# An Algorithm for Single-item Capacitated Economic Lot Sizing with Piecewise Linear Production Costs and General Holding Costs

Dong X. Shaw
*School of Industrial Engineering, Purdue University*
*West Lafayette, Indiana 47907*
*e-mail: shawdx@ecn.purdue.edu*

Albert P.M. Wagelmans
*Econometric Institute, Erasmus University Rotterdam*
*P.O. Box 1738, 3000 DR Rotterdam, The Netherlands*
*e-mail: wagelman@opres.few.eur.nl*

October 18, 1995

**Abstract**

We consider the Capacitated Economic Lot Size problem with piecewise linear production costs and general holding costs, which is an NP-hard problem but solvable in pseudo-polynomial time. A straightforward dynamic programming approach to this problem results in an $O(n^2\bar{c}\bar{d})$ algorithm, where $n$ is the number of periods, and $\bar{d}$ and $\bar{c}$ are the average demand and the average production capacity over the $n$ periods, respectively. However, we present a dynamic programming procedure with complexity $O(n^2\bar{q}\bar{d})$, where $\bar{q}$ is the average number of pieces of the production cost functions. In particular, this means that problems in which the production functions consist of a fixed set-up cost plus a linear variable cost are solved in $O(n^2\bar{d})$ time. Hence, the running time of our algorithm is only *linearly* dependent on the magnitude of the data. This result also holds if extensions such as backlogging and start-up costs are considered. Moreover, computational experiments indicate that the algorithm is capable of solving quite large problem instances within a reasonable amount of time. For example, the average time needed to solve test instances with 96 periods, 8 pieces in every production function and average demand of 100 units, is approximately 40 seconds on a SUN SPARC 5 workstation.

**Key words:** Economic lot sizing, dynamic programming, computational complexity.

## 1 Introduction

In the single-item Capacitated Economic Lot Sizing Problem we consider a production facility which manufactures a single product to satisfy known integer demands over a finite planning horizon of $n$ periods. At each period, the production and holding cost function are given, and the amount of production is subject to an integer capacity limit. Backlogging may be allowed, in which case the associated penalty costs are assumed to be incorporated into the holding cost functions. The problem is that of determining the amounts to be produced in each period such that all demand is satisfied and the total production and holding costs are minimized.

This problem has been the focus of extensive study and continues to receive considerable attention. Many uncapacitated versions of the problem have been shown to be polynomially solvable if some assumptions about the cost structure are made. These problems are typically solved by dynamic programming. For example, Wagner and Whitin (1958) presented an $O(n^2)$ dynamic programming algorithm for the uncapacitated model with fixed-charge production costs and linear holding costs, and without backlogging. Zangwill (1966) presented polynomial algorithms for the extension of Wagner and Whitin's model in which backlogging is allowed and the cost functions are concave. An uncapacitated model with convex cost structures has been studied by Veinott (1964). One can also incorporate start-up costs into the model. This type of cost was introduced by Schrage (1984) – in a slightly different context – to distinguish between the normal set-up costs, which are incurred in each period in which production takes place, and costs that appear only in the first of a consecutive set of periods in which items are produced. The computational complexities of several uncapacitated problems have recently been improved; see Federgruen and Tzur (1991), Aggarwal and Park (1994) and Van Hoesel *et al.* (1994).

The capacitated problem is NP-hard, even for many special cases (see Florian *et al.* (1980) and Bitran and Yanasse (1982)). A notable exception is the problem with concave cost functions and constant capacities, which has been shown to be polynomially solvable in $O(n^4)$ time by Florian and Klein (1971). Recently, Van Hoesel and Wagelmans (1995) showed that this bound can be improved to $O(n^3)$ if backlogging is not allowed and the holding cost functions are linear. We refer to Bitran and Yanasse (1982) and Chung and Lin (1988) for several other special cases which are polynomially solvable. It should also be mentioned that the related problem in which bounds are given on the inventory levels, rather than on the production levels, can be solved in polynomial time using an algorithm presented by Love (1973).

