alternative method is to use Algorithm 1 with the input $x_N$ set to be as low as possible. Here, case one always applies and so we can retrieve $S_n$. This will be further discussed in Section 3.2.2.

Given $S_n$, $s_n$ is calculated as the value of $y < S_n$ for which

$$G_n(s_n) = G_n(S_n) + K.$$  

It is clear from Figure 4.2 that when $G_n$ is strictly convex, there must exist a unique minimum and hence a basestock policy.

Scarf [22] noticed that $G_n$ might in fact have several minima and maxima. To handle this, he used the notion of $K$-convexity.

**Definition 2** Let $K \geq 0$, and let $f(x)$ be a differentiable function. We say that $f(x)$ is $K$-convex if $\forall a > 0, \forall x$,

$$f(a + x) \geq -f(x) + af'(x) - K.$$  

When differentiability is not assumed, $\forall a, b > 0, \forall x$,

$$K + f(a + x) - f(x) - a\left[\frac{f(x) - f(x - b)}{b}\right] \geq 0.$$  

$K$-convexity has the following properties:

- When $K = 0$, $K$-convexity reduces to an ordinary convexity.
- If $f(x)$ is $K$-convex, then $\forall h$, $f(x + h)$ is $K$-convex.
- If $f$ and $g$ are $K$-convex and $M$-convex respectively, then $\alpha f + \beta g$ is $(\alpha K + \beta M)$ convex when $\alpha, \beta > 0$. 

21
A $K$-convex function is allowed to depart from convexity by dipping below its tangents, but never more than $K$. With $K$-convexity, an $(S_n, s_n)$ type policies are still optimal for the system. Scarf [22] proved this by showing that the function $G_n$ is $K$-convex when $L(x)$ is convex, (4.5) holds, and the function $G_n$ is differentiable. We generalize Scarf’s argument by relaxing his assumption on $G_n$ being differentiable.

We first illustrate how $K$-convexity ensures the existence and optimality of base-stock policies. Consider Figure 4.3. The condition for $K$-convexity holds for the function $f$.

![Figure 4.3: A K-convex function f. The base-stock policy exists and is optimal.](image)

We now define $S_n$ to be the global minimizer over all real $y_n$.

- If $x_{n-1} < S_n$, the minimizing $y_n \geq x_{n-1}$ is $y_n = S_n$.
- If $x_{n-1} \geq S_n$, the minimizing $y_n$ is $x_{n-1}$.

Hence the minimizing policy is a base-stock policy.

Note that at inventory level $A$, we might be expected to stock up to level $B$ that gives a (local) minimum cost. Even though $f$ decreases, it does not decrease by enough to outweigh the fixed cost $K$. The $(S_n, s_n)$ policy holds.
Figure 4.4: If we take any point between $C$ and $D$, and check it against point $S'$, the condition for $K$-convexity is violated.

We now consider a function $f$ that is not $K$-convex (Figure 4.4). Note that for all points between $[C, D]$, it is optimal to stock up to the level $S'$. The existence of another minimizer $S'$ contradicts our base-stock policy. Having seen the motivation for $K$-convexity, we now prove that the function $G_n$ is $K$-convex by induction. Scarf [22] proved this by assuming that $G_n(y)$ was differentiable. Since it was neither proven nor obvious that $G_n(y)$ is differentiable, we generalize the proof by not assuming differentiability.

**Initialization** $G_1(y) = cy + L(y)$. Since $L(y)$ is 0-convex, $G_1$ is 0-convex, and by property (i) is thus $K$-convex.

**Induction** Assume that $G_2, \ldots, G_n$ are $K$-convex. We need to show that

$$G_{n+1}(y) = cy + L(y) + \alpha \int_0^\infty C_n(y - \xi) \varphi(\xi) d\xi$$  \hspace{1cm} (4.12)

is $K$-convex too. The first two terms are 0-convex. By applying properties (ii) and (iii) to the last term, we only need to check the $K$-convexity of $C_n(y)$ to conclude
From the relationship between $C_n(y)$ and $G_n(y)$ and the meaning of an $(S_n, s_n)$ policy, we have

$$C_n(y) = \begin{cases} K + c(S_n - y) + C_n(S_n) = K - cy + G_n(S_n) & y < s_n, \\ -cx + G_n(y) & y \geq s_n. \end{cases}$$

(4.13)

Recall our objective: given a convex function $G_n$, we want to show that the function $C_n$ as defined in (4.13) satisfies

$$K + f(a + x) - f(x) - a \left[ \frac{f(x) - f(x - b)}{b} \right] \geq 0.$$

Consider the three cases as follows.

**Case One: $y > s_n$.**

$$C_n(y) = -cx + G_n(y).$$

The first term is linear and the second is $K$-convex. The function $C_n$ is therefore $K$-convex.

**Case Two: $y < s_n < y + a$.** From (4.13), we have

$$C_n(a + y) - C_n(y) = [K - c(y + a) + G_n(y + a)] - [-c(y) + G_n(y)]$$

$$= K + G_n(y + a) - G_n(y) - ca.$$

(4.14)

$$C_n(y) - C_n(y - b) = [-c(y) + G_n(y)] - [-c(y - b) + G_n(y - b)]$$

$$= G_n(y) - G_n(y - b) - cb.$$

(4.15)
Thus

\[ K + [C_n(a + y) - C_n(y)] - \frac{a}{b} [C_n(y) - C_n(y - b)] \]

\[ = K + [K + G_n(y + a) - G_n(y) - ca] - \frac{a}{b} [G_n(y) - G_n(y - b) - cb] \]

\[ = K + K + G_n(y + a) - G_n(y) - \frac{a}{b} [G_n(y) - G_n(y - b)] \]

\[ > K + G_n(y + a) - G_n(y) - \frac{a}{b} [G_n(y) - G_n(y - b)] \]

\[ \geq 0. \]

The function \( C_n \) is therefore \( K \)-convex.

**Case Three:** \( y + a < s_n \) From (4.13), we have

\[ C_n(a + y) - C_n(y) = [-c(y + a) + G_n(y + a)] - [-c(y) + G_n(y)] \]

\[ = G_n(y + a) - G_n(y) - ca. \] (4.16)

Repeating the same steps as in Case Two, but substituting (4.14) by (4.16), we have

\[ K + [C_n(a + y) - C_n(y)] - \frac{a}{b} [C_n(y) - C_n(y - b)] \]

\[ = K + [G_n(y + a) - G_n(y) - ca] - \frac{a}{b} [G_n(y) - G_n(y - b) - cb] \]

\[ = K + G_n(y + a) - G_n(y) - \frac{a}{b} [G_n(y) - G_n(y - b)] \]

\[ \geq 0. \]

The function \( C_n \) is therefore \( K \)-convex.

This completes the induction and demonstrates the optimality of \((S_n, s_n)\) policies. \( \square \)

### 4.2.2 The Case With Time Lag

**Mathematical Formulation.** We generalize the model described in the previous section. There is a **time lag** \( \lambda \geq 0 \) in the delivery of purchased goods: stock ordered in period \( n \) will only arrive in period \( n - \lambda \). The assumptions and definitions for the demand
random variable $\xi$, holding cost $h$, purchase cost $c$, penalty cost $p$ and discount factor $\alpha$ are the same as in Section 4.2.1. Instead of describing the system by $x_n$ and $y_{n+1}$ however, we have the following variables.

The amount of **stock on hand** at the beginning of period $n$ is $x_n$.

The **stock in transit**, $w_m$, ($\forall m = 1, \ldots, \lambda - 1$) is to come $m$ period(s) away from the present time.

The **total stock configuration** $u_n$ at the beginning of period $n$ is given by:

$$u_n = x_n + \sum_{m=1}^{\lambda-1} w_m.$$

Note that the period index $n$ is suppressed for $w$.

The **order quantity** $z_n$ is the control variable to be decided at the end of period $n$.

We begin a period $n$ with a given stock configuration $x_n, w_1, \ldots, w_{\lambda-1}$. Having observed the demand $\xi_n$, we wish to decide an order amount $z_n$. As a consequence of our decision, we begin the next period $n-1$ with a new stock configuration:

$$x_{n-1} = x_n - \xi_n + w_1,$$

$$w_{m-1} = w_m, \quad \forall m = 2, \ldots, \lambda$$

$$w_\lambda = z_n.$$ 

The procedure is repeated until we reach the end of the time horizon, in which all excess stock and backlogged demands are set to zero. A numerical example is provided to illustrate the dynamic of the variables (Figure 4.5).

Consider a vendor selling tickets for a concert five days ahead. If he orders more tickets from the supplier, they will be delivered in 4 days time.

In $n = 5$ days away from the concert, the vendor has 10 tickets on hand, 3 to be delivered tomorrow, 2 the following day and so on. $u_n$ is the the sum of all stocks on hand and in
transit. Having observed a demand of \( t_5 = 14 \) during the day, he places an extra order of \( z_5 = 8 \) units. He begins the next day with the leftover stock from yesterday plus the

<table>
<thead>
<tr>
<th>Variables</th>
<th>Period n</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stock level</td>
<td>( x(n) )</td>
</tr>
<tr>
<td>Stocks in transit:</td>
<td>( w_1 )</td>
</tr>
<tr>
<td></td>
<td>( w_2 )</td>
</tr>
<tr>
<td></td>
<td>( w_3 )</td>
</tr>
<tr>
<td>Order amount</td>
<td>( z(n) )</td>
</tr>
<tr>
<td>Total stock</td>
<td>( u(n) )</td>
</tr>
<tr>
<td>Demand</td>
<td>( t(n) )</td>
</tr>
</tbody>
</table>

Figure 4.5:

stock that just arrives, \( w_1 = 3 \). Note that among the stocks in transit, he is now waiting for \( w_3 = z(5) = 14 \) tickets to be delivered in 3 days. Regardless of the demand \( t_4 \), the vendor would not place any order today. Tickets arriving in period \( n = 0 \) could not be sold.

Similarly for \( n = 3, 2, 1, z(n) = 0 \). On these days, he could only wait for the arrivals of tickets as his decisions would not increase his sales. Note that so far we have not specified the optimization problem. The expected discounted total cost is a function of \( x \) and \( w_m \):

\[
C_n(x_n^2, w_1^2, \ldots, w_{\lambda-1}^2) = \min_{z^2 \geq 0} \left\{ c(z^2) + \int_{0}^{\infty} C_{n-1}(x_n^2 + w_1^2 - \xi, w_2^2, \ldots, w_{\lambda-1}^2, z^2) \varphi(\xi) d\xi + \Delta(\hat{x}^2) \right\}.
\]

\[
C_n(x_n^1, w_1^1, \ldots, w_{\lambda-1}^1) = \min_{z^1 \geq 0} \left\{ c(z^1) + \int_{0}^{\infty} C_{n-1}(x_n^1 + w_1^1 - \xi, w_2^1, \ldots, w_{\lambda-1}^1, z^1) \varphi(\xi) d\xi \right\}.
\]

We can reduce the computational work substantially by considering only the periods affected by the ordering and the arrivals of the units. It can be seen from Figure 4.5
that for all periods \( n < \lambda + 1 \), we do not make any more decisions. Orders placed during these periods will not contribute to the system. With this insight, Scarf [22] decomposed (4.16) into:

\[
C_n(x_N, w_1, \ldots, w_{\lambda-1}) = L(x_1) + \alpha \int_0^\infty L(x_N + w_1 - \xi) \varphi(\xi) d\xi + \ldots + \alpha^{\lambda-1} \int_0^\infty \cdots \int_0^\infty L(x_N + w_1 + \ldots + w_{\lambda-1}) \nonumber
\]

\[
+ w_{\lambda-1} - \xi_N - \ldots - \xi_N - \alpha \int_0^\infty \cdots \int_0^\infty L(\xi_N - \ldots - \xi_N - \lambda) \varphi(\xi_N) \ldots \varphi(\xi_N - \lambda) d\xi_N \ldots d\xi_N - \lambda + f_n(x_N + \ldots + w_{\lambda-1}),
\]

where

\[
f_n(u) = \min_{y \geq u} \{ c(y - u) + \alpha^{\lambda-1} \int_0^\infty \cdots \int_0^\infty L(y - \xi_N - \ldots - \xi_N - \lambda) \varphi(\xi_N) \ldots \varphi(\xi_N - \lambda) d\xi_N \ldots d\xi_N - \lambda + \alpha \int_0^\infty f_{n-1}(y - \xi) \varphi(\xi) d\xi \},
\]

Note that \( C_n \) is separated into two parts: the calculation of the costs which cannot be modified under the given input and \( f_n(u) \). The latter calculates the costs that are affected by our decisions, and so must be optimized.

