APPROXIMATE DYNAMIC PROGRAMMING ALGORITHMS FOR PRODUCTION-PLANNING PROBLEMS

A Thesis by

Ning Liu

Bachelor of Science, Xi'an University of Science and Technology, 2010

Submitted to the Department of Industrial and Manufacturing Engineering
and the faculty of the Graduate School of
Wichita State University
in partial fulfillment of
the requirements for the degree of
Master of Science

December 2013

APPROXIMATE DYNAMIC PROGRAMMING ALGORITHMS FOR PRODUCTION-PLANNING PROBLEMS

The following faculty members have examined the final copy of this thesis for form and content, and recommend that it be accepted in partial fulfillment of the requirement for the degree of Master of Science with a major in Industrial and Manufacturing Engineering.

Esra Büyüktahtakın, Committee Chair

Janet Twomey, Committee Member

Thomas K. DeLillo, Committee Member

ACKNOWLEDGEMENTS

I would like to express my gratitude to my advisor, Dr. Esra Büyüktahtakın. She has always supported me with great patience. During my two years of study at Wichita State University, despite my poor ability to do research, she has never given up on me. Instead, she has always given me inspiration and excitation. She has been ready to discuss issues and has helped me whenever I needed help, and her attitude has encouraged me in both my work and my life. Thanks for everything, Dr. Esra.

It is my honor to have Dr. Janet Twomey and Dr. Thomas K. DeLillo as my committee members. Their great feedback and discussions have guided me to significantly improve my research. I also thank my colleagues Halil Cobuloglu, Emanuel des-Bordes, Alperen Kantas, Rajesh Narasimhan and Eyyub Kibis, for their support during the past two years. The times they shared with me have been wonderful.

Finally, I give special thanks to my parents who have always trusted me in what I am doing. Without their financial support and trust, I would not be able to finish this research.

ABSTRACT

The capacitated lot-sizing problem (CLSP) is a core problem for successfully reducing overall costs in any production process. The exact approaches proposed for solving the CLSP are based on two major methods: mixed-integer programming and dynamic programming. This thesis provides a new idea for approximating the inventory cost function to be used in a truncated dynamic program for solving the CLSP. In the proposed method, by using only a partial dynamic process, the inventory cost function is approximated, and then the resulting approximate cost function is used as a value function in each stage of the approximate dynamic program.

In this thesis, six different algorithms are developed for the CLSP, based on three different types of approximate dynamic programming approaches. The general methodology combines dynamic programming with data fitting and approximation techniques to estimate the inventory cost function at each stage of the dynamic program. Furthermore, three main algorithmic frameworks to compute a piecewise linear approximate inventory cost function for the CLSP are provided. The first approach integrates regression models into an approximate dynamic program. The second approach uses the information obtained by a partial dynamic process to approximate the piecewise linear inventory cost function. The third approach uses slope-check and bisection techniques to locate the breakpoints of the piecewise linear function in order to approximate the inventory cost function for the CLSP.

The effectiveness of the proposed methods are analyzed on various types of CLSP instances with different cost and capacity characteristics. Computational results show that approximation approaches could considerably decrease the computational time required by the dynamic program and the integer program for different CLSP instances. Furthermore, in most cases, some of the proposed approaches can accurately capture the optimal solution of the problem.

TABLE OF CONTENTS

Chapter **Page**

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

Introduction

Production planning and management is critical for companies to reduce their overall production costs. The purpose of production planning is to ensure that manufacturing runs effectively and efficiently and that production is sufficient to meet the demand of customers with the lowest possible cost. However, due to changing demand, capacities, and other factors in the long-term planning horizon, it is difficult for manufacturers and managers to make cost-effective decisions regarding how much to produce and how much inventory to keep in storage in a production period. The problem of determining the quantity of production for a single item in each production period within a limited production capacity is known as the single-item capacitated lot-sizing problem (CLSP). The CLSP is a core problem for successfully reducing overall costs in any production process.

More formally, the CLSP can be described as follows: For a finite time horizon T, production and inventory levels are determined for each time period in order to meet periodic demands within the production capacity limits without backlogging so that the sum of periodic setup, production, and inventory costs are minimized. The CLSP is NP-hard [1] and forms the basis of many production planning and inventory problems.

For each period $t \in [1, T]$, let p_t, f_t, and h_t denote the per-unit production, setup, and inventory costs for period t, respectively, and let x_t, y_t, and s_t be the production, binary setup, and ending inventory variables for period t, respectively. Then, the CLSP can be formulated as

$$\min \sum_{t=1}^{T} (p_t x_t + f_t y_t + h_t s_t) \tag{1.1}$$

subject to the following:

$$s_{t-1} + x_t - d_t = s_t \qquad t = 1, \ldots, T \tag{1.2}$$

$$x_t \leq c_t y_t \qquad t = 1, \ldots, T \tag{1.3}$$

$$s_t, x_t \geq 0 \qquad t = 1, \ldots, T \tag{1.4}$$

$$y_t \in \{0, 1\} \qquad t = 1, \ldots, T. \tag{1.5}$$

where d_t is the demand, and c_t is the production capacity in period t. Note that the initial inventory is assumed to be zero while it can take any positive value. This model and its variants have been extensively studied in the literature.

Lot sizing is a fundamental problem in optimization that has attracted the wide interest of both practitioners and researchers. Its importance stems from its applications in production/inventory planning and supply chain management. As mentioned in earlier studies [2, 3, 4, 5], the structure of the single-item lot-sizing problem has been at the core of production-planning problems involving multiple products and levels over a finite discrete time horizon. Therefore, the development of effective solution algorithms for the lot-sizing problem is important, and any results found for the CLSP can be used to develop solution methods for more complicated lot-sizing problems.

Most of the exact approaches proposed for the lot-sizing problem are based on integer and dynamic programming algorithms (for a detailed review see, e.g., the work of Pochet and Wolsey [6]). In this paper, approximate approaches based on the dynamic programming (DP) are proposed. The basic idea of the developed approximate approaches is as follows: In each period, a truncated version of the DP formulation of the CLSP is used to compute the inventory costs for a subset of all possible inventory levels. By using the information from those inventory levels that are solved exactly, the inventory cost function is approximated in order to solve for all possible inventory levels. Section 4.1 provides the approach that uses regression to approximate the inventory cost function in each period of the planning horizon. Section 4.2 provides the direct-connection approach, which approximates the inventory cost

function based on piecewise linear functions obtained by a subset of inventory cost values computed using exact DP recursion. Section 4.3 provides the slope check and bisection approach, which combines the approximated piecewise linear function and DP method together to compute the inventory costs in each period.

This thesis is organized as follows: Chapter 2 provides a discussion of the contributions of this research with related previous research. Chapter 3 presents a dynamic programming formulation for the CLSP, while in Chapter 4, approximate dynamic programming (ADP) approaches are proposed to solve the CLSP. Experimental studies to show the effectiveness of the proposed ADP approaches with computational results are presented in Chapter 5.

CHAPTER 2

Literature Review

In this chapter, we give a brief review of the related literature on the lot-sizing problem in three major categories: mixed-integer programming (Section 2.1), dynamic programming (Section 2.2), and approximate dynamic programming algorithms (Section 2.3). These are presented in the following subsections.

2.1 Mixed-Integer Programming Approaches to Lot-Sizing Problems

One important methodology used to solve many multi-period production planning and supply chain management models is to formulate them as mixed-integer programs (MIPs). In this approach, given a certain planning horizon and the model parameters, which are assumed to be known and deterministic during this horizon, the MIP model is solved using branch-and-bound and branch-and-cut tree-based search algorithms [7]. Pochet and Wolsey [6] provide a very nice review of MIP approaches used for production planning problems.

Barany et al. [8] propose valid inequalities for the single-item lot-sizing problem, which they then use for multi-item problems. Constantino [9] provides cutting planes based on submodular inequalities for the CLSP with start-up costs, and uses these cuts for the multi-item version of the problem. Pochet and Wolsey [2] and Belvaux and Wolsey [10, 11] provide formulations and valid inequalities for the multi-item and multi-stage lot-sizing problems. Miller et al. [12] give a tight formulation for the multi-item lot-sizing problem with constant demands and setup times. For a good classification for both single-item and multi-item lot-sizing problems, the reader is referred to the work of Wolsey [3]. In another study, Hartman et al. [13] propose inequalities for the CLSP that are derived from the end-of-stage optimal solutions of a DP algorithm for the CLSP problem. Later, Büyüktahtakın et al. [14] extend these inequalities to solve the multi-item CLSP.

2.2 Dynamic Programming Approaches to Lot-Sizing Problems

Bellman and Kalaba [15] introduced dynamic programming to solve Markov decision processes, which are powerful analytical tools used for sequential decision making under uncertainty ([16]). In a dynamic programming approach, the problem is attacked by decomposing it into a sequence of interrelated subproblems defined by a recursive function and by solving solve the smallest subproblem. The problem is then enlarged by obtaining the current optimal solution from the preceding subproblem until the original problem is solved in its entirety [17]. In particular, for solving sequential decision-making engineering problems, dynamic programming has been widely used.

Variants of the CLSP have been studied in the dynamic programming literature. While Wagner and Whitin [18] were the first to provide an $O(T^2)$ algorithm for the uncapacitated lot-sizing problem (ULSP), later $O(T \log T)$ algorithms for the same problem were developed [19, 20, 21]. Florian et al. [1] present a DP algorithm with complexity $O(D_T C_T)$ for the CLSP, where $D_T = \sum_{t=1}^{T} d_t$ is the cumulative demand, and $C_T = \sum_{t=1}^{T} c_t$ is the cumulative capacity over all periods. For the CLSP, Chen et al. [22] develop a continuous dynamic programming algorithm that is exponential in complexity but has been demonstrated to be computationally efficient in practice. VanHoesel and Wagelmans [23] give fully polynomial approximation schemes for the CLSP. For the same problem, Baker et al. [24] provide a branch-and-bound algorithm, while Chung et al. [25] present a hybrid branch-and-bound and DP algorithm, [26]. Lagrangian relaxation approaches, which decompose the CLSP into a set of uncapacitated single-item lot-sizing problems by relaxing capacity constraints, are applied to solve the problem in the literature [27, 28, 29, 30]. Lot-sizing problems with different special cases including other heuristic solution approaches such as metaheuristics and column generation techniques are reviewed in detail [31, 32, 33].