Several articles have appeared in which methods for NP-hard special cases of the capacitated problem are proposed. These methods are typically based on branch-and-bound (for instance, Baker *et al.* (1978) and Erenguc and Aksoy (1990)), dynamic programming (for instance, Kirca (1990) and Chen *et al.* (1994a,1994b)) or a combination of the two (for instance, Chung *et al.* (1994) and Lofti and Yoon (1994)). It should be mentioned that Chen *et al.* (1994b) is the only article which reports computational results for problem instances with more than 24 periods and quite general cost structures.

In the last decade, a lot of research on lot-sizing problems has been done along the direction of determining a (partial) polyhedral description of the set of feasible solutions and applying branch-and-cut methods; see, for example, Pochet (1988) and Leung *et al.* (1989). The main motivation for studying the polyhedral structure of single-item capacitated lot sizing problems is to use the results to develop efficient solution methods for problems, such as multi-item problems, that contain this model as a substructure. However, the branch-and-cut approach has not (yet) resulted in competitive algorithms for the single-item capacitated lot sizing problems themselves. The reason is that generating a single cut could be as time consuming as solving the whole problem, because some separation algorithms require the solution of a linear program with dimensions $O(n^3) \times O(n^3)$.

For the general Capacitated Economic Lot Size Problem a straightforward dynamic programming procedure with complexity $O(n^2\bar{c}\bar{d})$ was presented by Florian *et al.* (1980), where $\bar{c}$ and $\bar{d}$ are the average production capacity and demand over the $n$ periods, respectively. Since this problem is NP-hard, such a pseudo-polynomial complexity is the best one can hope for. However, because the complexity depends quadratically on the magnitude of the data, this approach is not likely to be very practical for solving larger problem instances.

In this paper, we present a new pseudo-polynomial dynamic programming algorithm for problems with a quite general cost structure. The only restriction is that the production cost functions are piecewise linear (not necessarily convex or concave). We allow general holding cost functions, backlogging and start-up costs. As a large class of functions can be approximated by a piecewise linear function, we believe our approach can cover a wide range of problems encountered in practice. Nevertheless, our algorithm has a surprisingly simple structure and runs in $O(n^2\bar{q}\bar{d})$ time, where $\bar{q}$ is the average number of pieces of the production cost functions. Our computational experiments indicate that the algorithm is capable of solving quite large problem instances within a reasonable amount of time. This can be explained by the fact that the running time depends only linearly on the magnitude of the data.

This paper is organized as follows. First, in Section 2, we introduce notation and give a mathematical description of the Capacitated Economic Lot Size problem. In Section 3 we present the basic dynamic programming procedure for the problem without backlogging and with production cost functions consisting of a fixed set-up cost plus one linear part. The extensions of the basic algorithm to the cases with backlogging, start-up costs and general piecewise linear production costs are discussed in Section 4. In Section 5 we present computational test results, and Section 6 contains the conclusion.

## 2 Problem Description

The general Capacitated Economic Lot Size Problem can be formulated as

$$
(CELS) \qquad
\begin{array}{lllr}
\min & \sum_{t=1}^{n}(\bar{p}_t(x_t, x_{t-1}) + h_t(s_t)) & & (1) \\
\text{s.t.} & s_t = s_{t-1} + x_t - d_t & t = 1, 2, \ldots, n & (2) \\
 & x_t \leq c_t & t = 1, 2, \ldots, n & (3) \\
 & x_0 = 0 & & (4) \\
 & s_o = s_n = 0 & & (5) \\
 & x_t \geq 0 & t = 1, 2, \ldots, n & (6) \\
 & s_t \text{ free variable} & t = 1, 2, \ldots, n & (7)
\end{array}
$$

where $\bar{p}_t(\cdot, \cdot)$ and $h_t(\cdot)$ are the production and holding cost functions of period $t$, respectively, and $c_t$ and $d_t$ are, respectively, the production capacity and demand in period $t$. The variables $x_t$ and $s_t$ denote the production level and ending inventory in period $t$. The fact that the production costs in period $t$ are not only a function of $x_t$, but of $x_{t-1}$ as well, reflects the possibility of start-up costs. Clearly, a feasible production schedule exists if and only if $\sum_{t=1}^{n} c_t \geq \sum_{t=1}^{n} d_t$. By imposing the additional constraints $s_t \geq 0$, $t = 1, \ldots, n$, we obtain a model without backlogging. Then the necessary and sufficient condition for the existence of a feasible production schedule is $\sum_{t=1}^{k} c_t \geq \sum_{t=1}^{k} d_t$ for all $1 \leq k \leq n$. Also, without loss of generality, we can assume in that case $c_k \leq \sum_{t=k}^{n} d_t \equiv D_k$, where $D_k$ denotes the tail demand of period $k$.