More importantly, however, we have reduced the minimization problem of multiple variables \((x_n, w_m)\) into one with a single variable \( u_n \). \( f_n(u) \) describes a system of zero time lags, analogous to equation (4.22) in the previous section.

The only difference is in the second term, which is now replaced by

\[
\alpha^{\lambda-1} \int_0^\infty \cdots \int_0^\infty L(y - \xi_N - \ldots - \xi_N - \lambda) \varphi(\xi_N) \ldots \varphi(\xi_N - \lambda) d\xi_N \ldots d\xi_N - \lambda.
\]

For a dynamic inventory system with \( \lambda \) time lag, the optimal policies are always in the form of base-stock policies [22]. There exists the levels \((S_n, s_n)\), such that we order from \( u_n = (x + w_1 + \ldots + w_{\lambda-1}) \) up to level \( S_n \), when \( u_n < s_n \). Otherwise we do not order.

Assuming that \( L(y) \) is convex, the expression (4.23) is convex. This is sufficient for the arguments in section 3.2.1 to hold [22]. We establish the proof for Scarf’s claim above as
follows.

**Lemma 1** Let \( \xi \) be a random variable with density \( f \). Suppose that \( L(y) \) is convex, i.e.,
\[
L(y+z) \leq L(y) + z \frac{\partial}{\partial y} L(y).
\]
Then \( H(y) = EL(y-\xi) = \int L(y-m)f(m)dm \) is also convex.

**proof.**
\[
H(y+z) = EL(y+z-\xi) = \int L(y+z-m)f(m)dm
\]
\[
\leq \int [L(y-m)+z \frac{\partial}{\partial y} L(y-m)]f(m)dm
\]
(since the integrand of the second integral is bigger for all values of \( y \))
\[
= \int L(y-m)f(m)dm + z \int \frac{\partial}{\partial y} [L(y-m)f(m)]dm
\]
\[
= \int L(y-m)f(m)dm + z \frac{\partial}{\partial y} \int L(y-m)f(m)dm
\]
\[
= H(y) + z \frac{\partial}{\partial y} H(y).
\]

This can easily be extended to the case where \( \xi \) is a sum of random variables, and the integration is with respect to their joint density. Alternatively, if \( L(y-\xi_1-\ldots-\xi_n) \) is convex in \( y \), then fixing \( \xi_2,\ldots,\xi_n \) and integrating with respect to the density of \( \xi_1 \) gives a convex function of \( y \). Releasing \( \xi_2,\ldots,\xi_n \) one by one and integrating with regard to their respective densities (induction) gives the result.

The existence and optimality of base-stock policies are thus established. \( \square \)

**Algorithmic Issues.** In this section we present the results from coding Scarf’s functional equation in Matlab. There are two variables that can be used to describe the vendor’s decision:

\( y_n \): the level of stock after ordering and is found by solving equation (4.18),

\( z_n \): the order amount and is found by solving equation (4.16).
We get the optimal decision by solving either equation, and we can get one from the other via their relationship: \( y_n = z_n + u_n \).

We have based our code on equation (4.16), using a top down recursion method. At each time period, we arbitrarily choose a value \( z_n \) until we reach the end of the horizon. Different choices \( z_n \) will give different costs, which will then be minimized. For simplicity we try values of \( z_n \) increasing from 0 to a maximum. Since \( C_n \) is a convex function, we get decreasing costs as we increase \( z_n \), until the minimum is reached. The stopping criteria is when the cost starts increasing. This is the optimal order quantity \( z_n \).

Consider a problem of minimizing the cost over 3 time horizon (Figure 4.6). At \( n = 3 \), we arbitrarily choose \( z_3 = 0 \). This result in a certain stock configuration at time \( n = 2 \). And so at time stages \( n = 2, 1 \), we arbitrarily choose values of \( z \) as well, going from 0 to the maximum (here it is set to be 2). Once we recover the costs for nodes 3, 4, 5, we can take the minimum of these values as the optimal cost for node 2. Similarly, the optimal value for node 1 is decided by the minimum between node 2, 6, and so on. This recursion can be considered as a variety of the top-down method discussed in Chapter 3.

![Figure 4.6](image-url)
Consider $N = 4$, $\xi \sim Pn(7)$, $\lambda = 2$. Column 2 tells us the optimal order quantity, $z_n$, given $u_4$. The rest of Column 1 thus becomes the consequence of $u_4$ and Column 2. We wish to compare the last column $u_n + z_n$ to the levels $S_n$.

An alternative formulation will be the bottom-up method, using equation (4.18) instead. Here, we start from period $n = 1$ and calculate the costs for all possible starting stock $u$ for this period. Once we store all these possibilities, we can go to period $n = 2$ and calculate the costs, using values stored from period $n = 1$. When we get to the initial time period $n = 3$, we can take the minimum of all costs since we have calculated all possible stock configurations. Note that this method is greatly simplified since we only need to consider the stock configuration $u$ rather than $x, w$ as before.

As such, we realize that the top-down algorithm we have implemented might not be the most efficient. It is, however, sufficient to give us desirable results to be compared with the multi-installation models in Chapter 5. Also, the results can be used to further study the two questions we have presented before: (4.23)

1) What is the relationship between the base-stock levels $(S_n, s_n)$ and $z_n$? (4.24)

2) Having found $z_n$ from Algorithm 2, can we easily retrieve the base-stock levels?

Figure 4.7 gives the result from the Matlab implementation of Algorithm 2. We shall use these results to illustrate the method of retrieval of the levels $S_n$. In Section 4.2, we consider $y_n$ instead of $z_n$. As mentioned, the analysis of one can be easily applied to the other.

**Step 1:** Analyze the problem before Algorithm 2 starts. If there are any periods $n \leq \lambda$, we set $S_n$ for such $n$ to equal zero. We do not want to order any items that will reach the system outside the time horizon. As such, we can ignore periods 2 and 1.
altogether (in Figure 4.7).

**Step 2:** For all $n > \lambda$, consider the two cases as before:

a: if $z_n > 0$, we order the amount $z_n$ to bring the level $u_n$ to $y_{n+1} = u_n + z_n$. In this case, $S_{n+1}$ is equal to $y_{n+1}$.

b: if $z_n = 0$, we do not order. We have $y_{n+1} = u_n$. The result is inconclusive and we can only deduce that $S_{n+1} \leq u_n$.

In summary, $y_n(u)$ is a function of $u$ such that $y_n(u) = x_n + u_n$ whereas $S_n$, is constant for any initial stock configuration $u$.

We see in Figure 4.7 that $z_4, z_3 \neq 0$ and so we can deduce that the base-stocks $S_4 = 29, S_3 = 29$. The base-stocks in the other periods can be set to 0 since we cannot make any decisions here.

Consider Figure 4.8. For cases A and D, we start the period with low stock levels. Hence we always have $z > 0$ and situation (a) holds. Set $S_n = u_n + z_n$ and note that basestock levels $S_n$ in the two cases are the same. As a consequence of (b), we cannot retrieve $S_n$

<table>
<thead>
<tr>
<th>$n$</th>
<th>Total stock $u(n)$</th>
<th>Order amount $z(n)$</th>
<th>$u(n) + z(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>8</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>29</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.8: $N = 4, \xi \sim Pn(7), \lambda = 2$. Now we consider 4 cases: A, B, C, D. Each case has a different initial stock configuration.

when $z = 0$. In cases B, C, we begin with very high stock levels and so the column $u_n + z_n$ gives us the total excess stock we have, rather than the basestock levels.

It is thus not a problem if we get $z = 0$ from our algorithm. As seen, if we want to retrieve $S_n$, we can just begin our program with $u_n$ as low as possible (Case D). Repeat
the procedure for 'lower' periods if necessary.

This analysis to retrieve the base-stock levels \((S_n, s_n)\) is applied to our results when we are comparing this model with Tsitkilis and Muharremoglu inventory model in Section 5.2.
Chapter 5

Multi-echelon Inventory System

Similar analysis and methods from single-installation models can be applied to a $J$-installation system. In practice, however, the procedure will necessitate the recursive computation of a sequence of functions of at least $J$ variables.

This motivates the search for more efficient ways to handle such complex systems. Clark and Scarf in 1960 [8] proved the optimality of a decomposition approach: they decomposed multi-installation systems into a family of single-installation subproblems, each being an optimization problem of a single variable. Recently, Tsitkiliis and Muharrremoglu [18] proposed a different decomposition approach- a single unit single customer decomposition- and claimed the new method to be more versatile and efficient than its predecessors.

We present and study both models in details. A comparison of the algorithms and numerical examples are given at the end of the chapter.
5.1 Classical Decomposition of Multi-Echelon Systems

5.1.1 Introduction

Consider a serial system of $J$ installations, with installation $j - 1$ receiving stock from installation $j$ for all $j = 2 \ldots J$. Demand originates in installation 1 and installation $J$ receives stock from the outside supplier. Denote the time lag by $\lambda_n$, which may vary in different periods. To simplify the notation, however, we assume it to be constant and suppress the index $n$. The delivery of stock in an installation is affected by the time lag as well as the availability of stock at the higher installation.

The general approach of the Clark and Scarf decomposition method is as follows. We break up the serial system and treat one installation at one time, starting with installation 1 and moving upwards. Each installation can be modelled as a dynamic inventory problem discussed in Section 3.2.2 with slight modification to the definition of the costs. The key idea here is that at one time, we only need to optimize the order amount in one installation, hence decomposing the multi-installation problem into a family of single-variable optimization problems.

We first discuss the problem assumptions and formulation, provide a sketch of the algorithm and lastly, outline the proof of optimality presented in [8].
5.1.2 Problem Formulation

The same conditions and definitions in Section 4.2.2 apply as when we are considering a single-installation problem.

**Definition 3** An echelon stock refers to the stock on hand at a given installation, plus all the stock in transit to as well as on hand at the lower installation.

\[
\hat{x}_j = \begin{cases} 
  x_j & , j = 1 \\
  x_j + x_{j-1} + \sum_{m=1}^{\lambda-1} w_m & , otherwise
\end{cases}
\]  

(5.1)

where \(x_j\) is just the stock on hand at \(j\).

The echelon is numbered according to the highest installation present. By recursion, echelon stock \(j\) is a function of the stocks in and in between all installations \((j, j-1, \ldots, 1)\) for \(j \geq 3\). From here on, we shall use the notion echelon instead of installation wherever applicable.

We make the following assumptions:

**Assumption TC:** The transportation cost from one installation to the other must be linear (4.6). A set up cost \(K\) is only permitted in the purchase cost applicable to installation \(J\).

**Assumption LC:** The linear holding and backlogging costs \(L(x)\) in installation one is as described in (4.4). At installation \(j > 1\), however, \(L(.)\) becomes a function of the echelon stock, that is the stock at installation \(j\), the stock in transit to installation \((j-1)\) and the stock on hand at installation \((j-1)\).

**Definition 4** The natural one-period cost at any installation refers to \(L(.) \geq 0\), a function of the echelon stock at that installation.

The decomposition of a multi-echelon system into single-installation problems is justified by the following modifications:
• For all $j < J$, we regard the shipment cost from echelon $j$ to $j - 1$ as the purchase cost $c$. Note the restriction caused by Assumption (TC) in this case. The actual purchase cost of the product is considered only for the highest echelon $j = J$.