2.3 Approximate Dynamic Programming Approaches to Lot-Sizing Problems

Dynamic programming has often been dismissed because it suffers from the curse of dimensionality: as the problem size increases, the size of the state space grows exponentially

(Powell [34]). Therefore approximation algorithms are developed based on MIP and DP formulations of the CLSP to get over this difficulty. In 1963, a polynomial DP approximation algorithm is introduced by Bellman et al. [35] for allocation processes. Kleywegt et al. [36] propose approximation methods to find good solutions with reasonable computational effort for a stochastic inventory routing problem that is similar to the CLSP. Toriello et al. [37] provide a valuing inventory function with concave piecewise lines and combine solutions to single-period subproblems of the inventory routing problem using dynamic programming techniques. The authors show that their approach reduces the computational time without losing the optimality of the solution. Levi et al. [38] provide an approximation algorithm for the multi-item CLSP based on generating flow-cover inequalities and randomized rounding. An ADP algorithm is studied for the economic lot scheduling problem by Adelman and Barz [39].

In ADP algorithms, the regression technique is an important tool to approximate value functions. Muggeo [40] provides a method for estimating regression models with unknown breakpoints, which are values whose effect on the response changes abruptly. On the other hand, Hudson [41] provides an important least-square technique for regression constrained to be non-negative, non-decreasing or convex. The non-decreasing property of the fitted function is an important tool for ADP algorithms for CLSP due to the non-decreasing property of the actual inventory cost function.

2.4 Contributions of Paper

In this thesis, we provide the following five main contributions. First, we provide a new idea of approximating the inventory cost function to be used in a truncated dynamic program for solving CLSP. In the proposed method, by using only a partial dynamic process, the inventory cost function is approximated, and then the resulting approximate cost function is used as a value function in each stage of the approximate dynamic program.

Second, we present an algorithmic framework to compute a number of non-decreasing piecewise linear approximate inventory cost functions for solving the CLSP. The approximate

cost function allows for estimating the cost value of each inventory level in a given stage of the DP through a combination of sampling, data-fitting and approximation techniques without sacrificing solution quality. Furthermore, we propose three main approaches to compute a piecewise linear approximate inventory cost function for the CLSP. The first approach integrates regression models into an approximate dynamic program. The second approach uses the information obtained by a partial dynamic process to approximate the piecewise linear inventory cost function. The third approach uses slope-check and bisection techniques to locate the breakpoints of the piecewise linear function in order to approximate the inventory cost function for the CLSP.

Third, we develop a new type of approximate approach, which estimates the inventory cost function using a subset of inventory levels in a stage of the DP for solving the CLSP. In the literature of capacitated lot-sizing problems, a number of approximate approaches based on dynamic programming and integer programming have been studied as discussed in this chapter. Most of the ADP approaches focus on approximating the recursion function for the dynamic process. Instead of approximating the recursion function, in this thesis, a truncated version of the original recursion function is used but the inventory cost function of the DP is approximated using a subset of inventory levels in a stage of the DP.

Fourth, we perform computational experiments to analyze the effectiveness of the proposed methods on various types of CLSP instances with different cost and capacity characteristics. Experimental results show that some of the proposed ADP approaches are very efficient, both in terms of solution quality and solution computation time. In particular, results demonstrate that ADP slope-check and bisection methods save considerable solution times for different types of lot-sizing instances with different characteristics without losing optimality, when compared to the corresponding DP and MIP formulation solutions.

Fifth, we believe that a similar technique can be employed for solving other problems aside from lot-sizing, such as knapsack and parallel equipment replacement problems, for which DP approaches exist. Furthermore, because CLSP is a special type of capaci-

tated fixed-charge problem, proposed ADP algorithms can be extended to develop solution algorithms for the general fixed-charge problem. The long-term goal is to generalize the implementation of proposed ADP approaches to as many problems as possible.

CHAPTER 3

DP Approach to CLSP

The DP approach in this thesis is similar to the dynamic programming approach of Hartman et al. [13], who define the state of the system in period t as the inventory level at time t. In an earlier thesis, Florian et al. [1] define the state of the system in period t as the cumulative level of production through time t.

The term $F_t(s_t)$ is defined as the minimum cost of making optimal production decisions through period t with an ending inventory level s_t. Note that periodic production decisions must meet demand d_t, and production levels must not exceed production capacity limits c_t, in each period. Note that the minimum inventory level at period $t = 1, \ldots, T$ in any feasible solution is given by

$$L_t = \max \left\{ 0, \max_{\tau=t+1,\ldots,T} \sum_{j=t+1}^{\tau} (d_j - c_j) \right\}, \tag{3.1}$$

because inventory must always be nonnegative and sufficient to cover future demands if capacity in future periods is insufficient to cover these demands.

Similarly, by defining $C_t = \sum_{j=1}^{t} c_j$ and $D_t = \sum_{j=1}^{t} d_j$, $\forall t = 1, \ldots, T$, the maximum inventory level at period $t = 1, \ldots, T$ in any optimal solution is given by

$$U_t = \min \left\{ C_t - D_t, \sum_{j=t+1}^{T} d_j \right\}, \tag{3.2}$$

where the first term of equation (3.2) defines the maximum inventory that could accumulate after t periods, and the second term gives the cumulative total demand in future periods. Because this is a finite horizon problem, zero inventory remains after the final period of production in an optimal solution.

Let $S_t = \{L_t, \ldots, U_t\}$. Note that in period $t = 1, \ldots, T$, given inventory level $L_t \leq s_t \leq U_t$, production in any optimal solution at period t lies in the set $X_{t,s_t} = \{\max\{0, s_t +$

$d_t - U_{t-1}\},\ldots,\min\{c_t, s_t + d_t - L_{t-1}\}\}$. By setting $F_0(0) = 0$ (since it is assumed that $s_0 = 0$), the forward dynamic programming recursion can now be written as

$$F_t(s_t) = \min_{x_t \in X_{t,s_t}} \{p_t x_t + f_t y_t + h_t s_t + F_{t-1}(s_t + d_t - x_t)\}\,,\ \forall t = 1,\ldots,T,\ L_t \le s_t \le U_t, \tag{3.3}$$

where $y_t = 1$, if $x_t > 0$, and $y_t = 0$, otherwise. The optimal objective function value is defined as $F_T(0)$. The complexity of this algorithm is $O(TD_T^2)$ [13].

Figure 3.1 illustrates this DP technique for an instance of CLSP where $T = 4$, $c_t = (4, 3, 4, 5)$, $d_t = (2, 3, 3, 3)$, $p_t = (1, 2, 1, 1)$, $f_t = (8, 7, 6, 7)$, and $h_t = (1, 1, 1, 1)$. As can be seen, each node represents a feasible inventory level in each period, and each node is labeled by the F-function value representing the minimum cost to have the corresponding inventory level at the end of the period. A connection between two nodes represents a feasible production activity. For example, $F_1(0)$ is not connected to $F_2(2)$ because the capacity in period 2 is too low to produce enough products for the connection of the two nodes (i.e., $d_2 + 2 = 5 > c_2 = 3$). As shown in the graph, the best solution is obtained by setting $s_t = (2, 2, 3, 0)$, and the minimum cost of production, setup, and inventory is 42.

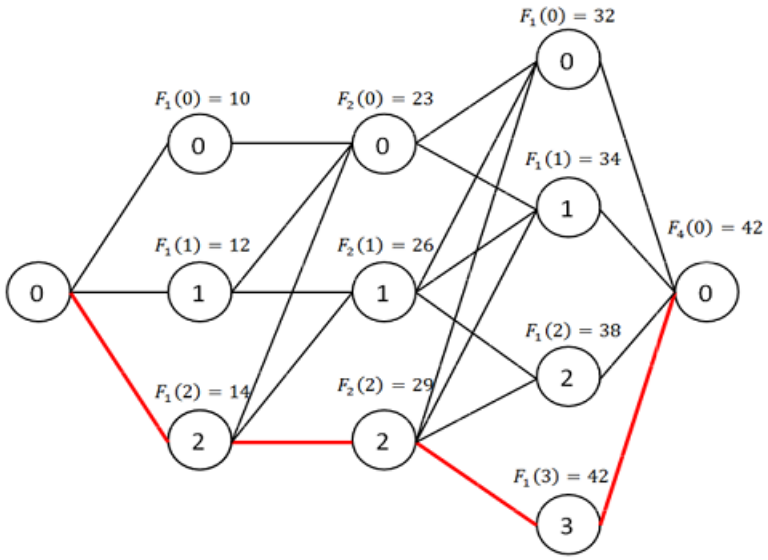

Figure 3.1: Graphical representation of forward dynamic programming recursions.

CHAPTER 4

Proposed ADP Approaches to CLSP

In this thesis, approximate algorithms based on the dynamic program for the CLSP given in Chapter 3 are investigated. When the production is continuous, the inventory function is found to be piecewise linear [22]. In this case, the production and inventory levels are discrete. However, a piecewise linear function can still be stated on the costs accrued through time t as a function of the inventory at stage t. In particular, the aim here is to design ADP algorithms to approximate those piecewise lines and calculate the cost of the inventory levels in stage t by the approximated piecewise linear function in order to reduce the solution time of the DP.

Several different ADP approaches are proposed in this thesis. The main idea behind these approaches is to use some of the inventory levels to estimate the inventory cost function and calculate the cost of other inventory levels by the estimated function. After computing the cost values for those points using a truncated version of DP, the DP recursion equation (3.3) is modified in a way that the minimum cost for stage t is calculated based on the approximated costs through the first $t-1$ periods instead of actual optimal costs for the first $t-1$ periods. Once the minimum costs for the sampled points are calculated using the truncated DP, an approximate inventory cost function based on these points is fitted by using techniques such as regression, direct connection of piecewise lines, and connection of lines through checking slopes of the lines. Then using this approximate function, the cost values, which are not computed for the other inventory levels, can be estimated. Therefore, inventory levels are divided into two sets: real points or sampled points whose cost values are calculated by the truncated DP, and other approximate points whose cost values are estimated by the generated approximate inventory cost function. Let $s_{t,a}$ represent the inventory level for sampled points in stage t, where $s_{t,a} \in S_t$ and a is the a^{th} sampled point.

To determine the number of sampled points, the following equation is used:

$$N_t = \alpha * (U_t - L_t + 1) \tag{4.1}$$

where α is the percentage of the total possible inventory levels in period t, whose cost values will be computed by the truncated DP. Note that $|S_t| = U_t - L_t + 1$.

To determine the location of sampled points, the following equation is applied:

$$s_{t,a} = \begin{cases} L_t & \text{if } a = 1; \\ \left\lfloor a * (U_t - L_t) * \frac{1}{N_t} \right\rfloor & \text{if } a = 2, \ldots, N_t - 1; \\ U_t & \text{if } a = N_t; \end{cases} \tag{4.2}$$

where, a is the order of the a^{th} sampled point, and $s_{t,a}$ represents the inventory level of the a^{th} sampled point in period t. Note that equation (4.2) is divided into three parts to ensure that the sampled points include the minimum inventory level L_t and the maximum inventory level U_t.