An interesting special case is the model without backlogging and start-up costs, and with production cost functions consisting of a fixed set-up cost and a linear part, i.e.,

$$
\bar{p}_k(x_k, x_{k-1}) = \begin{cases} 0 & \text{if } x_k = 0, \\ f_k + p_k x_k & \text{if } x_k > 0. \end{cases}
$$

This special case can be formulated as

$$
(CELS_1) \qquad
\begin{aligned}
\min \quad & \sum_{t=1}^{n}(p_t x_t + f_t y_t + h_t(s_t)) && && (8)\\
\text{s.t.} \quad & s_t = s_{t-1} + x_t - d_t && t = 1, 2, \ldots, n && (9)\\
& x_t \le c_t\, y_t && t = 1, 2, \ldots, n && (10)\\
& s_o = s_n = 0 && && (11)\\
& x_t \ge 0,\ s_t \ge 0,\ y_t \in \{0, 1\} && t = 1, 2, \ldots, n && (12)
\end{aligned}
$$

We show in the next section that this problem, which will serve as the basic model, can be solved in $O(nD_1) = O(n^2\bar{d})$ time, where $\bar{d} = \sum_{t=1}^{n} d_t/n$.

## 3 The Basic Dynamic Programming Procedure

In this section we present our dynamic programming algorithm for the basic model, i.e., the problem without backlogging and with production cost functions each consisting of a fixed set-up cost plus a linear part. We will first derive recursion formulas and then show how they can be evaluated efficiently.

### 3.1 The Recursion Formulas

For any period $k$ and inventory level $s \in \{0, 1, 2, \ldots, D_k\}$, we define $F_k(s)$ as the minimal cost incurred in periods $k$ to $n$, when the starting inventory in period $k$ is equal to $s$. Hence,

$$
\begin{aligned}
F_k(s) = \quad \min \quad & \sum_{t=k}^{n}(p_t x_t + f_t y_t + h_t(s_t)) && && (13)\\
\text{s.t.} \quad & s_t = s_{t-1} + x_t - d_t && t = k, k+1, \ldots, n && (14)\\
& x_t \le c_t\, y_t && t = k, k+1, \ldots, n && (15)\\
& s_{k-1} = s,\ \ s_n = 0 && && (16)\\
& x_t \ge 0,\ s_t \ge 0,\ y_t \in \{0, 1\} && t = k, k+1, \ldots, n, && (17)
\end{aligned}
$$

By definition, $F_k(s) = +\infty$ if the above program is infeasible. We also define $F_k(s) = +\infty$ for $s < 0$. Clearly, we have the following backward recursive equation which determines the optimal production level in period $k$:

$$
\begin{aligned}
F_k(s) = \quad \min\{ \quad & h_k(s - d_k) + F_{k+1}(s - d_k),\\
& \min_{1 \le x_k \le c_k} \{f_k + p_k x_k + h_k(s - d_k + x_k) + F_{k+1}(s - d_k + x_k)\} \ \ \} \qquad (18)
\end{aligned}
$$

Define

$$g_k(\tau) = p_k\tau + h_k(\tau) + F_{k+1}(\tau), \tag{19}$$

for $\tau = 0, 1, 2, \ldots, D_{k+1}$, and

$$G_k(s') = \min\{g_k(\tau) \mid (s'+1)^+ \leq \tau \leq \min((s'+c_k)^+, D_{k+1})\} \tag{20}$$

for $s' = -d_k,\ -d_k+1, \ldots, D_{k+1}-1$. Then, for $0 \leq s \leq D_k$,

$$F_k(s) = \min\{h_k(s-d_k) + F_{k+1}(s-d_k),\ f_k - p_k(s-d_k) + G_k(s-d_k)\} \tag{21}$$