• For all $j$, the demands for echelon $(j + 1)$ are the optimal order amounts $z_n$ from echelon $j$. As for echelon one, the random variable $\xi \sim F(x)$ models the demand as per normal. Since $z_n$ is a function of $x_n$ and $w_m$, we have in fact incorporated the stock at echelon $j$ into our optimization for $(j + 1)$. With this formulation, Assumption (LC) is satisfied.

The multi-installation system is thus formulated as a single-installation problem for each $n$, as described in Section 4.2.2. To complete the transformation, we have to represent the relationship between available stock and threshold level in any two echelons. Except for the highest echelon $J$- which is connected to the supplier with unlimited supply- each echelon has a finite number of stock on hand. That is, the optimal order quantity in echelon $j$ might not be satisfied if there is not enough stock in echelon $j + 1$ to send over. For echelons greater than one, a penalty cost term is introduced.

To illustrate this, we construct a small example concerning two echelons, with a time lag of 2 units in the delivery of stock in between the echelons. Considering only installation one, we recall from Section 4.2.2 that $C_n(x_1, w_1)$ is the minimum expected discounted cost, if there are $n$ periods remaining.

Recall that

• $\bar{x}_j$ is the optimal threshold level at echelon $j$,

• $\hat{x}_j$ is the echelon stock at echelon $j$,

• $x_j$ is the number of stock on hand at installation $j$.

For simplicity, we suppress the time index $n$ for $x$, $\hat{x}$, $\bar{x}$ and show the echelon index $j$ instead. It should be clear that when we are describing the dynamic between two
chapter 5. Multi-Echelon Inventory System

Solving this system of equations, using Algorithm 2, we recover the echelon threshold level $\bar{x}_1$ for all time periods. By the definition, when $x_1 + w_1 < \bar{x}_1$, we would order and the minimum cost will be

\[
c(\bar{x}_n - u) + \alpha^2 \int \int L(\bar{x}_n - u - \xi_1 - \xi_2)\varphi(\xi_1)\varphi(\xi_2)d\xi_1d\xi_2 + \alpha \int_0^\infty f_{n-1}(\bar{x}_n - u - \xi)\varphi(\xi)d\xi. \tag{5.4}
\]

plus some non-optimizing terms in $C_n(x_1, w_1)$ which are independent of the order quantity at the current period. We now move up to echelon two, noticing that there is an echelon stock $\hat{x}_2$ and there is the level $\bar{x}_1$ which must be maintained in echelon one. If $\hat{x}_2 < \bar{x}_1$, we can only ship all the stock on hand, and so the actual cost in echelon one will be

\[
c(\hat{x}_2 - u) + \alpha^2 \int \int L(\hat{x}_2 - u - \xi_1 - \xi_2)\varphi(\xi_1)\varphi(\xi_2)d\xi_1d\xi_2 + \alpha \int_0^\infty f_{n-1}(\hat{x}_2 - u - \xi)\varphi(\xi)d\xi. \tag{5.5}
\]

plus some non-optimizing terms in $C_n(x_1, w_1)$. 

Solving this system of equations, using Algorithm 2, we recover the echelon threshold level $\bar{x}_1$ for all time periods. By the definition, when $x_1 + w_1 < \bar{x}_1$, we would order and the minimum cost will be

\[
c(\bar{x}_n - u) + \alpha^2 \int \int L(\bar{x}_n - u - \xi_1 - \xi_2)\varphi(\xi_1)\varphi(\xi_2)d\xi_1d\xi_2 + \alpha \int_0^\infty f_{n-1}(\bar{x}_n - u - \xi)\varphi(\xi)d\xi. \tag{5.4}
\]

plus some non-optimizing terms in $C_n(x_1, w_1)$.
Since $\bar{x}_1$ gives the optimal total cost, the expression (5.4) is always smaller than the expression (5.5). The penalty cost to be introduced in echelon two is thus:

$$c(\hat{x}_2 - \bar{x}_n) + \alpha^2 \int \int [L(\hat{x}_2 - u - \xi_1 - \xi_2) - L(\bar{x}_n - u - \xi_1 - \xi_2)]\varphi(\xi_1)\varphi(\xi_2)d\xi_1d\xi_2$$

$$+ \alpha \int_0^\infty [f_{n-1}(\hat{x}_2 - u - \xi) - f_{n-1}(\bar{x}_n - u - \xi)]\varphi(\xi)d\xi,$$

if $\hat{x}_2 < \bar{x}_n$ and zero if $\hat{x}_2 \geq \bar{x}_n$.

We summarize the problem formulation as follows. For echelon one, solve the functional equation (5.2) to get $\bar{x}_1$. For echelon two, solve $\min \{ C_n(x_2, w_1) + \text{term (5.6)} \}$ to get $\bar{x}_2$. For cases where $j > 2$, we apply the same analysis.

### 5.1.3 Algorithm and Optimal Policy

Let $w_j^m$ be the stock in transit, to come in $w$ period away to echelon $j$. We provide the sketch of the algorithm, illustrating the logical sequence of the steps taken to solve a multi-installation problem:

<table>
<thead>
<tr>
<th>echelon: $j$</th>
<th>steps:</th>
<th>considering a period $n$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$j = 1$</td>
<td>Input:</td>
<td>stock level $x_1$, stock in transit $\sum_{m=1}^{\lambda-1} w_m^1$, demand $\xi$</td>
</tr>
<tr>
<td></td>
<td>Optimization steps:</td>
<td>Solve (5.2) to obtain $y$, the 'order up' to level, and hence the order amount.</td>
</tr>
<tr>
<td></td>
<td>Output:</td>
<td>order amount $z_1$</td>
</tr>
<tr>
<td></td>
<td>Optimal threshold:</td>
<td>$\bar{x}_1$</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>( j = 2 )</th>
<th>Input:</th>
<th>stock level ( x_2 ), stock in transit ( \sum_{m=1}^{\lambda-1} w_m^2 ), demand ( \xi = z_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization steps:</td>
<td>1. Calculate ( \hat{x}_2 = x_2 + x_1 + w_m^1 )</td>
<td>2. Solve equation ( { (5.2) + (5.6) } ): [ \min { C_n(x_2, w_m) + \text{penalty term} } ]</td>
</tr>
<tr>
<td>Output:</td>
<td>order amount ( z_2 )</td>
<td>Optimal threshold: ( \bar{x}_2 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( j &gt; 2 )</th>
<th>Input:</th>
<th>stock level ( x_j ), stock in transit ( \sum_{m=1}^{\lambda-1} w_m^j ), demand ( \xi = z_{j-1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization steps:</td>
<td>1. Calculate ( \hat{x}<em>j = x_j + x</em>{j-1} + w_m^{j-1} )</td>
<td>2. Solve equation ( { (5.2) + (5.6) } ): [ \min { C_n(x_j, w_m) + \text{penalty term} } ]</td>
</tr>
<tr>
<td>Output:</td>
<td>order amount ( z_j )</td>
<td>Optimal threshold: ( \bar{x}_j ) or ((S_j, s_j)) when ( j = J )</td>
</tr>
</tbody>
</table>

The above procedure is done in every period \( n \). Note that solving the functional equation in each echelon is equivalent to solving a single-installation problem as described before. At echelon two, for example, the functional equation is a minimization of \((x_2, w_m)\) only. The level \( \bar{x}_1 \) and the demand \( \xi = z_1 \) have been decided in the calculation for echelon one. Lastly, to recover the threshold level(s) from the order amount, we refer the reader to the analysis done in section 4.2.2.

The decomposition method and Assumption \( (TC) \) result in the following optimal policy characterization:

- For echelon \( J \), there exists an echelon base-stock levels \((S_n, s_n)\).
• For \( j \neq J \), there exists the echelon threshold level \( \bar{x}_n \). It is not obvious to us why the optimal policy in these echelons should be restricted by Assumption (TC). It seems that a relaxation of this condition will not hinder any steps in the above algorithm.

### 5.1.4 The Proof of Optimality

To prove the optimality of the decomposition method, we need to show that the formulation of the multi-variable problem is equal to the formulation of the modified subproblems. We forget about the decomposition approach for a moment, re-formulate the problem as a multi-variable functional equation, and show that the latter is equal to what we have. For simplicity, we consider two installations. This can be extended to cases with \( j > 2 \).

Let \( C_n(\hat{x}_1, w_m, \hat{x}_2) \) be the minimum expected value of the discounted system costs if there are \( n \) periods remaining, \( \hat{x}_1 \) stock on hand at echelon 1, \( w_1 \) stock in transit between the two and \( \hat{x}_2 \) stock on hand at echelon 2. There are two decisions to be made:

• how much stock to order from the outside supplier, denoted by \( z \),

• how much stock to be sent from echelon two to one, denoted by \( y - (\hat{x}_1 + w_1) \),

where \( y \in [\hat{x}_1 + w_1, \hat{x}_2] \).

As a consequence of making the two decisions above, we have the cost in the next period

\[
C_{n-1}(\hat{x}_1 + w_1 - \xi, y - \hat{x}_1 - w_1, \hat{x}_2 + z - \xi),
\]

and the purchase and transportation costs

\[
c(z) + c_1(y - \hat{x}_1 - w_1),
\]

where \( c_1(.) \) is the transportation cost from echelon two to one.
There is also the holding costs $\bar{L} (\hat{x}_2) + L (\hat{x}_1)$. (5.6)

Thus,

$$
C_n (\hat{x}_1, w_m, \hat{x}_2) = \min_{\hat{x}_1 + w_1 \leq y \leq \hat{x}_2, z \geq 0} \left\{ c(z) + c_1(y - \hat{x}_1 - w_1) \right. \\
+ \left. \bar{L} (\hat{x}_2) + L (\hat{x}_1) \right. \\
+ \left. \alpha \int_0^\infty C_{n-1}(\hat{x}_1 + w_1 - \xi, y - \hat{x}_1 - w_1, \hat{x}_2 + z - \xi) \phi(\xi) d\xi \right\}. 
$$

Similarly for echelon one, we have

$$
C_n (\hat{x}_1, w_m) = \min_{y \geq \hat{x}_1 + w_1} \left\{ c_1(y - \hat{x}_1 - w_1) + L (\hat{x}_1) \right. \\
+ \left. \alpha \int_0^\infty C_{n-1}(\hat{x}_1 + w_1 - \xi, y - \hat{x}_1 - w_1) \phi(\xi) d\xi \right\}. \quad (5.9)
$$

We want to show next that the difference between (5.9) and (5.8) is a function of $\hat{x}_2$ alone. More formally,

**Theorem 1** *(Clark and Scarf [8])* There is a sequence of functions $g_n (\hat{x}_2)$ with $g_1 (\hat{x}_2) = \bar{L} (\hat{x}_2)$, such that

$$
C_n (\hat{x}_1, w_m, \hat{x}_2) = C_n (\hat{x}_1, w_m) + g_n (\hat{x}_2).
$$

We refer the reader to [8] for the proof. It follows that solving for $\bar{x}_1$ from $C_n (\hat{x}_1, w_m)$ in echelon one is independent of the optimization in echelon two. Furthermore, the calculation in echelon 2 is the calculation in echelon 1 plus an expression which depends only on $\hat{x}_2$. Thus the decomposition method is validated.
5.2 A Single-Unit Decomposition Approach

5.2.1 Introduction

Recall the multi-echelon inventory system studied in the previous section. The serial inventory system consists of $J$ stages, indexed by $1, \ldots, J$. Customer demand can only be satisfied by units in stage 1 and any demand that is not immediately satisfied is backlogged. The inventory at stage $i$, $(i = 1, \ldots, J - 1)$ is replenished by placing an order for units stored at stage $i + 1$ while stage $J$ receives replenishments from an outside supplier with unlimited stock, defined as stage $J + 1$. The system is periodically reviewed and thus, a discrete-time model can be employed.