The term $\tilde{F}_t(s_{t,a})$ is defined as the minimum cost of having an ending inventory level $s_{t,a}$ through period t based on the approximated costs through the first $t-1$ periods. This term is the truncated version of the value function $F_t(s_{t,a})$ given in equation (3.3). The following recursion is used to compute the minimum approximate cost in order to have an $s_{t,a}$ inventory level in period t for $a = 1, \ldots, N_t$:

$$\tilde{F}_t(s_{t,a}) = \min_{x_t \in X_{t,s_{t,a}}} \left\{ p_t x_t + f_t y_t + h_t s_t + \hat{F}_{t-1}(s_{t,a} + d_t - x_t) \right\}, \tag{4.3}$$

where $\hat{F}_{t-1}(s_{t-1})$ is the approximated inventory cost function value for s_{t-1}, in stage $t-1$.

Consider an example in which the possible inventory levels change from 0 to 99 in a given period t, and 10 percent of the total possible inventory levels is sampled, i.e., α is set to 10. Then, using equation (4.1), $N_t = 0.1 * (99 - 0 + 1) = 10$. From equation (4.2), $s_{t,1} = 0, s_{t,2} = 10, s_{t,3} = 20, ..., s_{t,10} = 100$. Then $\tilde{F}_t(s_{t,a})$ in computed for each $s_{t,a}$ for $a = 1, \ldots, 10$. Using equation (4.3), the corresponding $\tilde{F}_t(s_{t,a})$ values are computed and are used to approximate the inventory cost function in stage t, which is $\hat{F}_t(s_{t,a})$.

In this thesis, three different types of approximation approaches are introduced: in Section 4.1, an approach that involves approximation via regression; in Section 4.2, an approach that involves direct connection of real points; and in Section 4.3, an approach that involves a slope-check and bisection technique.

4.1 Regression Approximation Approach

Given that $\tilde{F}_t(s_{t,a})$ is the optimal cost value for the inventory level $s_{t,a}$, for every $s_{t,a}$, there is a two-dimensional coordinate system to represent a sampled point $(s_{t,a}, \tilde{F}_t(s_{t,a}))$ for $a = 1, \ldots, N_t$ for a given t. In the regression approximation approach, for any period t and $a = 1, \ldots, N_t$, using points $(s_{t,a}, \tilde{F}_t(s_{t,a}))$, a regression function is fitted, and values of the other inventory levels are calculated by the regression function.

For regression, the least squares method is used. That is, having real points $s_{t,a}$ for $a = 1, \ldots, N_t$, the following optimization problem is solved:

$$\min \sum_{a=1}^{N_t} \left(\tilde{F}_t(s_{t,a}) - \hat{F}_t(s_{t,a}) \right). \tag{4.4}$$

Two types of regression models are applied using this approach: linear and polynomial.

For linear regression, $\hat{F}_t(s_{t,a})$ is defined as

$$\hat{F}_t(s_{t,a}) = \beta_0 + \beta_1 * s_{t,a}, \tag{4.5}$$

where β_0 and β_1 are the parameters of the regression model and are defined as the unrestricted variables of the optimization model shown in equation (4.4).

For polynomial regression, $\hat{F}_t(s_{t,a})$ is defined as

$$\hat{F}_t(s_{t,a}) = \beta_0 + \beta_1 * s_{t,a} + \beta_2 * s_{t,a}^2 + \beta_3 * s_{t,a}^3 + \beta_4 * s_{t,a}^4 + \beta_5 * s_{t,a}^5, \tag{4.6}$$

where β_0 and β_1 are the parameters of the regression model and are defined as the unrestricted variables of the optimization model shown in equation (4.4).

For the linear regression approach, Figure 4.1 shows an example of how the approximate inventory function in a period t is obtained. As can be seen, the possible inventory

levels change from 0 to 100, and $A, B, \ldots, K$ represent the inventory cost values, $\tilde{F}_t(s_{t,a})$. Then the regression line obtained by the least squares error method using the optimization model (4.4) and (4.5) is

$$\hat{F}_t(s_{t,a}) = 29 + 0.4636 * s_{t,a}. \tag{4.7}$$

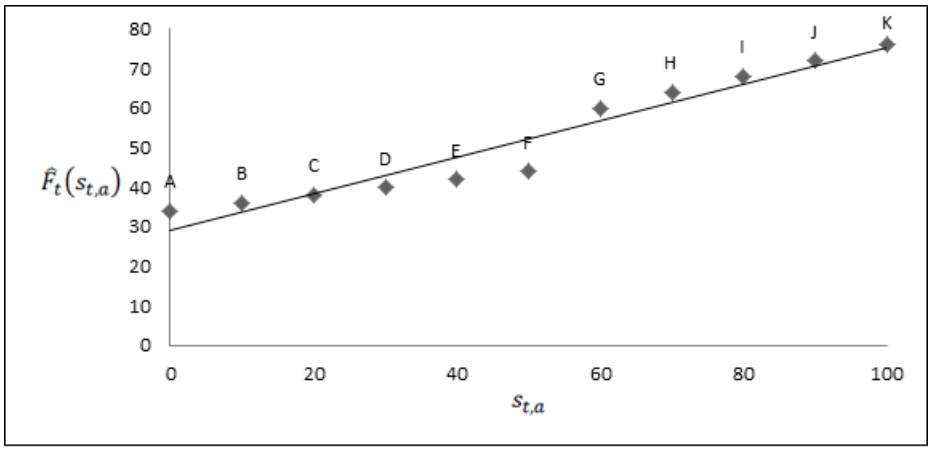

Figure 4.1: Example inventory cost function and corresponding regression line.

By replacing $s_{t,a}$ by s_t in the equation (4.7), the approximate inventory costs are computed for all inventories from 0 to 100, based on the following equation:

$$\hat{F}_t(s_t) = 29 + 0.4636 * s_t, \quad L_t \leq s_t \leq U_t. \tag{4.8}$$

Once $\hat{F}_T(0)$ is solved, the approximated cost values for all inventory levels in all periods are obtained; however, because those inventory costs are not calculated by the DP process but rather by the approximation function, the information on dynamic connections for many inventory levels has not been obtained. Therefore, the shortest-path problem is solved to find the shortest path from node $\hat{F}_T(0)$ to node $\hat{F}_0(0)$. For the example problem described in Chapter 3, after $\hat{F}_t(s_t)$ is obtained for all s_t, $t = 1, \ldots, T$ and $L_t \leq s_t \leq U_t$, the shortest-path problem is solved from the last period T to the first period 0. The corresponding shortest path is illustrated in Figure 4.2.

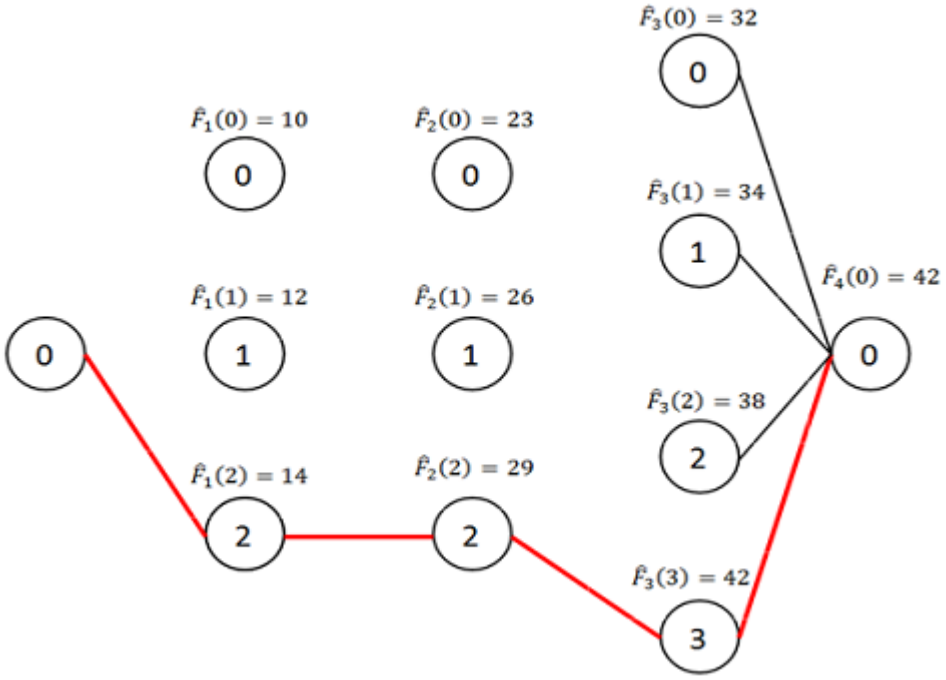

Figure 4.2: Example of shortest path from $\hat{F}_4(0)$ to $\hat{F}_0(0)$.

As shown in Figure 4.2, the $\hat{F}$ values for all inventory levels in all periods are depicted above each node. Now, the shortest-path problem from period 4 to period 1 is solved to find the minimum-cost path from node $\hat{F}_4(0)$ to $\hat{F}_0(0)$. Solving the recursion equation (4.3) based on the $\hat{F}_3(s_3)$ values such that s_3 has a production connection to $s_4 = 0$, the minimum-cost connection to $\hat{F}_4(0)$ is determined through $\hat{F}_3(3)$. Now, the problem becomes a smaller subproblem of finding the shortest path from $\hat{F}_3(3)$ to $\hat{F}_0(0)$. By repeating the same process recursively from the last period to the first period, the shortest path can be found, which is shown in the graph. This process, called the backtracking process (BP), is very important in order to find the dynamic connection for the ADP algorithms.

For the linear regression approach, the ADP algorithm is described in Algorithm 1 is given below. For the polynomial regression approach, the ADP algorithm is the same as Algorithm 1, except the regression function, where equation (4.5) is replaced with equation

Algorithm 1 ADP Linear Regression

Step 1. Set $t = 1$;
compute inventory cost function $F_1(s_1) = \{p_1 * (s_1 + d_1) + f_1 + h_1 s_1\}$, for each $L_1 \leq s_1 \leq U_1$; increment $t = t + 1$;
Step 2. Compute N_t using equation (4.1);
for $a = 1$ to N_t **do**
    Calculate inventory level $s_{t,a}$ using equation (4.2); solve DP recursion equation (4.3) to compute inventory cost $\tilde{F}_t(s_{t,a})$ for real point $s_{t,a}$;
end for
Solve regression optimization problem (4.4) to obtain the regression function (4.5);
for $s_t = L_t$ to U_t **do**
    Calculate inventory cost value $\hat{F}_t(s_t)$ based on equation (4.5);
end for
Increment $t = t + 1$;
Step 3. If $t < T$, then repeat Step 2; **else** call BP to find shortest path;
Step 4. Calculate the actual optimum cost of the shortest path, which is obtained from Step 3.

(4.6). This is called an ADP polynomial regression algorithm and is denoted as Algorithm 1′.