More specifically, this means

$$F_k(s) = \begin{cases} f_k - p_k(s-d_k) + G_k(s-d_k), & s = 0, 1, 2, \ldots, d_k - 1 \\ \min\{h_k(s-d_k) + F_{k+1}(s-d_k), f_k - p_k(s-d_k) + G_k(s-d_k)\}, & \\ & s = d_k,\ d_k+1, \ldots, D_k - 1 \\ h_k(D_{k+1}) + F_{k+1}(D_{k+1}), & s = D_k \end{cases}$$

Clearly, for given $k$ and $s$, $F_k(s)$ can be computed from $F_{k+1}(\cdot)$ and $G_k(\cdot)$ in $O(1)$ time. For any $s' \in \{\ -d_k,\ -d_k+1, \ldots, D_{k+1}-1\ \}$, $G_k(s')$ is the minimum of at most $c_k$ values, namely the minimum of $g_k(\tau)$ with $\tau$ ranging over the interval $[(s'+1)^+,\ \min\{(s'+c_k)^+, D_{k+1}\}]$. Hence, a straightforward way of determining $\{\ G_k(s') \mid s' = -d_k,\ -d_k+1, \ldots, D_{k+1}-1\ \}$ requires $O(D_kc_k)$ time. However, in the next subsection a more efficient method will be presented.

## 3.2 The Central Issue

To compute the function $G_k(\cdot)$ efficiently, we are going to exploit the fact that it requires the determination of the minima in a series of "shifting" subsequences of a larger sequence. To be more precise, the central issue in our dynamic programming approach is that we have to solve problems which resemble the following one.

*(CI)* Given positive integers $c$ and $D$ ($c \leq D$), and values $g(\tau)$, $\tau = 0, 1, \ldots, D$, determine for every $s = 0, \ldots, D-c$ the value $\min\{g(\tau) \mid s+1 \leq \tau \leq s+c\}$.

To solve this problem, we propose to use a simple data structure. Consider for any $s \in \{0, \ldots, D-c\}$ the values $g(s+1), \ldots, g(s+c)$. Let $\{t_1, t_2, \ldots, t_r\}$ be the maximal subset of $\{s+1, \ldots, s+c\}$ which has the following properties:

1. $t_r = s + c$,

2. $t_i$ is the largest value in $\{s+1, \ldots, t_{i+1} - 1\}$ such that $g(t_i) < g(t_{i+1})$, $i = 1, \ldots, r-1$.

Note that $g(t_1) = \min\{g(\tau) \mid s+1 \leq \tau \leq s+c\}$. We can keep track of the set $\{t_1, t_2, \ldots, t_r\}$ by storing its elements in increasing order in a Queue/Stack, i.e., a list with the property that elements at the top can only be deleted, while at the bottom elements can be deleted and added (see Aho *et al.* (1983)). This data structure can easily be implemented such that each deletion and each addition requires constant time (see, for instance, the next subsection).

The key result of our paper is the following.

**Theorem 1** *Problem (CI) can be solved in $O(D)$ time.*

**Proof.** We will show that the complexity result holds when the proposed data structure is used. First of all, note that it takes $O(c) = O(D)$ time to determine the elements of the Queue/Stack for $s = 0$.

Now let $\{t_1, t_2, \ldots, t_r\}$ be the elements of the Queue/Stack for any $s \in \{0, \ldots, D-c-1\}$. We will analyze how this set should be updated when $s$ is increased by 1, i.e., $s+1$ is deleted from the interval of consideration and $s+1+c$ is added to it. If $s+1 < t_1$, then its deletion does not affect the Queue/Stack. If $s+1 = t_1$, then it is deleted from the Queue/Stack and $t_2$ becomes the top element. Note that this update takes constant time.

Subsequently, all elements $t_p$ for which $g(t_p) \geq g(s+1+c)$ should be deleted from the bottom of the Queue/Stack. Clearly, the complexity of this part of the updating process is proportional to the number of deletions, which we denote by $m_s$.