Muharremoglu and Tsitsiklis (2001) introduced a new decomposition approach to solve such multi-echelon inventory systems [18]. The basic idea behind their novel approach is that the multi-echelon inventory system can be solved by decomposing it into a family of subsystems, each being a problem of matching a single-unit to a single-customer in the optimal way. The optimal policy for the subproblem is found to be the well-known echelon base-stock policy.

The motivation for using the new approach is as follows. First, formulating multi-echelon inventory problem as a decomposable system allows us more flexibility: the model is generalized to the case with stochastic leadtime and with an exogenous Markov chain influencing the system. The Clark and Scarf model is a special case of the new method with deterministic leadtimes and a single modulating state (that is, all disturbances from one period to the next is i.i.d.). The Markov modulation allows us to model non-i.i.d. disturbances. For example, in Muharremoglu and Tsitsiklis model, we can have different 'demand states': high demand ($S_1$), medium demand ($S_2$), low demand ($S_3$). At each state, the demand has different distributions or it might have the same distribution but with different parameters.

As a consequence, more realistic situations can be represented by this formulation. The model is also versatile; it can be modified to solve the more conventional problem for-
CHAPTER 5. MULTI-ECHelon INVENTORY SYSTEM

Figure 5.3: The underlying dynamics: there are three possible states, $S_1, S_2, S_3$. There are transition probabilities from one state to another, and each state might give a different demand distribution than others. The Clark and Scarf model is an example of this, where there is only one state $s$.

Simulations (such as the problems discussed in [8], [17]). Secondly, Muharremoglu and Tsitsiklis showed that there existed an efficient algorithm to solve such a mathematical model in polynomial time.

We present this new decomposition approach as outlined by Figure 5.4. First we define a generic stationary discrete time dynamical system, restrict it to a class of decomposable systems and show how this can be used to represent the multi-echelon inventory problem described above. We then characterize the optimal policies for the subsystems and lastly, discuss and implement the algorithm provided in [18].

5.2.2 Problem Formulation

A generic stationary discrete time dynamical system is of the form

$$x_{t+1} = f(x_t, u_t, w_t), \; t = 0, 1, \ldots, T - 1,$$

(5.10)

where $x_t$ is the state of the system at time $t$, $u_t$ is a control to be applied at time $t$, $w_t$ a stochastic disturbance, and $T$ is the time horizon. Assume that $x_t, u_t, w_t$ are elements of given sets $X, U, W$ respectively, and that $f : X \times U \times W \to X$ is a given mapping. We assume that, given $x_t$ and $u_t$, the random variable $w_t$ is conditionally independent from the past and has a given conditional distribution. The properties of
CHAPTER 5. MULTI-ECHELON INVENTORY SYSTEM

Figure 5.4: A summary of the ideas used in a single-unit decomposition approach to multi-echelon inventory models.
the disturbance \( w_t \) will be stated later on.

We define a **policy** \( \pi \) as a sequence of functions, \( \pi = (\mu_0, \mu_1, \ldots, \mu_{T-1}) \), where each \( \mu_t : X \to U \) maps the state \( x \) into a control \( u = \mu_t(x) \). Let \( \Pi \) be the set of all policies.

Given an initial state \( x_0 \) and a policy \( \pi \), the sequence \( x_t \) becomes a Markov chain with a well-defined probability distribution. For any time \( t < T \) and any state \( x \in X \), we define the **cost-to-go** \( J_{t,T}^\pi(x) \) (from time \( t \) until the end of the horizon) by

\[
J_{t,T}^\pi(x) = E\left\{ \sum_{\tau=t}^{T-1} \alpha^{\tau-t} \cdot g(x_\tau, \mu_\tau(x_\tau)) \mid x_t = x \right\},
\]

(5.11)

where \( g : X \times U \in [0, \infty] \) is a given cost-per-stage function and \( \alpha \) is a **discount factor** \((0 < \alpha \leq 1)\). The optimal cost-to-go function is defined by

\[
J_{t,T}^* = \inf_{\pi \in \Pi} J_{t,T}^\pi(x).
\]

A policy \( \pi^* \) is said to be **optimal** if

\[
J_{t,T}^\pi(x) = J_{t,T}^* \quad \forall t, \forall x \in X.
\]

We now restrict ourselves to a class of **decomposable systems**. Intuitively, this is a system consisting of multiple (countably finite) non-interacting subsystems that are driven by a common source of uncertainty, which evolves independently of the subsystems and is modulated by a Markov process \( s_t \).

**Definition 5** A discrete time system of the form (5.10) is said to be decomposable if it admits a representation with the following properties:

**A1.** The state space is a Cartesian product of the form \( X = S \times \hat{X} \times \hat{X} \cdots \), so that any \( x \in X \) can be represented as \( x = (s, x^1, x^2, \ldots) \) with \( s \in S \) and \( x^i \in \hat{X} \), for every \( i \geq 1 \).

**A2.** There is a set \( \hat{U} \) so that the control space \( U \) is the Cartesian product of countably
many copies of $\hat{U}$, that is, any $u \in U$ can be represented as $u = (u^1, u^2, \ldots)$ with $u^i \in \hat{U}$, for all $i \geq 1$.

**A3.** For each $t$, the conditional distribution of $w_t$ given $x_t$ and $u_t$, depends only on $s_t$.

**A4.** The evolution of equation (5.10) for $x_t$ is of the form

$$
\begin{align*}
    s_{t+1} &= f^s(s_t, w_t), \\
    x_{t+1}^i &= \hat{f}(x_t^i, u_t^i, w_t), \forall i \geq 1, \forall t,
\end{align*}
$$

for some functions $f^s : S \times W \to S$ and $\hat{f} : \hat{X} \times \hat{U} \times W \to \hat{X}$.

**A5.** The cost function $g$ is additive, of the form

$$
g(x_t, u_t) = \sum_{i=1}^{\infty} \hat{g}(x_t^i, u_t^i), \quad (5.12)
$$

for some function $\hat{g} : \hat{X} \times \hat{U} \to [0, \infty)$.

**A6.** The sets $\hat{X}$ and $W$ are countable. The sets $S$ and $\hat{U}$ are finite.

In a decomposable system, the control variable is a vector. Any policy $\pi$ can be represented in terms of component mappings $\mu^i_t : X \to \hat{U}$, so that

$$
u_t^i = \mu^i_t(x_t), \forall i, t.
$$

**Definition 6** A policy $\pi$ for a decomposable system is said to be **decoupled** if it can be represented in terms of mappings $\hat{\mu}_t : S \times \hat{X} \to \hat{U}$, so that

$$
u_t^i = \hat{\mu}_t(s_t, x_t^i), \forall i, t.
$$

Intuitively, the above definition requires the control $u_t^i$ that affects the $i$th subsystem to be made locally, without considering the state of the other subsystem.

For a decomposable system, the various state components $x_1^t, x_2^t, \ldots$ evolve independently of each other, the only coupling arising through the exogenous processes $s_t$ and $w_t$ and
the costs are additive. It should be clear that each subsystem can be controlled separately (using a decoupled policy) without any loss of optimality. Furthermore, all subsystems are identical and so the same mappings $\hat{\mu}_t$ can be used in each subsystem.

Let $J^*_{t,T}(s,x^i)$ be the optimal cost-to-go function for a subsystem that starts at time $t$ from state $(s,x^i)$ and evolves until the end of the horizon $T$. Note that this function is the same for all $i$, because we have assumed the subsystems to have identical dynamics and cost functions. Also, since the control set $\hat{U}$ is finite, an optimal policy is guaranteed to exist.

**Lemma 2** Consider a decomposable system.

1. For any $s = (s,x^1,x^2,\ldots) \in X$ and any $t \leq T$, we have

\[
J^*_{t,T}(x) = \sum_{i=1}^{\infty} j^*_{t,T}(s,x^i).
\]  

2. There exists a decoupled policy $\pi^*$ which is optimal, that is,

\[
J^*_{t,T}(x) = J^*_{t,T}(x), \forall t, \forall x \in X.
\]

3. For any $s,x^i$ and any remaining time $k$, let $\hat{U}_k^*(s,x^i) \subset \hat{U}$, be the set of all decisions that are optimal for a subproblem, if the state of the subproblem at time $T-t$ is $(s,x^i)$. A policy $\pi = \{\mu^i_t\}$ is optimal if and only if for every $i,t$, and any $x = (s,x^1,x^2,\ldots) \in X$ for which $J^*_{t,T}(x) < \infty$, we have

\[
\mu^i_t(x) \in \hat{U}_T^*(s,x^i).
\]

Now that the desirable mathematical formulation is described, we need to define the choices for the state, control, and disturbance variables for the inventory control system concerned.

The state of a system is usually chosen as the amount of stock on hand, or in transit
In this model however, the state of the system consists of countably infinite number of vectors, one for each unit-customer pair, and a variable $s_t \in S$ for the state of the modulating Markov chain,

$$x_t = \{s_t, (z^1_t, y^1_t), (z^2_t, y^2_t), \ldots \}. \quad (5.14)$$

For each unit-customer pair, $j$, $(j \in \mathbb{Z}^+)$, we have a vector $(z^j_t, y^j_t)$ with $z^j_t \in \mathbb{Z} = 0, 1, \ldots, J + 1$ and $y^j_t \in \mathbb{N}_0$, where $z^j_t$ is the location of unit $j$ at time $t$, and $y^j_t$ is the position of customer $j$ at time $t$. This results in a decomposable problem, with each unit-customer pair viewed as a separate subproblem.

The control vector is an infinite binary sequence $u_t = (u^1_t, u^2_t, \ldots)$ where the $j$th component $u^j_t$ corresponds to a release or hold decision for the $j$th unit.

$$u_t = \begin{cases} 
1 & \text{, if the unit is released from its current location} \\
0 & \text{, otherwise.} 
\end{cases} \quad (5.15)$$

The stochastic disturbance term, $w_t$ consists of the demand $d_t$, the random variables that model the uncertainty in the leadtimes, and whatever additional exogenous randomness that is needed to drive the Markov chain $s_t$.

The (nonnegative integer) demand $d_t$ during period $t$ is assumed to be Markov modulated. In particular, the probability distribution of $d_t$ depends on the state $s_t$ of the exogenous Markov chain, and conditioned on that state, is dependent of the history of the process until now. We also assume that $E[d_t|s_t = s] < \infty$, for every $s \in S$.

The leadtime between adjacent stages are bounded by some integer $l_t$. We assume that orders cannot overtake each other: an order cannot arrive at its destination before an earlier order does.

We now look at the dynamic of the subsystem, assessing the relationship between the state, control variable and stochastic disturbance.
The (Conceptual) Location of a Unit. Each of the actual stages in the system will constitute a location. Next, we insert $l_i - 1$ artificial locations between the locations corresponding to stages $i$ and $i + 1$, for $i = 1, \ldots, J$, in order to model the units in transit between these two stages. By doing this, the leadtime between two consecutive stages is one unit time.

**Definition 7** Let $A$ be the set of locations corresponding to the actual stages of the system, including the outside supplier. Also,

$$v(z) = \max_{i \in A \text{ and } i \leq z} i, \forall z > 0,$$

that is, starting from location $z$ and going in the direction of decreasing $z$, $v(z)$ is the first location corresponding to an actual stage.

Finally, a location 0 is created to specify the location of all the units that have been given to the customers.

The conceptual location of a unit is dependent on the variability of the leadtime and the decision to hold or release the unit. This is illustrated by Figures 5.5 and 5.6. Let $U(1, l_i)$ denote the uniform distribution on the integers $\{1, 2, \ldots, l_i\}$. We assume that there is a random variable $Y_i \sim U(1, l_i)$. Consider a 10-installation system, with location 11 referring to the outside supplier. The set of actual stages $A$ is $\{2, 5, 8, 10, 11\}$.

Note that our decision $u_i^t$ only affects the movement of the unit in the actual stages other than 0. If a unit is an artificial location, it is only affected by the random variable $Y$.