4.2 Direct-Connection Approach

In the direct-connection approach, instead of fitting a regression function, which involves an optimization process, information from the real points is used directly. The advantage of the direct-connection approach, compared to Algorithms 1 and 1′, is that this approach does not require an optimization process to find the approximate function and therefore saves computational time. Two algorithms are developed based on the direct-connection approach. In the first algorithm, a connection line is fitted for each two-nearest real points. Then, the values of the points, which are located between the two real points, are approximated using the computed line equation. The connection line between the two points is obtained by solving two linear equations information from the two real points. For example, suppose that the coordinates of two consecutive real points are $(s_{t,a_1}, \tilde{F}_t(s_{t,a_1}))$ and $(s_{t,a_2}, \tilde{F}_t(s_{t,a_2}))$. By solving the two linear equations $\tilde{F}_t(s_{t,a_1}) = \delta + m * s_{t,a_1}$ and $\tilde{F}_t(s_{t,a_2}) = \delta + m * s_{t,a_2}$, the value of δ and m of the line can be easily obtained. By repeating the

same process from the first real point to the last real point, a piecewise linear function can be obtained. This process is called the direct-connection process (DCP). The steps of the algorithm are shown in Algorithm 2. Figure 4.3 shows the approximate piecewise linear inventory cost function obtained from the real points for the example discussed in Chapter 4.

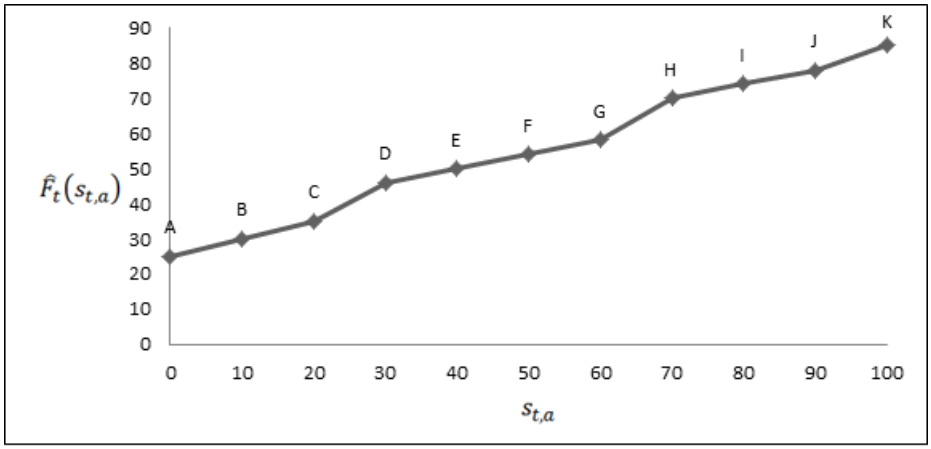

Figure 4.3: Example of an approximate inventory function via direct-connection approach.

In the second algorithm, using the direct-connection approach, instead of connecting two real points at a time, three consecutive points are connected through a quadratic equation. In this case, a quadratic equation is applied since the linear equation does not guarantee the connection of three points. Suppose the coordinates of three consecutive real points are $(s_{t,a_1}, \tilde{F}_t(s_{t,a_1}))$, $(s_{t,a_2}, \tilde{F}_t(s_{t,a_2}))$, and $(s_{t,a_3}, \tilde{F}_t(s_{t,a_3}))$. By solving the following three linear equations:

$$\tilde{F}_t(s_{t,a_1}) = \alpha_0 + \alpha_1 * s_{t,a_1} + \alpha_2 * s_{t,a_1}^2,$$

$$\tilde{F}_t(s_{t,a_2}) = \alpha_0 + \alpha_1 * s_{t,a_2} + \alpha_2 * s_{t,a_2}^2,$$

$$\tilde{F}_t(s_{t,a_3}) = \alpha_0 + \alpha_1 * s_{t,a_3} + \alpha_2 * s_{t,a_3}^2,$$

the values of α_0, α_1, and α_2 can be easily obtained to find the quadratic function connecting the three points s_{t,a_1}, s_{t,a_2} and s_{t,a_3}. This algorithm is denoted as Algorithm 2′. The steps of Algorithm 2′ are the same as those in Algorithm 2, except the approximate function including quadratic functions, which are obtained by connecting three consecutive real points at a time.

Algorithm 2 ADP Direct Connection

Step 1. Set $t = 1$;
compute inventory cost function $F_1(s_1) = \{p_1 * (s_1 + d_1) + f_1 + h_1 s_1\}$, for each $L_1 \leq s_1 \leq U_1$; increment $t = t + 1$;
Step 2. Compute N_t using equation (4.1);
for $a = 1$ to N_t **do**
    Calculate inventory level $s_{t,a}$ using equation (4.2); solve DP recursion equation (4.3) to compute inventory cost $\tilde{F}_t(s_{t,a})$ for real point $s_{t,a}$;
end for
Obtain approximate inventory cost function by DCP process;
for $s_t = L_t$ to U_t **do**
    Calculate inventory cost value $\hat{F}_t(s_t)$ based on approximate inventory cost function obtained by DCP process;
end for
Increment $t = t + 1$;
Step 3. If $t < T$, then repeat Step 2; **else** call BP to find shortest path;
Step 4. Calculate the actual optimum cost of the shortest path, which is obtained from Step 3.

4.3 Slope-Check Approach

In Sections 4.1 and 4.2, two types of function approximation methods are provided to approximate the inventory cost function. Given a period t and the actual piecewise linear inventory cost function, it can be seen that this inventory cost function is discontinuous at some points, while the approximate inventory cost functions obtained from Algorithms 1-2′ are continuous. Therefore, those approaches may not accurately estimate the actual optimal solution. To increase the accuracy of the algorithms, the slope-check approach is proposed. Here, instead of finding all piecewise lines, as did Chen et al. [22], the breakpoints of those piecewise lines are investigated. In order to achieve this, the inventory costs for real points are calculated by the truncated DP given in equation (4.3), the slope of the lines connecting

two consecutive points (segments) are computed by connecting two consecutive real points at a time, and then the slope of a segment is compared to the slope of the previous and next segment. Since the slope of a segment changes only when there are breakpoints in the interval of two real points, once that segment slope is shown to differ from the previous and following segments, the breakpoints can easily be located into a small interval of two consecutive real points. Knowing the intervals in which the breakpoints are located, DP can be used to calculate the points within those intervals to obtain the real values.

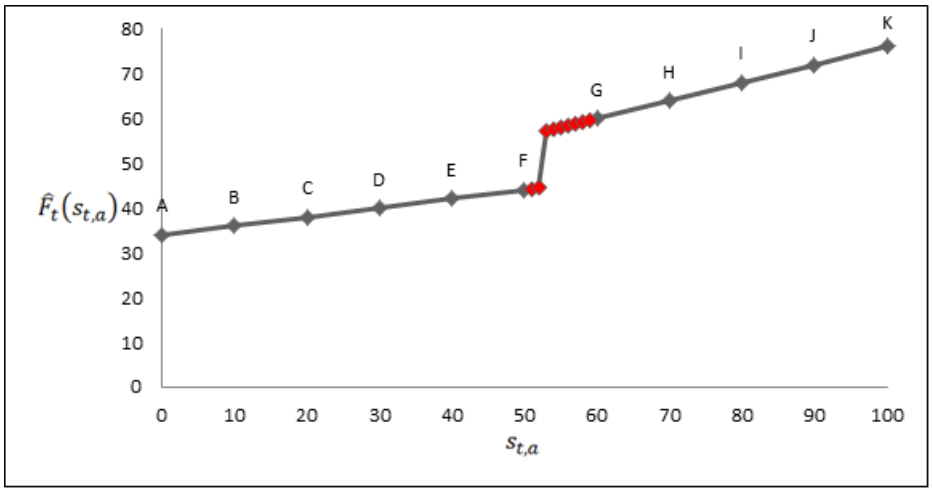

Figure 4.4: Example inventory cost function via slope-check approach.

Figure 4.4 shows an example of possible inventory levels changing from 0 to 100 and the corresponding piecewise lines. This figure also demonstrates how to locate the breakpoints of the real inventory cost function into a small interval. Based on the ADP approach, not all of the 101 points are computed directly from DP, but rather the inventory cost is calculated when inventory is 0, 10, . . . , 100, which are represented by $A, B, ..., K$, respectively. Let $s(AB)$ denote the slope of segment AB. Slope is defined similarly for all other lines. From Figure 4.4, it can be seen that $s(AB) = s(BC) = s(CD) = s(DE) = s(EF), s(EF) \neq s(FG), s(FG) \neq s(GH), s(GH) = s(HI) = s(IJ) = s(JK)$. Therefore, the inventory costs for inventory levels from 0 to 50 (A to F) are calculated based on the

linear equation of the line AF, the inventory costs for inventory levels from 60 to 100 (G to K) are calculated by the linear equation of the line GK, and inventory levels from 50 to 60 (F to G) are calculated by using DP. After the costs for all inventory levels are obtained for a given period t, then the same steps for the next period are performed. This algorithm is denoted as the ADP slope-check algorithm. Let $slope(a)$ represent the slope of the line that connects the two points $(s_{t,a}, \tilde{F}_t(s_{t,a}))$ and $(s_{t,a+1}, \tilde{F}_t(s_{t,a+1}))$, and let $line(a)$ represent the equation of the line that connects the two points $(s_{t,a}, \tilde{F}_t(s_{t,a}))$ and $(s_{t,a+1}, \tilde{F}_t(s_{t,a+1}))$. Then the steps of the ADP slope-check algorithm are given in Algorithm 3.