Hence, the total complexity of solving *(CI)* using the proposed data structure is $O(D) + O(\sum_{s=0}^{D-c-1} m_s)$. The crucial observation to bound the second term is that while increasing $s$ from 0 to $D-c$, every element of $\{1, \ldots, D\}$ will be deleted from the Queue/Stack at most once. In other words, each of these $D$ elements contributes at most 1 to the summation $\sum_{s=0}^{D-c-1} m_s$. Therefore, $\sum_{s=0}^{D-c-1} m_s \leq D$ and the desired result follows.

$\square$

## 3.3 The DP Algorithm

One way of implementing the proposed data structure for the computation of $G_k(\cdot)$, is by using an array $t(\cdot)$ of length $D_k$ to represent Queue/Stack $\{t_1, t_2, \ldots, t_r\}$. Let *First* and *Last* indicate the position in $t(\cdot)$ of the first and last element of the Queue/Stack, respectively. These variables are initialized as

```
$First = 1;$
$Last = 0;$
```

and we define the following procedures and function.

```
Procedure **Delete** $(g_k(j))$
**if** $j = t(First)$ **then**
   $First = First + 1;$
**endif**
**return**
```

```
Procedure **Add** $(g_k(j))$
**while** ( $g_k(Last) >= g_k(j)$ ) **and** ( $Last >= First$)
   $Last = Last - 1;$
**end-while**
$Last = Last + 1;$
$t(Last) = j;$
**return**
```

```
Function **Min_value**
Min_value=$g(t(First))$;
**return**
```

The following algorithm determines $G_k(s')$ for $s' = -d_k,\ -d_k + 1, \ldots + D_{k+1}$.

```
Algorithm **Compute** $G_k(\cdot)$
**begin**
   **for** $s' = -d_k$ up to $-c_k$
      $G_k(s') = g_k(0);$
   **Add** $(g_k(0));$
   **for** $s' = -c_k + 1$ up to $-1$
   **begin**
```

```
        **Add** $(g_k(s'+c_k))$;
        $G_k(s')$ =**Min_value**;
    **end**
    **for** $s'=0$ up to $D_{k+1}-c_k$
    **begin**
        **Delete** $(g_k(s'))$;
        **Add** $(g_k(s'+c_k))$;
        $G_k(s')$ =**Min_value**;
    **end**
    **for** $s'=D_{k+1}-c_k+1$ up to $D_{k+1}-1$
    **begin**
        **Delete** $(g_k(s'))$;
        $G_k(s')$ =**Min_value**;
    **end**
**end**
```

Clearly, each **Delete**( ) requires $O(1)$ time, and each **Add**( ) requires an amount of time which is proportional to the number of deletions from the rear of the Queue/Stack. As the total number of deletions is bounded by the total number of different elements, it should be clear that Algorithm **Compute** $G_k(\cdot)$ determines $\{\ G_k(s')\ |\ s'=-d_k,-d_k+1,\ldots,D_{k+1}-1\ \}$ from $\{\ g_k(\tau)\ |\ \tau=0,1,2,\ldots,D_{k+1}\ \}$ in $O(D_k)$ time. It is now easy to see that for given $k$, $1\le k<n$, $\{F_k(s)|s=0,1,2,\ldots,D_k\}$ can be computed in $O(D_k)$ time from $\{F_{k+1}(s)|s=0,1,2,\ldots,D_k\}$. The complete algorithm can be described as follows.

```
**Algorithm.** Dynamic Programming
**begin**
    (Initialization)
    **for** $s=0$ up to $d_n-1$
        $F(s)=f_n+p_n(d_n-s)$;
    $F(d_n)=0$;
    (Main loop)
    **for** $k=n-1$ down to 2
    **begin**
```

```
        **Compute** $G_k(\cdot)$;
        $F(D_k) = h_k(D_{k+1}) + F(D_{k+1})$;
        **for** $s = D_k - 1$ down to $d_k$
            $F(s) = \min\{F(s - d_k),\ f_k - p_k(s - d_k) + G_k(s - d_k)\}$;
        **for** $d_k - 1$ down to $s = 0$
            $F(s) = f_k - p_k(s - d_k) + G_k(s - d_k)$;
    **end**
    $G_1(-d_1) = \min\{g_1(\tau) \mid 0 \le \tau \le c_1 - d_1\}$;
    $F(0) = f_1 + p_1 d_1 + G_1(-d_1)$;
**end**
```

It is easy to see that the space complexity of this dynamic programming algorithm to compute $F(\cdot)$ is $O(D_1)$. However, in order to determine the optimal production schedule, we need to perform a backtracking. Hence, the space complexity is $O(nD_1)$ in that case.