**Position of Customer.** The position of a customer $i$ at time $t$ is denoted by the variable $z_t \in \{0, 1, \ldots\}$ and defined as follows.

- $z = 0$ If a customer who arrives at time $t$ and has his or her demand fulfilled, we define the position to be 0.
- $z = 1$ If a customer has arrived and has his or her demand backlogged, we define the position to be 1.
Figure 5.5: At the actual stages, we make decisions to hold the unit in the current stage. When a unit is already on the move (that is in one of the 'fictitious stages'), it follows the random variable $Y$. For a unit initially in stage 7 for example, it will jump to stage 6 with probability 0.5 and to stage 5 with the same probability. From stage 6 it will go to stage 5. These transition probabilities are equivalent to the leadtime random variable having the required uniform distribution.

Figure 5.6: At the actual stages, we make decisions to release the unit to the next downstream stage. Once released, the random variable $Y$ determines which downstream stage it will go to. For a unit released from stage 8 for example, it will go to stage 7, 6, or 5 with uniform probability. For a unit that is initially at a fictitious stage (that is, it has been on the move), such as at stage 7, it will continue to go downstream to stage 6 or 5 with uniform probability as well.

**Note that there might be more than one customers with same position 0 and 1**

$z \geq 1$ A customer who has not arrived will have position $z \neq 1$. If customers $1, 2, \ldots, n$ are waiting for their orders and are thus in position 1, the customer $i > n$ will have location number $i - n + 1$. For example, if customers number 1 and 2 are in position 1, then customer 3 will have location number $(3 - 2 + 1) = 2$, customer 4 in position 3 and so on.
Now we specify the dynamic of the customer position. For every \( t \),

\[
y_{t+1} = \begin{cases} 
y_t - d_t, & \text{if } y_t - d_t \geq 1 \text{ and } y_t > 1, \\
1, & \text{if } y_t - d_t < 1 \text{ and } y_t > 1, \\
1 - u_t 1_{z_t=1}, & \text{if } y_t = 1 \\
0, & \text{if } y_t = 0. 
\end{cases}
\]  
(5.16)

Note that the dynamic of the customer position is determined by the random variable \( d_t \). In addition, for location 1 it is also affected by the availability of the product in unit location 1.

To complete the optimization model, we define the following costs.

**a** For each stage \( i \), there is an inventory holding cost rate \( h_i \) that gets charged at each time period to each unit at that stage. We assume that the holding rate \( h_{J+1} \) at the external supplier is zero and the holding rate charged for a unit in transit corresponds to the holding rate in the destination echelon.

**b** For each stage \( i \), there is a cost \( c_i \) for initiating the shipment of a unit from stage \( i + 1 \) to stage \( i \).

**c** There is a backorder cost rate \( b \) which is charged at each time step for each unit of backlogged demand.

The one period costs are stationary and thus defined by

\[
\hat{g}(s_t, z_t, y_t, u_t) = \hat{h}_{zt} - \hat{h}_{zt} \cdot 1_{(z_t=1)} \cdot 1_{(y_t=1)} \cdot \\
+ b \cdot 1_{(y_t=1)} \cdot (1 - 1_{(z_t=1)} \cdot u_t) \\
+ \hat{c}_{zt} \cdot u_t. 
\]  
(5.17)

Note that the backlog and order costs are affected by our decision \( u_t \), whereas the holding cost is only affected by the unit and position locations.

The total cost function \( V_t(s, z, y, u) \) with \( t \) period(s) left until the end of the horizon is
thus given by:

$$V_t(s, z, y, u) = \hat{g}(s, z, y, u) + E\left[\hat{J}_{t-1}^{*}\left(\hat{f}(s, z, y, u, w)\right)\right]. \tag{5.18}$$

The first term on the right is the one period cost, and the second term is the expectation of the optimal cost for \(t-1\) period(s) left until the end of the horizon. The latter calculates recursively the cost in the future periods if we make a certain decision \(u_t\) under the stated conditions.

### 5.2.3 Policy Classification

**Definition 8** A state \(x_t = \{s_t, (z_t^1, y_t^1), (z_t^1, y_t^2), \ldots\}\) is called monotonic if and if the unit locations are monotonic functions of the unit labels, that is,

$$i < j \Rightarrow z_t^i \leq z_t^j.$$ 

**Monotonic policies** Given a policy \(\pi\), let \(\mu_t = (\mu_t^1, \mu_t^2, \ldots)\) be the part of the policy that applies to \(t\). The policy \(\pi\) is monotonic if it guarantees that a monotonic state \(x_t\) always results in a next state \(x_{t+1}\) that is monotonic. That is, for every \(t\), if \(x_t\) is monotonic, then \(x_{t+1} = f(x_t, \mu_t(x_t), w_t)\) is monotonic for every possible realization of \(w_t\). Restricting ourselves to this this of policy ensures that the orders will not overtake each other.

**Committed policies** These are policies such that if \(z_t^j = 1\) and \(y_t^j > 1\), then \(\mu_t^j(x_t) = 0\). In words, if unit \(j\) is at the last stage, it can only be released to satisfy the corresponding customer \(j\), and this can only happen if the customer has already arrived and is backlogged.

**Decoupled policies** Its definition is as mentioned above. Intuitively, this means that the decision to release or hold a unit is a function of the state of the modulating Markov chain, the location of the unit, and the position of the corresponding customer. Also, the function is the same for every unit.
State-dependent echelon base stock policies Its definition is as defined in previous chapters. In this model, the base-stock levels are dependent on the state of the Markov chain $s_t$.

Tsitsiklis and Muharremoglu [18] showed that we can restrict ourselves to the above policies without compromising optimality. That is, the intersection of committed, decoupled and monotonic policies are the state-dependent echelon base-stock policies. Moreover, such policies are optimal for the multi-echelon inventory system. We summarize the results that are needed here; as for the proofs, we refer the reader to Section 3.2 and 3.3 in [18]. The results are proven in both finite and infinite horizon analysis.

**Proposition 1 (Muharremoglu and Tsitksiklis Proposition 3.1)** A monotonic, committed, and decoupled policy is a state dependent echelon base-stock policy.

**Proposition 2 (Muharremoglu and Tsitksiklis Proposition 4.4)** The set of monotonic, committed, and decoupled policies is optimal.

**Theorem 2 (Muharremoglu and Tsitksiklis Theorem 4.1)** The set of state-dependent echelon base stock policies is optimal.

### 5.2.4 Algorithmic Issues

Having seen the decomposition method, we can restrict ourselves to solving a single-unit single-customer subproblems in order to solve the multi-echelon system. The resulting base-stock levels are readily obtained from the solutions of the single-unit single-customer subproblems. In this thesis we present Muharremoglu and Tsitsiklis algorithm for the finite horizon case. The infinite horizon algorithm is provided in Section 6.3 in [18].

Given the following:

- one period cost function (5.16),
Initialization of the Finite Horizon Algorithm (FHA):

\begin{align*}
\hat{J}_0^*(s,z,y) &= 0; \quad \forall (s,z,y) \\
\hat{J}_t^*(s,z,0) &= 0; \quad \forall (s,z,t) \\
\hat{J}_t^*(s,0,y) &= 0; \quad \forall (s,y,t) \\
y_t^*(s,z) &= -\infty; \quad \forall (s,z \in A,t) \\
K_t(s,z) &= 0; \quad \forall (s,z \in A,t) \\
y &= 0
\end{align*}

Recursion of the Finite Horizon Algorithm (FHA):

while \((K_t(s,z) = 0\) for some \((s,z \in A,t)\)) do

\begin{align*}
y &= y + 1 \\
&\text{for } (t = 1, \ldots, T) \\
&\quad \text{for } (z = 1, \ldots, N + 1) \\
&\quad \quad \text{for } (s = 1, \ldots, |S|) \\
&\quad \quad \quad \text{if } (z \in A \text{ and } K_t(s,z) = 0) \\
&\quad \quad \quad \quad \hat{J}_t^*(s,z,y) = \min_{u \in \{0,1\}} V_t(s,z,y,u) \\
&\quad \quad \quad \quad \hat{U}_{T-t}^*(s,z,y) = \left\{ u | u = \arg\min_{u \in \{0,1\}} V_t(s,z,y,0) \right\} \\
&\quad \quad \quad \quad \text{if } \hat{U}_{T-t}^*(s,z,y) = \{1\} \\
&\quad \quad \quad \quad \quad y_t^*(s,z) = y \\
&\quad \quad \quad \quad \text{if } \hat{U}_{T-t}^*(s,z,y) = \{0\} \\
&\quad \quad \quad \quad \quad K_t(s,z) = 1 \\
&\quad \quad \quad \text{else} \\
&\quad \quad \quad \quad \hat{J}_t^*(s,z,y) = V_t(s,z,y,0) \\
&\quad \quad \quad \quad \text{next } s \\
&\quad \quad \quad \text{next } z \\
&\quad \quad \text{next } t \\
&\quad \text{end while}
\end{align*}

Figure 5.7: The Finite Horizon Algorithm
CHAPTER 5. MULTI-ECHelon INVENTORY SYSTEM

• system dynamics mapping \( f(s, z, y, u, w) \) for all \((s, z, w, y, u, w)\),

• conditional probability distribution of \( w_t \),

• List of locations corresponding to actual stages, \( A \),

the algorithm goes progressively increasing value of \( y \), to find the optimal value of (5.18) via the decision variable \( u \) (Figure 5.7).

For each unit location \( z \), time \( t \) and state \( s \), we want to calculate the lowest customer position \( y^* \) such that it is optimal to release the unit if and if the customer location is below the threshold \( y^* \). To do this, we consider the functional equation of the cost (5.18), and choose the optimal decision \( u^* \). In addition, to keep track of the progress, we define the variables \( K_t(s, z) \) such that

\[
K_t(s, z) = \begin{cases} 
1 & \text{if the threshold level has been determined} \\
0 & \text{otherwise.} 
\end{cases} \tag{5.19}
\]

The variables \( K_t(s, z) \) are initialized to zero. Note from Figure 5.7 that the optimal threshold levels are calculated only for the actual stages \( z \in A \).

The algorithm is proven to run in polynomial time [18].

**Proposition 3 (Muharremoglu and Tsitksiklis Proposition 6.2)** The complexity of the Finite Horizon Algorithm is \( O(N \cdot Y_{\text{max}} \cdot \min Y_{\text{max}}, D \cdot |S|^2 \cdot T) \).

**Numerical Example.** Consider the problem with the following parameters:

• Number of locations \( N = 12 \),

• Set of actual stages including stage 0, \( A = [0, 1, 5, 8, 10, 11] \),

• Time limit of the finite horizon \( T = 20 \),

• The demand \( P_n(\lambda = 6) \),
• The holding cost is $h = 3$ in all actual stages but the outside supplier,

• The ordering cost $c = 4$ in all actual stages but the outside supplier,

• The backordering cost $b = 5$ in all actual stages but the outside supplier.

Note that for stage 0, the $y_t$ threshold values are all zero. Stage 0 refers the location of the unit that has been given to the customer. At the highest installation stage 11, we have the threshold values $y_t = 0$, $\forall t = 1, 2, 3, 4, 5, 6$ remaining until the end of the horizon. This means that nearing the end of the horizon, installation 11 must have enough stock to send downstream.

**Recovery of Base-Stock Policies.** As mentioned, the output of the algorithm is the largest value of $y_t$ calculated, which gives the customer location threshold for a particular unit location $z$ at time $t$ in state $s$.

The base-stock level of a location $v(z - 1)$, which is the location that corresponds to the next actual stage after $z$, can be determined given the threshold $y_{t}^*(s, z)$. Let $A' = A\{1\}$.

**Proposition 4 (Muharremoglu and Tsitksiklis Proposition 6.1)** Let $y_{t}^*(s, z)$ be determined through the Finite Horizon Algorithm. A state dependent echelon base-stock
policy with base-stock levels \( S_v(z^1) - 1 \) for all \((s, z \in A', t)\) is optimal for the overall system.