In the second algorithm of the slope-check approach, instead of calling the DP recursion equation (4.3) to calculate the inventory costs for the whole interval from two consecutive real points where the slope of the line changes, such as F and G as shown in Figure 4.4, a bisection method is used to locate the breakpoints of the real inventory cost function into a smaller interval. The process is described as follows: In the first iteration of the bisection method, when the slope of a connection line is shown to differ from the previous and following lines such as the line FG as shown in Figure 4.4, instead of using the DP recursion equation (4.3) to compute all the inventory costs in that interval where the slope changes, only the midpoint and the corresponding inventory cost are computed. After obtaining the inventory cost for the midpoint, the slope-check is applied to locate the breakpoints into a smaller interval. Considering the example in Figure 4.4, instead of using the DP recursion equation (4.3) to compute all inventory from 50 to 60 (F to G) as in the slope-check algorithm, only the midpoint of F to G is computed, which is shown as L in Figure 4.5. It can be seen that $s(LG) = s(GH)$, implying that the breakpoints are lying in the interval from F to L. Therefore, for the interval from L to G, the inventory costs are computed by the linear equation for line LG, while for the interval from F to L, the second iteration of the bisection method is applied to locate the breakpoints into a smaller interval in which the inventory costs are computed by DP. However, it is possible that for point L, $s(LG) \neq s(GH)$ and $s(EF) \neq s(FL)$. In such cases, the breakpoints cannot be located, either in the interval from

Algorithm 3 ADP Slope Check

Step 1. Assuming that there is no inventory at $t = 0$, then for t=1, compute inventory cost function $F_1(s_1) = \{p_1 * (s_1 + d_1) + f_1 + h_1 s_1\}$, for each $L_1 \leq s_1 \leq U_1$; increment $t = t + 1$;
Step 2. Compute N_t using equation (4.1);
for $a = 1$ to N_t **do**
  Calculate inventory level $s_{t,a}$ using equation (4.2); solve DP recursion equation (4.3) to compute inventory cost $\tilde{F}_t(s_{t,a})$ for real point $s_{t,a}$;
end for
for $a = 1$ to N_t **do**
  Compute $slope(a)$;
end for,
for $a = 2$ to N_t **do**
  If $slope(a) = slope(a-1)$,
  for $s_t = s_{t,a-1}$ to $s_{t,a+1}$ **do**
    Calculate inventory cost value $\hat{F}_t(s_t)$ based on equation $line(a-1)$;
  end for,
  Else If $slope(a) = slope(a+1)$,
  for $s_t = s_{t,a}$ to $s_{t,a+2}$ **do**
    Calculate inventory cost value $\hat{F}_t(s_t)$ based on equation $line(a)$;
  end for,
  Else
  for $s_t = s_{t,a}$ to $s_{t,a+1}$ **do**
    Call DP recursion equation DP recursion equation (4.3) to compute inventory cost $\tilde{F}_t(s_{t,a})$;
  end for,
end for
Increment $t = t + 1$;
Step 3. If $t < T$, then repeat Step 2; **else** call BP to find shortest path;
Step 4. Calculate the actual optimum cost of the shortest path, which is obtained from Step 3.

F to L or the interval from L to G in the first iteration of the bisection process; therefore, the DP recursion equation (4.3) is used to compute the inventory costs for all inventory levels from F to G. This algorithm is denoted as Algorithm 3′. Let $s_{t,b} = 0.5 * (s_{t,a} + s_{t,a+1})$, $s_{t,c} = 0.5 * (s_{t,a} + s_{t,b})$, $s_{t,d} = 0.5 * (s_{t,b} + s_{t,a+1})$. Let $\tilde{F}_t(s_{t,b})$, $\tilde{F}_t(s_{t,c})$, and $\tilde{F}_t(s_{t,d})$ represent the inventory cost for the inventory level $s_{t,b}$, $s_{t,c}$, and $s_{t,d}$, respectively.

To illustrate the method, b_1 is defined to represent the line that connect the two points $(s_{t,b}, \tilde{F}_t(s_{t,b}))$ and $(s_{t,a+1}, \tilde{F}_t(s_{t,a+1}))$, b_2 is defined to represent the line that connect the two points $(s_{t,a}, \tilde{F}_t(s_{t,a}))$ and $(s_{t,b}, \tilde{F}_t(s_{t,b}))$, c_1 is defined to represent the line that connect the two points $(s_{t,c}, \tilde{F}_t(s_{t,c}))$ and $(s_{t,b}, F_t(s_{t,b}))$, c_2 is defined to represent the line that connect the two points $(s_{t,a}, \tilde{F}_t(s_{t,a}))$ and $(s_{t,c}, \tilde{F}_t(s_{t,c}))$, d_1 is defined to represent the line that connect the two points $(s_{t,d}, \tilde{F}_t(s_{t,d}))$ and $(s_{t,a+1}, \tilde{F}_t(s_{t,a+1}))$, and d_2 is defined to represent the line that connect the two points $(s_{t,b}, \tilde{F}_t(s_{t,b}))$ and $(s_{t,d}, \tilde{F}_t(s_{t,d}))$. In addition, let $slope(b_1)$, $slope(b_2)$, $slope(c_1)$, $slope(c_2)$, $slope(d_1)$, and $slope(d_2)$ represent the slope of the line b_1, b_2, c_1, c_2, d_1, and d_2, respectively. The steps of Algorithm 3′ are the same as Algorithm 3, except that line 18 of Algorithm 3 is replaced by Algorithm 4.

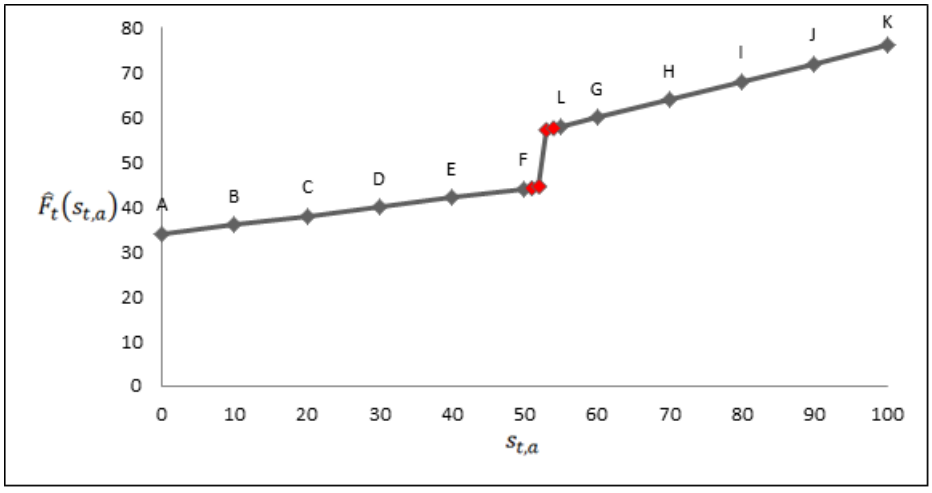

Figure 4.5: Example inventory cost function via slope-check bisection approach.

Algorithm 4 ADP Slope Check Bisection

if $slope(b_1) = slope(a+1)$ **then**
  for $s_t = s_{t,b}$ to $s_{t,a+1}$ **do**
    Calculate inventory cost value $\hat{F}_t(s_t)$ based on equation $line(a+1)$;
  end for,
  if $slope(c_1) = slope(a+1)$ **then**
    for $s_t = s_{t,c}$ to $s_{t,b}$ **do**
      Calculate inventory cost value $\hat{F}_t(s_t)$ based on equation $line(a+1)$;
    end for,
    for $s_t = s_{t,a}$ to $s_{t,c}$ **do**
      Call DP to calculate inventory cost value $\hat{F}_t(s_t)$;
    end for,
  else
    if $slope(c_2) = slope(a)$ **then**
      for $s_t = s_{t,a}$ to $s_{t,c}$ **do**
        Calculate inventory cost value $\hat{F}_t(s_t)$ based on equation $line(a)$;
      end for,
      for $s_t = s_{t,c}$ to $s_{t,b}$ **do**
        Call DP to calculate inventory cost value $\hat{F}_t(s_t)$;
      end for,
    end if
  else
    for $s_t = s_{t,a}$ to $s_{t,b}$ **do**
      Call DP to calculate inventory cost value $\hat{F}_t(s_t)$;
    end for,
  end if
else
  if $slope(b_2) = slope(a)$ **then**
    for $s_t = s_{t,b}$ to $s_{t,a}$ **do**
      Calculate inventory cost value $\hat{F}_t(s_t)$ based on equation $line(a)$;
    end for,
    if $slope(d_2) = slope(a)$ **then**
      for $s_t = s_{t,b}$ to $s_{t,d}$ **do**
        Calculate inventory cost value $\hat{F}_t(s_t)$ based on equation $line(a)$;
      end for,
      for $s_t = s_{t,d}$ to $s_{t,a+1}$ **do**
        Call DP to calculate inventory cost value $\hat{F}_t(s_t)$;
      end for,
    else
      if $slope(d_1) = slope(a+1)$ **then**
        for $s_t = s_{t,d}$ to $s_{t,a+1}$ **do**
          Calculate inventory cost value $\hat{F}_t(s_t)$ based on equation $line(a+1)$;
        end for,
        for $s_t = s_{t,b}$ to $s_{t,d}$ **do**
          Call DP to calculate inventory cost value $\hat{F}_t(s_t)$;
        end for,
      end if
    else
      for $s_t = s_{t,b}$ to $s_{t,a+1}$ **do**
        Call DP to calculate inventory cost value $\hat{F}_t(s_t)$;
      end for,
    end if
  end if
end if

CHAPTER 5

Computational Experiments

This section presents experiments on the computational effectiveness of the ADP algorithms proposed in the previous section in solving randomly generated CLSP instances. All computations are performed on a personal computer running Windows 7 with a 3.4 GHz CPU and 16 GB memory, using CPLEX 12.3.[1]. A time limit of 7,200 CPU seconds is set for all test instances.

5.1 Instance Generation

Test instances are generated based on the scheme employed by Atamturk and Munoz [4] and Hartman et al. [13]. Atamtürk and Munoz [4] noted the following: (a) the tightness of capacities with respect to demand and (b) the ratio between the setup cost and the inventory holding cost, which play major roles in defining the difficulty of a CLSP instance. To observe the effect of different capacities and cost parameters, instances for capacity-to-demand ratios $c \in \{3, 5, 8\}$, setup-to-holding cost ratios $f \in \{1,000, 10,000\}$, and number of stages $T \in \{90, 120, 150\}$ are generated. Five random instances for each combination of these parameters are generated, for a total of 90 instances. Unit production costs p_t, demands d_t, capacities c_t, and setup costs s_t are randomly generated from integer uniform distributions with ranges as follows: $p_t \in [1, 5]$, $d_t \in [1, 600]$, $c_t \in \left[0.70c\bar{d}, 1.10c\bar{d}\right]$, $s_t \in \left[0.90f\bar{h}, 1.10f\bar{h}\right]$, where $\bar{d}$ and $\bar{h}$ are averages for demand and holding costs. The inventory holding cost h_t is fixed at 1 for each period.

5.2 Implementation and Experimental Design

The efficiency of solving the following solution approaches in the computational experiments is as follows:

- **MIP**: Solution of formulation (1.1)–(1.5) with CPLEX 12.3 default setting.

[1]CPLEX is a trademark of ILOG, Inc.

- **DP**: Solution of DP given by recursion equation (3.3).
- **ADPregline**: Solution of Algorithm 1 described in Section 4.1.
- **ADPregpoly**: Solution of Algorithm 1′ described in Section 4.1.
- **ADPpwline**: Solution of Algorithm 2 described in Section 4.2.
- **ADPpwpoly**: Solution of Algorithm 2′ described in Section 4.2.
- **ADPslopecheck**: Solution of Algorithm 3 described in Section 4.3.
- **ADPbisection**: Solution of Algorithm 3′ described in Section 4.3.