The threshold value for \( y_t \) at installation 1 is not determined from the algorithm. However, since the base-stock value is determined from the threshold of the higher real installation, we can recover the base-stock value for installation 1 from \( y_t \) in installation 2.

### 5.3 Comparison and Discussion

To check the validity of the Muharremoglu and Tsitsiklis results, we run and compare the results with the base-stock policies recovered from the Clark and Scarf method. Figure 5.9 gives the results from the two algorithms. For simplification, we have taken the

<table>
<thead>
<tr>
<th>Time lag =2, time horizon=5, Lambda =2</th>
<th>Clark &amp; Scarf Base-stock/duration</th>
<th>Muharremoglu &amp; Tsitsiklis Base-stock/duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>0.67 sec</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.22 sec</td>
</tr>
<tr>
<td>Time lag =2, time horizon=6, lambda = 2</td>
<td>9</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>5.55 sec</td>
<td>0.22 sec</td>
</tr>
<tr>
<td>Time lag =3, time horizon=5, Lambda =3</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>0.27 sec</td>
<td>0.22 sec</td>
</tr>
<tr>
<td>Time lag =3, time horizon=6, Lambda =3</td>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>2.37 sec</td>
<td>0.22 sec</td>
</tr>
<tr>
<td>Time lag =3, time horizon=4, Lambda =3</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>0.34 sec</td>
<td>0.22 sec</td>
</tr>
<tr>
<td>Time lag =3, time horizon=5, Lambda =5</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>0.61 sec</td>
<td>0.33 sec</td>
</tr>
<tr>
<td>Time lag =3, time horizon=5, Lambda =7</td>
<td>28</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>1.22 sec</td>
<td>0.42 sec</td>
</tr>
</tbody>
</table>

Figure 5.9: The \( y_t \) threshold values for five real stages, excluding the outside supplier.

base-stock values of one installation, at the start of the system, that is time \( N \) periods remaining. The recovery of the base-stock policies from the Clark and Scarf algorithm is as proposed in Section 4.2.2 whereas the recovery for the Muharremoglu and Tsitsiklis
is done using Proposition 4.

We have failed to test the Clark and Scarf algorithm for larger parameters. In our formulation, the Clark and Scarf algorithm runs in exponential time, whereas the Muharremoglu and Tsitsiklis algorithm runs in polynomial time. These results cannot be compared, as we have not formulated the most efficient algorithm for the former.

From a user’s point of view however, Muharremoglu and Tsitsiklis algorithm is easier to implement and can be used to solve more general problems. Moreover, in the implementation of the algorithm, we do not have to truncate the distribution- when the support of the demand distribution is infinite (we have used Poisson distribution in these examples).

The Clark and Scarf algorithm, on the other hand, is problem specific and needs truncation in modelling the demand distribution. Although the optimal policies for the serial inventory systems have been characterized, the actual computation needed to recover the solutions is not straight forward. Studies of serial inventory systems after Clark and Scarf’s breakthrough paper still resorted to heuristic methods to simplify the amount of computations needed (see [25] for example); a fact that was noted by Scarf [23] as well.
Chapter 6

Dynamic Leadtime Management in Supply Chains

In *Dynamic Leadtime Management in Supply Chains* [19], Muharremoglu and Tsitkilis extended the idea used in [18]. Whereas the earlier paper has mainly dealt with the algorithmic aspects of decomposing a multi-echelon inventory system into single unit-single customer pairs, [19] focuses on the characterizations of a supply chain model. We describe briefly its findings here.

The subject of the paper is a new model of supply chain where the leadtimes are dependent on decisions made dynamically by choosing alternative shipping methods. The leadtimes are normally an external (deterministic or stochastic) parameter which we cannot control. In the new mathematical model, however, the leadtimes become the control variable.

The findings of the paper can be summarized as follow: assuming that the cost structure is supermodular (to be defined below) and the leadtime between adjacent stages is one, the problem can be decomposed into single unit-single customer subproblems which have *extended echelon base-stock policies* as the optimal policies. Many of the concepts and proofs of [18] can be applied to this new model. Thus the basic concepts of decomposition,
decoupling, and monotonicity are all relevant.

Definition 9 Extended echelon base-stock policies are threshold policies which specify a threshold for every pair of origin-destination stages (that is, for every origin stage \( z \) and destination stage \( w \) with \( z > w \)).

Consider the inventory locations \( z = 0 \ldots 8 \), with location 8 as the external supplier and location 0 allocated to items already given to the customer. The latter can thus be ignored. The leadtime from stage \( z+1 \) to \( z \) is one unit time. For each pair \( \{(z, w) : z > w, z, w \in \{1 \ldots 8\}\} \), there is a shipment or ordering cost from \( z \) to \( w \), and the paper shows that optimal policies can be characterized by base-stock values for all such pairs \( (z, w) \). Hence there are \((M + 1).M/2\) such pairs, with \( M = 7 \) the number of locations, not including the supplier.

The concept is a generalization of the well known echelon base stock policies. The term extended refers to the fact that thresholds are defined for pairs of echelons, rather than a single echelon. We will now explain the two new fundamental assumptions used in order for the decomposition to work: supermodular cost structure and 1 unit leadtime between adjacent stages.

The supermodularity condition on the costs of shipments \( c(z, w) \) states that if \( x > v > w > r \) are all locations, then

\[
c(x, r) + c(v, w) \geq c(x, w) + c(v, r).
\]

The condition is called additive if equality holds. The additive case is discussed in [?]. Intuitively, the condition says that order overtaking by jumping more locations is at least as costly as not overtaking. Order overtaking itself is not ruled out in the model. In fact it is possible for shipments to jump several locations and overtake other shipments. To find the minimum, however, we can restrict ourselves to cases without overtaking. This is illustrated in Figure 6.1.

An example of supermodularity is when the shipment cost is a convex function \( G(k) \) of the
number of locations skipped $k$. Consider the 8-installation supply chain described before. Figure 6.1.a illustrates two alternative shipments, one of them with order overtaking.

It is clear from the Figure 6.1.b., that

$$c(8, 1) + c(5, 4) = G(6) + G(0)$$
$$< c(5, 1) + c(8, 4) = G(3) + G(3).$$

Note that when the function $G(k)$ is linear, the equality holds (additive). On the other hand, when it is concave, we have a violation of supermodularity. We need to keep this in mind when considering whether supermodularity holds in a particular situation.

An example of such a situation is when we drop the assumption that the leadtime is one. Following the method in [18], the problem with leadtimes strictly greater than one can be modelled as problems with leadtimes equal to one, by introducing artificial locations. However, this creates a problem if the stock jumps to an artificial location. Supermodularity may not be valid here. The exact conditions under which supermodularity will hold (for cases with leadtimes greater than one) were not discussed and still need to be formulated.

Assuming supermodularity and that the leadtime is one, the problem can be decomposed into independent subproblems and so we can find the extended echelon base-stock policies for any two stages. The proof is parallel to [18]: the optimal solution needs to be found only among monotonic and committed policies. Monotonic is a precise definition of not overtaking, and committed refers to pairing of items and customers (in the abstract sense, including customers who have not yet arrived).

Although there is no explicit algorithm in the paper, it outlines the general method of decomposition into independent subproblems to be solved using dynamic programming. The basic structure of the programs for finite and infinite horizons discussed in [18] remains valid here. The only difference to be accounted for is the use of extended echelon policies rather than the normal echelon policies.
The model studied is very interesting from the standpoint of practical applications. Many supply chain problems fall under this category, not just the case of alternative shipments discussed above. Other seemingly different problems can sometimes be interpreted as such. For example, the paper mentions the case of dual suppliers.

In conclusion, we see that the work in [19] parallels that in [18] quite closely, although there are some important differences (see the table below). The assumption of supermodularity is essential here. It is a sufficient condition for the decomposition results to hold, but is it also a necessary condition? What about the cases where supermodularity fails?

A summary of the differences between the two papers: (6.0)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial inventory system</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Stochastic leadtime</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Markov modulated</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Leadtime</td>
<td>n</td>
<td>1</td>
</tr>
<tr>
<td>Base stock</td>
<td>echelon</td>
<td>extended echelon</td>
</tr>
<tr>
<td>Shipment cost</td>
<td>additive</td>
<td>supermodular</td>
</tr>
<tr>
<td>Algorithm</td>
<td>explicit</td>
<td>needs elaboration</td>
</tr>
</tbody>
</table>

The cases of stochastic leadtime, Markov modulation, and leadtimes greater than one are discussed as possible extensions in the paper. These extensions are possible, but exact conditions are to be formulated to ensure validity of the solution, particularly to ensure supermodularity.
Figure 6.1: (a) Consider two items, one being at stage 8 and the other at stage 5. The diagram shows all possible routes for shipment from these stages. (b) The cost function $G(k)$ is convex.

Choice of Shipments:
- 8 -> 1 and 5 -> 4 (overtaking)
- 8 -> 4 and 5 -> 1
Chapter 7

Conclusion

Our main intention in this thesis was to study a new decomposition approach to multi-echelon systems— as suggested by Muharremoglu and Tsitsiklis [18]— and to compare the results with some benchmark. The choice of Clark and Scarf method as the benchmark was quite natural, since their paper [8] was considered a milestone in the multi-echelon inventory literature; it was selected as one of most influential papers by several publications and societies [16]. The focus on the two methods was not intended to suggest that many other research were considered less significant. Some of these were mentioned in the literature review section.

The result of the Muharremoglu and Tsitsiklis algorithm was a set of threshold values: the furthest location a customer should be if an item was to be released from its current position. From a threshold value, the basestock level at a stage next in the downstream could be recovered. In this thesis we developed a Matlab implementation of the algorithm which allowed a numerical visualization and a confirmation of the relationship between the threshold and base-stock values, by drawing a comparison to the base-stock levels recovered from the Clark and Scarf algorithm. This Matlab program still needed to be generalized to include the case of infinite horizons, and to allow Markov chain modulation.

Dynamic Programming is a natural choice for solving inventory problems. However
dynamic programming is not a unique algorithm, there are top-down and bottom-up versions, and various variations of these exist. In this thesis, we used one formulation of dynamic programming for the Clark and Scarf method. In future work, the different variations could be studied and compared. It would also be beneficial to compare the Clark and Scarf against Muharremoglu and Tsitsiklis' algorithms for the same problem domain, with large sample data, to see the performance and scalability of both. The results should be a confirmation of the theoretical complexity results.

Another interesting problem is to formulate the precise conditions for Muharremoglu and Tsitsiklis' method of decomposition to work. Likewise, from the supply chain problems encountered in day-to-day industrial practice, we would like to know which specific type of problems would be amenable to such a decomposition.
Appendix A

Numerical Example for Section 4.2.1

PARAMETERS

<table>
<thead>
<tr>
<th>Costs</th>
<th>Purchasing</th>
<th>K</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>c</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Holding</td>
<td>h</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Penalty</td>
<td>p</td>
<td>3</td>
</tr>
<tr>
<td>Discount</td>
<td></td>
<td>α</td>
<td>0.9</td>
</tr>
<tr>
<td>Demand</td>
<td></td>
<td>ξ</td>
<td>(0, 1, 2, 3)</td>
</tr>
</tbody>
</table>

We have \( P(\xi = i) = 0.25, \ \forall i = 0, 1, 2, 3. \)

- Step 1.0: \( n = 1, C_0(x_0) = 0. \)

- Step 1.1: Minimize \( G_1(y) \) to get the optimal \( y_1 \) (Table A.1).

- Step 1.2: For the optimal value of \( y_1^* \), we have \( C_1(x) \) in Table A.2. The domain of \( x_1 \) is chosen such that if we have the lowest stock \( y_1 = -5 \) and the highest demand \( \xi_1 = 3 \), we have the value for \( x_1 = -8 \). Similarly, if we have the highest stock \( y_1 = 5 \) and the lowest demand \( \xi_n = 0 \) we have the upper bound on \( x_n = 5 \). Hence
we consider $x \in [-8, 5]$.

• Step 1.3: Set $n = 2$.

• Step 2.1: Minimize $G_2(y)$ to get the optimal $y_2$ (Table A.1). Note that the calculation of the expectation of $C_1(y - \xi)$ is as follows. For the value of $y = 5$, we consider having each value of demand $\in [0, 1, 2, 3]$. Since the probability is uniform, we just take the average. And so

$$E[C_1(y - \xi)] = 0.25 \left[ C_1(5) + C_1(4) + C_1(3) + C_1(2) + C_1(1) \right]$$

The procedure is repeated for all values of $y$. Hence the result in Table A.1.

• We recover the minimizing $y_2^* = 2$ and so we stop the algorithm.
**Figure A.1:** The calculation of (4.9) by considering \( y \in [-5, 5] \)

<table>
<thead>
<tr>
<th>( n )</th>
<th>( y )</th>
<th>( h \cdot y )</th>
<th>( p \cdot \text{integral} )</th>
<th>( L(y) )</th>
<th>( \text{Exp}(C(n)) )</th>
<th>( c \cdot y )</th>
<th>( G(n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-5</td>
<td>0</td>
<td>19.5</td>
<td>19.5</td>
<td>0</td>
<td>-10</td>
<td>9.5</td>
</tr>
<tr>
<td></td>
<td>-4</td>
<td>0</td>
<td>16.5</td>
<td>16.5</td>
<td>0</td>
<td>-8</td>
<td>8.5</td>
</tr>
<tr>
<td></td>
<td>-3</td>
<td>0</td>
<td>13.5</td>
<td>13.5</td>
<td>0</td>
<td>-6</td>
<td>7.5</td>
</tr>
<tr>
<td></td>
<td>-2</td>
<td>0</td>
<td>10.5</td>
<td>10.5</td>
<td>0</td>
<td>-4</td>
<td>6.5</td>
</tr>
<tr>
<td></td>
<td>-1</td>
<td>0</td>
<td>7.5</td>
<td>7.5</td>
<td>0</td>
<td>-2</td>
<td>5.5</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>4.5</td>
<td>4.5</td>
<td>0</td>
<td>0</td>
<td>4.5 min</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1.5</td>
<td>2.5</td>
<td>0</td>
<td>2</td>
<td>4.5</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>10</td>
<td>15</td>
</tr>
</tbody>
</table>

| 2 | -5 | 0 | 19.5 | 19.5 | 17.5 | -10 | 25.25 |
| | -4 | 0 | 16.5 | 16.5 | 15.5 | -8 | 22.45 |
| | -3 | 0 | 13.5 | 13.5 | 13.5 | -6 | 19.65 |
| | -2 | 0 | 10.5 | 10.5 | 11.5 | -4 | 16.85 |
| | -1 | 0 | 7.5 | 7.5 | 9.5 | -2 | 14.05 |
| | 0 | 0 | 4.5 | 4.5 | 7.5 | 0 | 11.25 |
| | 1 | 1 | 1.5 | 2.5 | 5.5 | 2 | 9.45 |
| | 2 | 2 | 0 | 2 | 3.5 | 4 | 9.15 min |
| | 3 | 3 | 0 | 3 | 2.5 | 6 | 11.25 |
| | 4 | 4 | 0 | 4 | 0.5 | 8 | 12.45 |
| | 5 | 5 | 0 | 5 | -2.5 | 10 | 12.75 |
Figure A.2: The shaded row gives the numerical values for the function $C_1(x)$, $x \in [-8, 5]$. 
Appendix B

Glossary for Section 5.2

List of variables used.

<table>
<thead>
<tr>
<th>category</th>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>$t$</td>
<td>time $0....T$</td>
</tr>
<tr>
<td></td>
<td>$T$</td>
<td>maximum $t$ or the time horizon</td>
</tr>
<tr>
<td>State</td>
<td>$x$</td>
<td>system state</td>
</tr>
<tr>
<td></td>
<td>$s$</td>
<td>state of the modulating Markov chain</td>
</tr>
<tr>
<td></td>
<td>$f$</td>
<td>State transition $nextstate = f(x, s, u)$</td>
</tr>
<tr>
<td></td>
<td>$z$</td>
<td>item location, $z \in {0..N}$, including actual installations and artificial locations</td>
</tr>
<tr>
<td></td>
<td>$A$</td>
<td>set of actual installations</td>
</tr>
<tr>
<td></td>
<td>$l$</td>
<td>$l_i$ = upperbound on lead time from installation $i + 1$ to $i$</td>
</tr>
<tr>
<td></td>
<td>$v$</td>
<td>function of $z$ giving next actual location downstream</td>
</tr>
<tr>
<td></td>
<td>$y$</td>
<td>customer location</td>
</tr>
</tbody>
</table>
### Appendix B. Glossary for Section 5.2

<table>
<thead>
<tr>
<th>Demand</th>
<th>d</th>
<th>$d(t)$ is demand time $t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decision</td>
<td>u</td>
<td>decision variable $\epsilon U$</td>
</tr>
<tr>
<td></td>
<td>U</td>
<td>control space</td>
</tr>
<tr>
<td></td>
<td>$\mu$</td>
<td>mapping that maps $X$ to $U$</td>
</tr>
<tr>
<td></td>
<td>$\pi$</td>
<td>policy is a set $\pi = {\mu_0, \mu_1, \ldots, \mu_{T-1}}$</td>
</tr>
<tr>
<td>Cost</td>
<td>$h$</td>
<td>holding cost</td>
</tr>
<tr>
<td></td>
<td>$c$</td>
<td>shipment or order cost</td>
</tr>
<tr>
<td></td>
<td>$b$</td>
<td>backorder cost</td>
</tr>
<tr>
<td></td>
<td>$g$</td>
<td>period cost $g(s_t, z_t, y_t, u_t)$</td>
</tr>
<tr>
<td></td>
<td>$\alpha$</td>
<td>discount factor</td>
</tr>
<tr>
<td></td>
<td>$J$</td>
<td>cost to go, define $J_{t,T}^\pi(x) = \text{expected discounted cost from } t &lt; T \text{ to the end of horizon}$</td>
</tr>
<tr>
<td></td>
<td>$J^*$</td>
<td>optimal cost to go, minimized over the set of policies</td>
</tr>
<tr>
<td></td>
<td>$V$</td>
<td>$V_t(s, z, y, u)$ is current period cost + optimal cost to go</td>
</tr>
<tr>
<td>Threshold</td>
<td>$y^*$</td>
<td>threshold values $y^*_t(s, z)$</td>
</tr>
<tr>
<td>Base Stocks</td>
<td>S</td>
<td>$S^*_t(s) = \text{base stocks derived from thresholds}$</td>
</tr>
<tr>
<td>Others</td>
<td>K</td>
<td>$K_t(s, z)$ used in the algorithm to keep track whether certain elements have been computed</td>
</tr>
</tbody>
</table>
Appendix C

Matlab Code

We attach the Matlab codes in the following order.

- The Clark and Scarf algorithm
  - CSA.m
  - F.m
- The Muharremoglu and Tsitsiklis algorithm
  - Fha.m
  - Actual.m
  - G.m
  - V.m

% Clark & Scarf Algorithm (CSA)
% Matlab version by Jennifer Kusuma, Univ of Melbourne
% date: Sept 2005
%============================================================================
% VERSION 1.0
%========================================================================

% Clark & Scarf defined the recurrence relations
% the exact form of the algorithm as used here is not from their paper

%========================================================================
% clear commands and variables
%========================================================================
clc;
clear;
disp('CSA demo, Please wait...')
disp('stochastic version ')

%========================================================================
% size and other parameters
%========================================================================
% all indices start from 1
% left to right is from 1 to N, the reverse of order in spreadsheet
N = 5; % time horizon
x = zeros(1,N); x(N) = 0;
global alpha;
alpha = 2;
% x is stock on hand, the start only X(N) is given,
% all others will be computed
% demand model
global Lambda; % declared global so that it can be used in f.m
Lambda = 3;
global demand;
demand = zeros(1,10);
% demand(v) is the poisson distribution of v-1 arrivals in a period, given
% Lambda
% we assume that Lambda is small and number of arrivals > maxarr is negligible
% since vectors and arrays in Matlab start from index 1,
% we interpret v= 1 as 0 arrivals, v = 10 as 9 arrivals, etc
global maxarr;
maxarr = 5; % larger maxarr will run longer!
% however chopping the poisson distribution at maxarr will not make it a
% true probability distribution (sum=1), hence we need to normalize the
% distribution using totdemand
% totdemand = 0;
% for v=1:1:maxarr
%   demand(v) = exp(-Lambda) * (Lambda.^(v-1))/gamma(v);
%   % formula for poisson distribution with gamma replacing factorials
%   totdemand = totdemand + demand(v);
% end
%
% for v = 1:1:maxarr
%   demand(v) = demand(v)/totdemand;
% end
% % now sum of demand is = 1
% totdemand=0;
for v=1:maxarr-1
    demand(v) = exp(-Lambda) * (Lambda.^(v-1))/gamma(v);
    % formula for poisson distribution with gamma replacing factorials
    totdemand = totdemand + demand(v);
end
demand(maxarr) = 1- totdemand;

w = [0];
% incoming stock, w(1) arrives in 1 period, etc
% number of w must be Lambda-1

% cost calculation
% cost parameters: all costs are linear
    % holding cost per unit, cannot be negative
    global h;  h = 1;
    % ordering cost per unit
    global c;  c = 3;
    % penalty cost per unit, cannot be negative
    global p;  p = 5;

% decision variable, amount to order, z must be >= 0
% initialized to 0
% z = zeros(1,N);
global max; max = 20;
% maximum order quantity
% this is an artificial limit currently used in the algorithm
% drop later with improved algorithm

% stores current and best solutions
global zcurrent zbest;
zcurrent = zeros(1,N);
zbest = zeros(1,N);

%============================================================================
% main program: just call function f
%============================================================================
tic;
% clock starts ticking
% call main function f
cost=0;
[zmin, cost]= f(N, x, w);

%============================================================================
% display results
function [zmin, cost] = f(N,x,w)
    global Lambda;
    global alpha;
    global h;
    global c;
    global p;
    global max;
    global maxarr;
    global demand;
    % f is the minimization function in 1 variable,
    % it calls itself recursively
    % it has 2 results, zmin is the minimal order z
    % cost is the minimal total cost
    %==============================================================================
    disp('Clark & Scarf results')
    disp('----------')
    disp('minimum z')
    disp(zmin)
    disp('----------')
    disp('minimum cost is')
    disp(cost)
    disp('----------')
    disp('best order quantities from 1 to N')
    disp(zbest);
    toc
    % elapsed time since tic
    %==============================================================================
global zcurrent zbest;

% terminates if N=1
if N == 1
    exp = 0; % expectation
    for v= 1:1:maxarr
        exp = exp + demand(v)*periodcost(1,x,w,v-1,0);
    end
    cost = exp;
    zmin = 0;
    zcurrent(N) = zmin;
    return
end

% these are local variables contained in each call to function f
% initialize to zero
xn = zeros(1,N-1);
wn = zeros(1,alpha-1);
% t = zeros(1,N-1);
% cost is initialized to a very large number
cost = Inf;

% The following uses Sequential search
% Consider: Bisection or gradient method instead
% zt is a trial value for z
% starts from 0 going up to max
% loop stops when the next integer value produces a higher cost
% this assumes that the function to be minimized is unimodal
for zt = 0:1:max
    exp = 0;
    for v=1:1:maxarr
        [xn, wn] = nextstate(N,x,w,v-1,zt);
        % nextstate is the system dynamics of calculating next period
        [zminn, costn] = f(N-1, xn,wn);
% this is the recursive call to itself with a smaller N-1
exp = exp + demand(v) * (costn + periodcost(N, x, w, v-1, zt));
end

costnew = exp;
zcurrent(N) = zt;
if costnew < cost
  % better solution found
  zmin = zt;
  cost = costnew;
  zbest = zcurrent;
else
  % quit when it gets worse
  break;
end
end