5.2.1 Summary of Experimental Results

Tables 5.1 to 5.8 report the results of the computational experiments, where the columns are defined as follows:

- **exp**: Solution approach.
- **percent(%)**: Percentage of total DP nodes solved exactly by ADP method considered, i.e., α in equation (4.1).
- **aprxN**: Number of nodes solved using ADP for all ADP methods. In this column, (•) indicates the number of nodes solved by exact DP for the DP method.
- **bestub**: Best objective function value obtained by method considered.
- **optgap(%)**: Initial optimality gap given best objective function value (mipopt) found by CPLEX, and best solution obtained by method considered (bestub), i.e., gap = 100(bestub−mipopt)/mipopt (percent deviation from optimality).
- **time**: Total CPU seconds required to solve problem by considered method.

Tables 5.1, 5.2, and 5.3 present the results for $T = 90$ with $c = 3$, $c = 5$, and $c = 8$, respectively, where each table entry corresponds to the average performance of an algorithm over five instances. From those results, when the setup ratio f is $1,000$ (low setup cost), the MIP strategy takes the shortest time to give the best solution. On the other hand, when f is increased to $10,000$, the MIP becomes extremely slow compared to the solution times of other algorithms. Therefore, it can be concluded that the setup cost is a key parameter for the computational time for MIP. However, f does not significantly influence the computational time of DP. From the computational results, it can be observed that the key parameter influencing the computational time of DP is the capacity multiplier c. When c is increased from 3 to 8 (i.e., production capacity is increased), the computational time for DP increases more than three times for the instances with $f = 1,000$. As c increases, it can be seen that the effectiveness of the ADP strategies with respect to DP increases.

These three tables show that for those instances where $f = 10,000$, all ADP strategies appear to provide an improvement over MIP and DP strategies, except ADPregpoly strategy, in terms of computational time. In particular, it can be seen that for all $f = 10,000$ instances, the ADPslopecheck and ADPbisection strategies reduce the optimality gap to zero with much less solution time compared to MIP, DP, and all other ADP strategies, when the percent is set to 5 or more. However, the ADPregpoly strategy performed poorly in these instances, especially when $f = 1,000$. This behavior can be explained by the fact that when a polynomial function is fitted to the sampled inventory values in a stage t, the polynomial function does not approximate the inventory cost-value function well in that stage. This is because the fitted polynomial function does not have a non-decreasing property, which has been owned by the actual inventory cost function.

For all ADP algorithms, only ADPslopecheck and ADPbisection strategy provide the optimal solution. Furthermore, for every type of data, the ADPslopecheck and ADPbisection save considerable computational time compared to the solution time by DP. Additionally, as the computational time of MIP and DP increases significantly with the increase of f

and c values, respectively, the computational time of the ADPslopecheck and ADPbisection strategy do not change much with respect to changing values of f and c. ADPslopecheck and ADPbisection solve all instances in less than 10 seconds with 5 and 10 percent. In all instances, it can be seen that the ADPbisection solves the problem a little faster than the ADPslopecheck method. It is also observed that the ADPpwline strategy with 10% also saves a considerable amount of time from DP; however, this strategy does not give the optimal solution. When the overall averages of 30 instances are observed, the ADPbisection strategy is the most efficient strategy, both in terms of solution quality and solution time.

From the results for $T = 90$, since performance of the ADPslopecheck and ADPbisection methods are shown to be most effective among all ADP algorithms, the computational comparison for the instances where $T = 120$ and $T = 150$ is restricted to the ADPslopecheck and ADPbisection methods in order to compare with MIP and DP.

Tables 5.4 and 5.5 present the results for $T = 120$ with $f = 1,000$ and $f = 10,000$, where each table entry corresponds to the average performance of an algorithm over five instances. From the results, it can be seen that it takes more computational time for all methods to solve the instances with 120 compared to the instances with $T = 90$. Again, when the setup ratio f is $1,000$, the MIP strategy takes the shortest time to give the best solution. On the other hand, when f is increased to $10,000$, then the MIP method becomes extremely slow. It is also observed that the computational time for ADPslopecheck and ADPbisection increases as c increases; however, their performance is still better compared to DP in terms of solution time. Therefore, as c increases, the effectiveness of ADPslopecheck and ADPbisection with respect to DP increases.

Tables 5.6 and 5.7 present the results for $T = 150$ with $f = 1,000$ and $f = 10,000$, where each table entry corresponds to the average performance of an algorithm over five instances. As expected, the overall computational time required by any method is larger for instances where $T = 150$ than instances where $T = 120$. It is observed that the ADPbisection method performs the best when the percentage is set to 5, while the same method with 1

Table 5.1: Summary of experiments for $c = 3$ and $T = 90$.

c	f	exp	percent(%)	aprxN	bestub	optgap(%)	time
3	1,000	MIP	-	-	116,634	0.00	0.65
		DP	-	(714,621)	116,634	0.00	18.08
		ADPregline	10	71,462	161,983	38.88	7.56
			30	214,386	161,983	38.88	12.01
			50	357,310	161,983	38.88	16.45
		ADPregoly	10	71,462	793,867	80.65	9.03
			30	214,386	817,018	600.50	15.31
			50	357,310	808,010	592.77	21.23
		ADPpwline	10	71,462	133,454	14.42	2.18
			50	357,310	124,294	6.57	10.16
			90	643,159	118,781	1.84	18.19
		ADPpwpoly	10	71,462	134,149	15.02	2.16
			50	357,310	124,912	7.10	10.14
			90	643,159	118,782	1.84	18.12
		ADPslopecheck	1	328,830	116,923	0.00	9.23
			5	114,393	116,634	0.00	3.41
			10	110,289	116,634	0.00	3.34
			15	133,373	116,634	0.00	3.99
		ADPbisection	1	250,690	116,923	0.00	7.13
			5	88,656	116,634	0.00	2.66
			10	98,543	116,634	0.00	2.94
			15	127,008	116,634	0.00	7.80
3	10,000	MIP	-	-	419,909	0.00	466.09
		DP	-	(781,175)	419,909	0.00	20.41
		ADPregline	10	390,588	646,163	53.88	7.94
			30	78,118	646,163	53.88	13.23
			50	234,353	646,163	53.88	17.94
		ADPregoly	10	390,588	1,199,696	185.70	9.52
			30	78,118	1,197,704	185.23	16.35
			50	390,588	1,205,252	187.03	22.99
		ADPpwline	10	703,058	681,317	62.25	2.37
			50	78,118	543,710	29.48	11.07
			90	390,588	446,449	6.32	19.74
		ADPpwpoly	10	703,058	679,677	61.86	2.36
			50	39,059	556,474	32.52	11.03
			90	78,118	452,976	7.87	19.76
		ADPslopecheck	1	662,537	419,909	0.00	18.60
			5	296,072	419,909	0.00	8.44
			10	215,286	419,909	0.00	6.28
			15	213,767	419,909	0.00	6.20
		ADPbisection	1	622,161	419,909	0.00	17.33
			5	249,501	419,909	0.00	7.21
			10	190,812	419,909	0.00	5.67
			15	200,976	419,909	0.00	8.80

Table 5.2: Summary of experiments for $c = 5$ and $T = 90$.

c	f	exp	percent(%)	aprxN	bestub	optgap(%)	time
5	1,000	MIP	-	-	106,785	0.00	0.57
		DP	-	(980,274)	106,785	0.00	43.52
		ADPregline	10	98,027	158,282	48.22	10.65
			30	294,082	158,282	48.22	20.54
			50	490,137	158,282	48.22	31.09
		ADPregoly	10	98,027	1,026,561	861.33	12.63
			30	294,082	1,033,581	867.91	25.42
			50	490,137	1,060,009	892.66	37.12
		ADPpwline	10	98,027	122,647	14.85	4.98
			50	490,137	115,538	8.20	24.20
			90	882,246	107,996	1.13	43.24
		ADPpwpoly	10	98,027	123,635	15.78	4.94
			50	490,137	116,361	8.97	24.11
			90	882,246	107,996	1.13	43.38
		ADPslopecheck	1	215,608	106,821	0.03	10.62
			5	93,729	106,785	0.00	4.72
			10	120,170	106,785	0.00	6.08
			15	162,014	106,785	0.00	8.12
		ADPbisection	1	130,947	106,821	0.03	6.86
			5	75,205	106,785	0.00	3.99
			10	111,980	106,785	0.00	5.81
			15	157,508	106,785	0.00	11.24
5	10,000	MIP	-	-	315,654	0.00	317.00
		DP	-	(954,005)	315,654	0.00	41.61
		ADPregline	10	95,401	536,156	69.86	10.37
			30	286,202	535,903	69.78	20.44
			50	477,003	535,903	69.78	30.08
		ADPregoly	10	95,401	1,245,770	294.66	12.66
			30	286,202	1,245,770	294.66	24.49
			50	477,003	1,216,090	285.26	36.49
		ADPpwline	10	95,401	747,512	136.81	4.76
			50	477,003	489,212	54.98	22.91
			90	858,605	359,693	13.95	41.12
		ADPpwpoly	10	95,401	754,150	138.92	4.82
			50	477,003	502,304	59.13	22.93
			90	858,605	371,492	17.69	41.12
		ADPslopecheck	1	542,915	316,341	0.22	26.09
			5	185,953	315,654	0.00	8.94
			10	167,373	315,654	0.00	8.16
			15	191,651	315,654	0.00	9.29
		ADPbisection	1	450,328	316,341	0.22	21.46
			5	148,756	315,654	0.00	7.35
			10	149,406	315,654	0.00	7.45
			15	182,417	315,654	0.00	10.99

Table 5.3: Summary of experiments for $c = 8$ and $T = 90$.

c	**f**	**exp**	**percent(%)**	**aprxN**	**bestub**	**optgap(%)**	**time**
8	1,000	MIP	-	-	101,963	0.00	0.48
		DP	-	(1,014,387)	101,963	0.00	64.43
		ADPregline	10	101,439	159,643	56.57	13.11
			30	304,316	159,643	56.57	27.86
			50	507,194	159,643	56.57	43.11
		ADPregoly	10	101,439	1,153,658	1031.45	14.40
			30	304,316	1,190,923	1067.99	29.02
			50	507,194	1,130,174	1008.41	51.27
		ADPpwline	10	101,439	118,229	15.95	7.41
			50	507,194	110,790	8.66	35.78
			90	912,948	102,967	0.98	64.83
		ADPpwpoly	10	101,439	118,971	16.68	7.89
			50	507,194	111,699	9.55	36.58
			90	912,948	102,879	0.90	65.12
		ADPslopecheck	1	142,997	101,963	0.00	10.25
			5	79,255	101,963	0.00	5.69
			10	115,801	101,963	0.00	8.34
			15	161,738	101,963	0.00	11.55
		ADPbisection	1	78,713	101,963	0.00	6.35
			5	65,669	101,963	0.00	4.88
			10	109,697	101,963	0.00	7.98
			15	158,273	101,963	0.00	11.38
8	10,000	MIP	-	-	258,975	0.00	3.28
		DP	-	(956,410)	258,975	0.00	57.34
		ADPregline	10	95,641	513,977	98.47	12.36
			30	286,923	513,977	98.47	25.78
			50	478,205	513,977	98.47	39.05
		ADPregoly	10	95,641	1,522,174	487.77	15.44
			30	286,923	1,522,174	487.77	32.26
			50	478,205	1,345,375	419.50	54.79
		ADPpwline	10	95,641	804,342	210.59	7.81
			50	478,205	449,528	73.58	38.03
			90	860,769	311,693	20.36	67.36
		ADPpwpoly	10	95,641	816,978	215.47	7.98
			50	478,205	462,783	78.70	39.75
			90	860,769	322,630	24.58	69.22
		ADPslopecheck	1	356,980	258,975	0.00	24.29
			5	124,816	258,975	0.00	8.38
			10	135,069	258,975	0.00	9.15
			15	172,583	258,975	0.00	11.01
		ADPbisection	1	268,050	258,975	0.00	18.35
			5	97,764	258,975	0.00	6.68
			10	122,871	258,975	0.00	8.45
			15	163,256	258,975	0.00	11.10

Table 5.4: Summary of experiments for $f = 1,000$ and $T = 120$.

c	**f**	**exp**	**percent(%)**	**aprxN**	**bestub**	**optgap(%)**	**time**
3	1,000	MIP	-	-	152,310	0.00	1.03
		DP	-	(1,335,824)	152,310	0.00	33.22
		ADPslopecheck	1	604,631	152,310	0.00	17.26
			5	205,155	152,310	0.00	5.72
			10	202,843	152,310	0.00	5.75
			15	247,162	152,310	0.00	6.95
		ADPbisection	1	457,260	152,310	0.00	12.98
			5	159,397	152,310	0.00	4.57
			10	182,591	152,310	0.00	5.25
			15	236,158	152,310	0.00	6.72
5	1,000	MIP	-	-	141,278	0.00	0.78
		DP	-	(1,700,354)	141,278	0.00	74.01
		ADPslopecheck	1	377,265	141,278	0.00	18.52
			5	161,919	141,278	0.00	7.80
			10	208,379	141,278	0.00	10.15
			15	280,875	141,278	0.00	13.65
		ADPbisection	1	241,915	141,278	0.00	12.35
			5	130,877	141,278	0.00	6.50
			10	194,919	141,278	0.00	9.65
			15	273,673	141,278	0.00	13.48
8	1,000	MIP	-	-	136,205	0.00	0.57
		DP	-	(1,773,765)	136,205	0.00	113.20
		ADPslopecheck	1	244,046	136,205	0.00	17.77
			5	137,948	136,205	0.00	9.74
			10	202,165	136,205	0.00	14.38
			15	282,773	136,205	0.00	20.10
		ADPbisection	1	140,081	136,205	0.00	11.17
			5	116,028	136,205	0.00	8.47
			10	192,450	136,205	0.00	13.88
			15	277,304	136,205	0.00	19.84

Table 5.5: Summary of experiments for $f = 10,000$ and $T = 120$.

c	**f**	**exp**	**percent(%)**	**aprxN**	**bestub**	**optgap(%)**	**time**
3	10,000	MIP	-	-	567,323	0.00	1511.18
		DP	-	(1,428,064)	567,323	0.00	37.65
		ADPslopecheck	1	1,258,548	568,267	0.17	38.35
			5	549,328	567,323	0.00	16.18
			10	402,916	567,323	0.00	11.90
			15	392,237	567,323	0.00	11.68
		ADPbisection	1	1,182,820	568,267	0.17	35.07
			5	470,742	567,323	0.00	13.90
			10	358,767	567,323	0.00	10.88
			15	369,991	567,323	0.00	11.07
5	10,000	MIP	-	-	406,281	0.00	434.62
		DP	-	(1,774,921)	406,281	0.00	76.78
		ADPslopecheck	1	976,850	406,281	0.00	47.77
			5	334,093	406,281	0.00	16.09
			10	301,539	406,281	0.00	14.75
			15	350,917	406,281	0.00	17.00
		ADPbisection	1	807,283	406,281	0.00	39.18
			5	264,821	406,281	0.00	12.82
			10	269,919	406,281	0.00	13.18
			15	334,364	406,281	0.00	16.40
8	10,000	MIP	-	-	344,090	0.00	73.49
		DP	-	(1,874,613)	344,090	0.00	125.37
		ADPslopecheck	1	659,565	344,093	0.00	49.42
			5	233,733	344,090	0.00	17.26
			10	257,980	344,090	0.00	18.98
			15	328,618	344,090	0.00	24.47
		ADPbisection	1	492,689	344,093	0.00	37.17
			5	185,487	344,090	0.00	13.83
			10	236,656	344,090	0.00	17.62
			15	317,305	344,090	0.00	23.58

Table 5.6: Summary of experiments for $f = 1,000$ and $T = 150$.

c	f	exp	percent(%)	aprxN	bestub	optgap(%)	time
3	1,000	MIP	-	-	193,238	0.00	1.30
		DP	-	(2,233,782)	193,238	0.00	58.33
		ADPslopecheck	1	963,515	193,451	0.11	27.88
			5	328,396	193,238	0.00	9.58
			10	332,147	193,238	0.00	9.77
			15	408,348	193,238	0.00	12.06
		ADPbisection	1	733,154	193,451	0.11	21.70
			5	258,523	193,238	0.00	7.67
			10	300,424	193,238	0.00	9.01
			15	391,848	193,238	0.00	11.72
5	1,000	MIP	-	-	179,091	0.00	0.83
		DP	-	(2,756,440)	179,091	0.00	118.23
		ADPslopecheck	1	619,452	179,666	0.32	30.43
			5	269,233	179,091	0.00	13.23
			10	341,637	179,091	0.00	16.89
			15	457,302	179,091	0.00	22.65
		ADPbisection	1	403,801	179,666	0.32	20.87
			5	220,677	179,091	0.00	11.09
			10	320,151	179,091	0.00	16.10
			15	445,808	179,091	0.00	22.36
8	1,000	MIP	-	-	171,627	0.00	0.65
		DP	-	(2,894,003)	171,627	0.00	180.78
		ADPslopecheck	1	361,155	172,220	0.35	27.64
			5	219,037	171,627	0.00	16.46
			10	326,718	171,627	0.00	24.68
			15	458,984	171,627	0.00	34.90
		ADPbisection	1	204,945	172,441	0.47	17.29
			5	187,074	171,627	0.00	14.32
			10	312,649	171,627	0.00	23.96
			15	451,167	171,627	0.00	34.58

percent provide results with 0.47% optimality gap for instances where $c = 8$. As c decreases, the performance of ADPbisection method with 1 percent improves.

Table 5.8 present the results for overall averages, where each table entry corresponds to the average performance of an algorithm over 30 instances. It can be seen that the solution time for any method considered increases as T increases. It can also be seen that the overall winner is the ADPbisection method with $percentage(\%) = 5$ without zero optimality gap.

Table 5.7: Summary of experiments for $f = 10,000$ and $T = 150$.

c	**f**	**exp**	**percent(%)**	**aprxN**	**bestub**	**optgap(%)**	**time**
3	10,000	MIP	-	-	699,082	0.00	6092.44
		DP	-	(2,149,909)	699,082	0.00	56.64
		ADPslopecheck	1	1,892,184	700,534	0.21	54.52
			5	869,321	699,082	0.00	25.55
			10	642,594	699,082	0.00	18.96
			15	627,024	699,082	0.00	18.45
		ADPbisection	1	1,776,145	702,114	0.43	51.49
			5	747,275	699,082	0.00	21.50
			10	578,573	699,082	0.00	16.72
			15	594,013	699,082	0.00	17.35
5	10,000	MIP	-	-	512,524		5863.12
		DP	-	(2,668,207)	512,524		115.67
		ADPslopecheck	1	1,561,452	512,524	0.00	74.16
			5	551,659	512,524	0.00	26.49
			10	483,445	512,524	0.00	23.48
			15	547,157	512,524	0.00	26.48
		ADPbisection	1	1,327,469	512,524	0.00	63.64
			5	443,376	512,524	0.00	21.33
			10	433,168	512,524	0.00	21.13
			15	521,257	512,524	0.00	25.35
8	10,000	MIP	-	-	421,190		1322.75
		DP	-	(2,773,435)	421,190		176.58
		ADPslopecheck	1	1,014,191	424,303	0.74	71.34
			5	372,276	421,190	0.00	26.24
			10	392,282	421,190	0.00	27.93
			15	493,139	421,190	0.00	35.09
		ADPbisection	1	782,246	424,303	0.74	56.41
			5	296,907	421,190	0.00	21.19
			10	358,868	421,190	0.00	25.86
			15	475,478	421,190	0.00	34.36

Table 5.8: Summary of experiments for overall averages over $T = 90$, $T = 120$, and $T = 150$.

T	**exp**	**percent(%)**	**aprxN**	**bestub**	**optgap(%)**	**time**
90	MIP	-	-	219,987		131.35
	DP	-	(900,145)	219,987		40.90
	ADPslopecheck	1	374,978	220,155	0.04	16.51
		5	149,036	219,987	0.00	6.60
		10	143,998	219,987	0.00	6.89
		15	172,521	219,987	0.00	8.36
	ADPbisection	1	300,148	220,155	0.04	12.91
		5	120,925	219,987	0.00	5.46
		10	130,551	219,987	0.00	6.38
		15	164,906	219,987	0.00	10.22
120	MIP	-	-	291,248		336.94
	DP	-	(1,647,924)	291,248		76.70
	ADPslopecheck	1	686,817	291,406	0.03	31.51
		5	270,363	291,248	0.00	12.13
		10	262,637	291,248	0.00	12.65
		15	313,764	291,248	0.00	15.64
	ADPbisection	1	553,675	291,406	0.03	24.65
		5	221,226	291,248	0.00	10.01
		10	239,217	291,248	0.00	11.74
		15	301,466	291,248	0.00	15.18
150	MIP	-	-	362,792		2213.51
	DP	-	(2,579,296)	362,792		117.71
	ADPslopecheck	1	1,068,658	363,783	0.29	47.66
		5	434,987	362,792	0.00	19.59
		10	419,804	362,792	0.00	20.29
		15	498,659	362,792	0.00	24.94
	ADPbisection	1	871,293	364,083	0.35	38.57
		5	358,972	362,792	0.00	16.18
		10	383,972	362,792	0.00	18.80
		15	479,929	362,792	0.00	24.29

For instances where $T = 150$, the ADPbisection method with 5 percent performed 137 times better than MIP approach and 7 times better than the DP approach.