% calculation of period costs is straightforward
% costs are linear except that they cannot be negative
% t is a scalar, = demand at period k
function L = periodcost(k, x, w, t, zi)
  ordercost = zi * c;
  if x(k) <= 0
    holdingcost = 0;
  else
    holdingcost = h * x(k);
  end
  if x(k) >= t
    penaltycost = 0;
  else
    penaltycost = p * (t-x(k));
  end
end
L = ordercost + holdingcost + penaltycost;
end

% this function is where the next state updating is done
% t is a scalar, = demand in the current period
function [xn, wn] = nextstate(N,x,w,t,zi)
    for s = N-1:-1: 1
        xn(s) = x(s+1) + w(1) - t;
        for j = alpha-1:-1:1
            if j == (alpha -1)
                wn(j) = zi;
            else
                wn(j) = w(j+1);
            end
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Finite Horizon Algorithm (FHA)
% By Alp Muharremoglu and John Tsitsiklis (Rev 2003)
% Matlab version by Jennifer Kusuma, Univ of Melbourne
% date: Oct 2005
%==================================================================
% VERSION 1.2
%==================================================================
% in addition to the assumptions in the paper, we assume:
% 1. only 1 state Markov chain, i.e. it is not Markov modulated,
% (in the following the state is omitted)
% 2. the external customer demand is Poisson with parameter Lambda
3. the lead times are stochastic with uniform distribution

clc;
clear;
disp('FHA Version 1.2 deterministic and stochastic cases, Please wait...')
disp(' ')
answer = input('deterministic (0) or stochastic (1)?');

% size and other parameters
% Because in Matlab all matrix indices start from 1, we have shifted all
% indices in the paper by 1
% index 0 in the paper becomes 1 here, index 7 in the paper becomes 8 etc
% N = 11; % NUMBER of locations
% example of a small configuration
N = 5;
% in the example in Fig 1, location 8 is the outside supplier
% this N is not N in the paper, since we start from 1 and include the
% outside supplier (see the bounds in the loop below)
% M=100;
M = 5; % maximum number of customer distances
% this number is to be set by trial and error
% if too small, loop will produce error
global A; % declaring it global so that A can be used in f.m and Actual.m
% A = [1,2, 5, 8, 10, 11]; % SET of actual locations, i.e. stages including outside supplier
A = [1,2,5]; % 1 is the customer, 2 the actual inventory, 3 supplier
LA = length(A);  % NUMBER of stages  
% note that A defines v(z)= the first actual location in decreasing order  
% starting from z, here v(8)=5, v(5)=2  
% definition of v(z) in the function v.m  

T = 5;  % TIME limit of the finite horizon  
U = 2;  % U is the number of possible decisions, 2=release, 1 =hold  
% in the paper U is 1 or 0  

% demand model  
global Lambda; % declared global so that it can be used in f.m  
Lambda = 3  ;  
% the demand d(t) of the Poisson distribution is generated  
% by the subfunction randpois(lambda) in f.m  

% cost calculation  
% the one time cost is defined by the function g(z,y,u)  
% cost parameters: all costs are linear  
% holding cost for each location  
global h;  
h = [ 0 1 0 0 0 ];  
% h = [ 0 1 0 0 1 0 0 1 0 1 0 ];  
% holding cost at location 1(given to customer)  
% and 8 (outside supplier) are zero  

% ordering cost for actual stages  
global c;  
c = [ 3 3 0 0 0 ];  
% c = [ 0 3 0 0 3 0 0 3 0 3 3 ];  
% order cost at location 1 is zero
% backorder cost assumed constant

global b;
b = 5;

% initialization of algo

Jstar = zeros(T,N,M);
% Jstar is the optimal cost-to-optimal from time t to T, for all z and y
% the values of Jstar will be computed by the algorithm

V = zeros(T,N,M,U); % cumulative cost to be minimized
% capital V is not v(z)
% V = g + E(Jstar(f))
% f is system dynamics mapping the current location
% f is influenced by the random disturbance
% g is the one time cost function
% E expectation
% V depends on U, the decision to release or hold
% computation of V is done in the recurrent loop
% with function calls to f.m and g.m

% ystar = repmat(-Inf,T,LA);
ystar = repmat(0,T,LA);
% repmat creates a matrix by repeating -Inf for all elements
% there does not seem to be any difference if Inf is replaced by 0
% ystar = threshold values to be computed
% initialized to very large negative number (negative infinity)
% ystar can be used in a base stock policy after it has been computed
% note that ystar is only defined if z is in A

K = zeros(T,LA); % creates a matrix of zeros,
%K is an indicator variable = 0 if undetermined, else 1
% row 1 and column 1 of K =1
% note that ystar is only defined if z is in A
K(1,:)=1;
K(:,1)=1;

y = 0; %counter starts from 0, incremented at each loop step

%====================================================================
% initialization of the transition probabilities of customer locations
%====================================================================
% transition table of customer locations using Poisson distribution
% given y at time t, the next y could be y, y-1,......2
% the next y can never be 1 except when y =2 or y=1
% see enclosed excel table ytrans.xls
% the total of all row probabilities =1
ytrans = zeros(M,M); %transition probabilities

for row = M:-1:3 % loop decreasing by -1 from M to 3
    for column = row:-1:3
        dif = row-column;
        ytrans(row,column) = exp(-Lambda)*(Lambda.^dif)/gamma(dif+1);
        % .^ is power operator
        % gamma is a system function in Matlab
        % gamma is almost like factorial, gamma(n)=factorial(n-1)
        % except that gamma is also defined for non-integers
    end
    ytrans(row,2) = 1 - sum(ytrans(row,3:row));
% initialization of the transition probabilities of item locations
% case release, u=2

% transition table of stochastic item locations
% given z at time t, the next z could be z, z-1,......v(z)
% we assume that probability distribution is uniform
% ztrans(row,column) is the probability of transition from row to column
% assuming that the decision in not to hold item
% defined only for row > column, see enclosed excel table ztransrel.xls
% the total of all row probabilities =1

ztransrel = zeros(N,N); % transition probabilities

if answer == 0 % case deterministic
    for row = N:-1:2
        ztransrel(row,row-1) = 1;
    end

else % case stochastic
    for row = N:-1:3 % loop decreasing by -1 from N to 3
        if ismember (row, A)
            nextstage= v(row); prob = 1/(row - nextstage);
        end

        end

end
for column = row-1:-1:nextstage
    ztransrel(row,column) =prob;
end
else
    ztransrel (row, row-1)= 1;
end
end

ztransrel(1,1)=1;ztransrel(2,1)=1;ztransrel(2,2)=0;
end

%====================================================================%
% initialization of the transition probabilities of item locations
% case hold, u=1
%====================================================================%
% defined only for row > column, see enclosed excel table ztransrel.xls
ztranshold = zeros(N,N); %transition probabilities
if answer == 0 % case deterministic
    for row = N:-1:1
        if ismember(row, A)
            ztranshold(row,row) =1;
        end
    end
else % case stochastic

    for row = N:-1:3 % loop decreasing by -1 from M to 3
        if ismember(row, A)
            ztranshold(row,row)=1;
        else
            ztranshold (row, row-1)= 1;
        end
    end

end
end
end
ztranshold(1,1)=1; ztranshold(2,1)=0; ztranshold(2,2)=1;
end

% ==============================================================
% recurrence loop
% ==============================================================
tic;
while min(min(K)) == 0 & y < M

    % if min(K) = 0 then some element of K = 0
    % == is the equal sign in Matlab
    % = is used for assignment
    % the stopping criteria is if all elements of K are nonzero and y=N
    % in the paper the criteria y = M is not used
    % it has been added here to avoid programming error:
    % "index out of bounds"
    y = y + 1;
    % increment counter
    for t = 2:T
        for z = 2:N
            % starts from 2 since 1 is 0 in the paper
            if ismember(z,A) & K(t,Actual(z))==0
                % case u = 1, hold
                % [z1,y1]= f(t,z,y,1); % next state of the system if u=1
                ytrans(2,2)=1;  ytrans(2,1)=0;
                % no transition since U=1
                % the following loop computes the expectation
                % compute V as g + expectation(f)
                Expect = 0;

```
for ya = 1:y
    % Expect = Expect + Jstar(t-1,z1,column)*ytrans(y,column);
    for za = 1:z
        Expect = Expect+Jstar(t-1,za,ya)*ytrans(y,ya)*ztranshold(z,za);
    end
end

t(t,z,y,1)= g(z,y,1) + Expect;

% case u = 2, release
% [z2,y2]= f(t,z,y,2); % next state of the system if u=2
% in this case U =2, we must test z=2
if z==2
    ytrans(2,2)=0;
    ytrans(2,1)=1;
else
    ytrans(2,2)=1;
    ytrans(2,1)=0;
end
% the following loop computes the expectation
Expect = 0;
for yb = 1:y
    for zb= 1:z
        Expect = Expect+Jstar(t-1,zb,yb)*ytrans(y,yb)*ztransrel(z,zb);
    end
end

t(t,z,y,2)= g(z,y,2) + Expect;

if t(t,z,y,1) < t(t,z,y,2) % minimum when u=1
    Jstar(t,z,y) = t(t,z,y,1); % optimal cost-to-go
APPENDIX C. MATLAB CODE

K(t,Actual(z)) = 1; % mark K(t,z) as computed
else % minimum when u=2
    Jstar(t,z,y) = V(t,z,y,2); % optimal cost-to-go
    ystar(t,Actual(z)) = y; % threshold value
end

else
    Jstar(t,z,y) = V(t,z,y,1); % optimal cost-to-go
end
end
end

% display results
%==============================================================
disp('Finite Horizon Algorithm Results')
disp('----------')
disp('ystar threshold values')
disp(ystar)
disp('----------')
disp('y maximum')
if y == M
    disp('try again with larger value of M')
end

% if y maximum is = M then it stops because M has been reached, the result
% may not be optimal, try again with larger M
disp(y)
if y == M
    disp('try again with larger value of M')
else
    disp('iteration completed')
end
toc

% Actual(z)
% function to map z to an index of the set A
% e.g. Actual(2)=1, Actual(5)=2, Actual(8)=3
% undefined if z in not in A

function loc = Actual(z)
global A;
% A= [2, 5, 8, 10, 11]; % SET of actual locations, i.e. stages including outside supplier
[tf,loc] = ismember(z,A);
% if z is a member of A then z = A(loc)
end

% one time cost function g(z,y,u)
% this function is independent of t
% it implements formula 4 in section 6.1 of the paper
% indicator functions are replaced by if statements
function gresult = g(z,y,u)
% cost parameters: all costs are linear
% see main program for values
global h;
global c;
global b;
u1 = u - 1; % since the value of u is 1 or 2
% u1 is the original u in the paper
if y==1
    gresult = 0;
APPENDIX C. MATLAB CODE

```matlab
end
if z==1
    gresult = 0;
end
if y==2
    if z==2
        gresult = h(z) - h(z)*u1 + b*(1-u1) + c(z)*u1;
    else
        gresult = h(z) + b + c(z)*u1;
    end
else
    gresult = h(z) + c(z)*u1;
end

% v(z)
% given z, v(z) is the highest actual stage lower than z
% as in the paper
% undefined if z in not in A

function nextstage = v(z)
global A; LA = length(A);
% eg A = [2, 5, 8, 10, 11]
% v(10)=v(9)=5   v(2)=1 v(1)=1
    if z == 1
        nextstage = 1;
    end
    for search = LA:-1:1
        if z == A(search) % found
            if search < 2
                nextstage = 1;
            break;
        end
    end
end
```
else
    nextstage = A(search -1);
end
break;
else
    if z > A(search)
        nextstage = A(search);
        break;
    end
end
end
end
Bibliography