From the above results, the following points can be concluded:

- As f increases, the computational time for MIP increases.
- As c increases, the computational time for DP and ADP approaches increase, and the computational time for MIP decrease.

- As c increases, the effectiveness of ADPslopecheck and ADPbisection with respect to DP increases, and as f increases, the effectiveness of ADPslopecheck and ADPbisection with respect to MIP increases.

- The ADPbisection is the best approach among all ADP approaches.

- ADPslopecheck and ADPbisection are recommended in particular when the setup cost and capacity is large.

- In terms of solution time and quality, the overall averages show that the best implementation is the ADPbisection method where $percentage(\%) = 5$.

CHAPTER 6

Conclusions and Future Work

6.1 Conclusions

In this thesis are proposed new ADP approaches for the CLSP, which is the core problem of production planning. In Chapter 1, CLSP is introduced and its MIP formulation is presented. In Chapter 2, previous research for the CLSP is summarized based on three categories: MIP, DP, and ADP. In addition, the contributions of the thesis are discussed and the main differences between the proposed ADP approaches and previous research are described. Chapter 3 presents a DP formulation for the CLSP with an example problem, which demonstrates how the DP process is applied to obtain the optimal value for the problem. In Chapter 4, six different algorithms are proposed based on three types of ADP approaches: regression approximation approach, direct-connection approach, and slope-check approach. Chapter 4 describes the steps of the algorithms and provides examples to demonstrate them.

Chapter 5 presents the instance-generation procedure for the experimental study, experimental design, and computational results to show the effectiveness of the proposed ADP approaches. Based on the results, performance of the MIP approach is the best when the setup-to-holding cost ratio is 1,000. However, when this ratio is increased to 10,000, the MIP performs extremely slow, in order to provide an optimal solution compared to other algorithms, and the ADPbisection method is the fastest among all approaches compared. In all instances presented in this thesis, the ADPslopecheck and ADPbisection methods would provide the optimal solution and save a considerable amount of computational time from DP, if the percentage of sampled inventory levels is at least 5 percent. Therefore, ADPslopecheck and ADPbisection can be considered as improved algorithms of DP for the CLSP.

6.2 Future Directions

The lot-sizing problem considered in this thesis is a single-item problem, and results show that ADPslopecheck and ADPbisection algorithms are very efficient for the problem. However, the applicability and performance of those approaches have not been investigated for the multi-item capacitated lot-sizing problems. Therefore, one important direction for future research could be the extension of those approaches to the multi-item capacitated lot-sizing problem. Furthermore, these algorithms could be used for solving the CLSP as a subproblem in decomposition approaches to multi-item problems.

Aside from lot sizing problems for which there are DP approaches, others exist, such as knapsack and parallel equipment replacement problems, where one could employ similar ADP techniques as those presented here.. In particular, since ADPslopecheck and ADPbisection are very effective for approximating value functions having piecewise linear characteristics, these two approaches are expected to work in particular for problems that involve piecewise linear functions. The goal here is to extend these methods as far as possible to maximize the number of problems that can benefit from the approach.

REFERENCES

LIST OF REFERENCES

[1] M. Florian, J. K. Lenstra, and A. H. G. Rinnooy Kan. Deterministic production planning: Algorithms and complexity. *Management Science*, 26(7):669–679, 1980.

[2] Y. Pochet and L. A. Wolsey. Solving multi-item lot-sizing problems using strong cutting planes. *Management Science*, 37(1):53–67, 1991.

[3] L. A. Wolsey. Solving multi-item lot-sizing problems with an MIP solver using classification and reformulation. *Management Science*, 48(12):1587–1602, 2002.

[4] A. Atamtürk and J. C. Muñoz. A study of the lot-sizing polytope. *Mathematical Programming*, 99(3):443–465, 2004.

[5] A. Atamtürk and S. Küçükyavuz. An $O(n^2)$ algorithm for lot sizing with inventory bounds and fixed costs. *Operations Research Letters*, 36(3):297–299, 2008.

[6] Y. Pochet and L. A. Wolsey. *Production Planning by Mixed Integer Programming*. Springer, 2006.

[7] G. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley, 1988.

[8] I. Barany, T. J. Van Roy, and L. A. Wolsey. Strong formulations for multi-item capacitated lot sizing. *Management Science*, 30(10):1255–1261, 1984.

[9] M. Constantino. A cutting plane approach to capacitated lot-sizing with start-up costs. *Mathematical Programming*, 75:353–376, 1996.

[10] G. Belvaux and L. A. Wolsey. BC–PROD: A specialized branch-and-cut system for lot-sizing problems. *Management Science*, 46(5):724–738, 2000.

[11] G. Belvaux and L. A. Wolsey. Modelling practical lot-sizing problems as mixed integer programs. *Management Science*, 47(7):993–1007, 2001.

[12] A. J. Miller, G. L. Nemhauser, and M. W. P. Savelsbergh. A multi-item production planning model with setup times: Algorithms, reformulations, and polyhedral characterization for a special case. *Mathematical Programming*, 95:71–90, 2003.

[13] J. C. Hartman, İ. E. Büyüktahtakın, and J. C. Smith. Dynamic-programming-based inequalities for the capacitated lot-sizing problem. *IIE Transactions*, 42(12):915–930, 2010.

[14] İ. E. Büyüktahtakın, J. C. Smith, and J. C. Hartman. Partial objective function inequalities for the multi-item capacitated lot-sizing problem. *Working Paper*, 2013.

LIST OF REFERENCES (continued)

[15] R. Bellman and R. Kalaba. On adaptive control processes. *Automatic Control*, 4:1–9, 1959.

[16] Hsu H. Schaefer A.J. Alagoz, O. and M.S. Roberts. Markov decision processes: A tool for sequential decision making under uncertainty. *Medical Decision Making*, 30:474–483, 2010.

[17] İ. E. Büyüktahtakın. Dynamic programming via linear programming. In J. J. Cochran, Jr. L. A. Cox, P. Keskinocak, J. P. Kharoufeh, and J. C. Smith, editors, *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Hoboken, NJ, 2011.

[18] H. M. Wagner and T. M. Whitin. Dynamic version of the economic lot size model. *Management Science*, 5:89–96, 1958.

[19] A. Aggarwal and J. K. Park. Improved algorithms for economic lot size problems. *Operations Research*, 41(3):549–571, 1993.

[20] A. Federgruen and M. Tzur. A simple forward algorithm to solve general dynamic lot sizing models with n periods in $O(n \log n)$ or $O(n)$ time. *Management Science*, 37(8):909–925, 1991.

[21] A. Wagelmans, S. van Hoesel, and A. Kolen. Economic lot sizing: An $O(n \log n)$ algorithm that runs in linear time in the Wagner-Whitin case. *Operations Research*, 40:S145–S156, 1992.

[22] H. D. Chen, D. W. Hearn, and C. Y. Lee. A new dynamic programming algorithm for the single item capacitated dynamic lot size model. *Journal of Global Optimization*, 4(3):285–300, 1994.

[23] C. P. M. Van Hoesel and A. P. M. Wagelmans. Fully polynomial approximation schemes for single-item capacitated economic lot-sizing problems. *Mathematics of Operations Research*, 26(2):339–357, 2001.

[24] K. R. Baker, P. Dixon, M. J. Magazine, and E. A. Silver. An algorithm for the dynamic lot-size problem with time-varying production capacity constraints. *Management Science*, 24(16):1710–1720, 1978.

[25] C. S. Chung, J. Flynn, and C. H. M. Lin. An effective algorithm for the capacitated single item lot size problem. *European Journal of Operational Research*, 75(2):427–440, 1994.

LIST OF REFERENCES (continued)

[26] İ. E. Büyüktahtakın. *Mixed Integer Programming Approaches to Lot-Sizing and Asset Replacement Problems*. PhD thesis, Industrial and Systems Engineering, University of Florida, August, 2009.

[27] N. Brahimi, S. Dauzere-Peres, and N. M. Najid. Capacitated multi-item lot-sizing problems with time windows. *Operations Research*, 54(5):951–967, 2006.

[28] M. Diaby, H. C. Bahl, M. H. Karwan, and S. Zionts. Capacitated lot-sizing and scheduling by lagrangean relaxation. *European Journal of Operational Research*, 59:444–458, 1992.

[29] H. Tempelmeier and M. Derstroff. A Lagrangean-based heuristic for dynamic multilevel multiitem constrained lotsizing with setup times. *Management Science*, 42(5):738–757, 1996.

[30] Thomas L. J. Trigeiro, W. W. and J. O. McClain. Capacitated lot sizing with setup times. *Management Science*, 35(3):353–366, 1989.

[31] N. Brahimi, S. Dauzere-Peres, N. M. Najid, and A. Nordli. Single item lot sizing problems. *European Journal of Operational Research*, 168(1):1–16, 2006.

[32] A. Drexl and A. Kimms. Lot sizing and scheduling – survey and extensions. *European Journal of Operational Research*, 99:221–235, 1997.

[33] B. Karimia, S. M. T. Fatemi Ghomia, and J. M. Wilson. The capacitated lot sizing problem: a review of models and algorithms. *Omega*, 31:365–378, 2003.

[34] W. B. Powell. *An Approximate Dynamic Programming Approach to Multi-dimensional Knapsack Problems*. Wiley, 2007.

[35] R. Bellman, R. Kalaba, and B. Kotkin. Polynomial approximation a new computational technique in dynamic programming: Allocation processes. *Mathematics of Computation*, 17:155–161, 1963.

[36] Anton J. Kleywegt, Vijay S. Nori, and Martin W. P. Savelsbergh. Dynamic programming approximations for a stochastic inventory routing problem. *Transportation Science*, 38:42–70, 2004.

[37] A. Toriello, G. Nemhauser, and M. Savelsbergh. Decomposing inventory routing problems with approximate value functions. *Naval Research Logistics*, 57(8):718–727, 2010.

LIST OF REFERENCES (continued)

[38] R. Levi, A. Lodi, and M. Sviridenko. Approximation algorithms for the capacitated multi-item lot-sizing problem via flow-cover inequalities. *Mathematics of Operations Research*, 33(2):416–474, 2008.

[39] D. Adelman and C. Barz. A unifying approximate dynamic programming model for the economic lot scheduling problem. *To appear in Mathematics of Operations Research*, pages 1–29, 2013.

[40] V. M. Muggeo. Estimating regression models with unknown break-points. *Statistics in Medicine, Wiley Online Library*, 22:3055–3071, 2003.

[41] Derek J. Hudson. Least-squares fitting of a polynomial constrained to be either non-negative non-decreasing or convex. *Journal of the Royal Statistical Society. Series B (Methodological)*, 31(1):pp. 113–118, 1969.