The main result of this section is summarized as follows.

**Theorem 2** *The basic Capacitated Economic Lot Size Problem can be solved in $O(nD_1) = O(n^2\bar{d})$ time.*

## 4 Extensions

In this section we will discuss how the dynamic programming algorithm presented in Section 3 can be adapted to solve extensions of the basic model.

### 4.1 Backlogging

If backlogging is allowed in the model, then $h_k(s)$, $1 \le k \le n$, is defined to be equal to the backlogging cost if $s < 0$. Furthermore, $F_k(s)$ has the same meaning as before, but this time it is defined for the range $s = -\tilde{D}_k, -\tilde{D}_k + 1, \ldots, -1, 0, 1, \ldots, D_k$, where $\tilde{D}_k \equiv \sum_{t=1}^{k} d_t$. Similarly, $g_k(\tau)$ is defined as in (19) for $\tau = -\tilde{D}_{k+1}, -\tilde{D}_{k+1} + 1, \ldots, -1, 0, 1, 2, \ldots, D_{k+1}$. Define

$$\bar{G}_k(s') = \min\{g_k(\tau) \mid s' + 1 \le \tau \le \min\{s' + c_k, D_{k+1}\}\} \tag{22}$$

for $s' = -\tilde{D}_{k+1}, -\tilde{D}_{k+1} + 1, \ldots, -1, 0, 1, 2, \ldots, D_{k+1} - 1$. Then the following holds.

$$F_k(s) = \begin{cases} \min\{h_k(s-d_k) + F_{k+1}(s-d_k),\ \ f_k - p_k(s-d_k) + \bar{G}_k(s-d_k)\}, & s \neq D_k \\ \\ h_k(D_{k+1}) + F_{k+1}(D_{k+1}), \quad s = D_k \end{cases}$$

Using the same data structure as in Section 3, $\bar{G}_k(\cdot)$ can be computed in $O(D_k)$ time from $F_{k+1}(\cdot)$. Hence, we have the following result.

**Theorem 3** *The Capacitated Economic Lot Size Problem with production cost functions each consisting of a fixed set-up cost and a linear part, can be solved in $O(n^2\bar{d})$ time, even if backlogging is allowed.*

## 4.2 Start-up Costs

We now consider the extension of the basic model in which start-up costs are included. Let $r_k$, $1 \leq k \leq n$, denote the start-up cost in period $k$. For any period $k$ and inventory level $s \in \{0, 1, 2, \dots, D_k\}$, we define two dynamic programming variables. Both of them refer to minimal costs incurred in periods $k$ to $n$ when the starting inventory in period $k$ is equal to $s$, but subject to some conditions:

| | |
|---|---|
| $F'_k(s)$ : | the minimal cost under the condition that there is no set-up in period $k$ |
| $F''_k(s)$ : | the minimal cost under the condition that a set-up occurs in period $k$, and not including the possible start-up cost in period $k$ |

Furthermore, we define $F_k(s)$ to be the minimum of these two variables. Then, the following triple recursion holds.

$$\begin{aligned} F'_k(s) = &\quad \min\{\ h_k(s-d_k) + F'_{k+1}(s-d_k),\ h_k(s-d_k) + r_{k+1} + F''_{k+1}(s-d_k)\ \} \\ F''_k(s) = &\quad \min_{0 \leq x_k \leq c_k}\{\ f_k + p_k x_k + h_k(s-d_k+x_k) + F_{k+1}(s-d_k+x_k)\}\ \ \} \\ F_k(s) = &\quad \min\{\ F'_k(s),\ F''_k(s)\ \} \end{aligned}$$

Note that we have taken into account the possibility of having a set-up without actual production. In general, this may occur in an optimal solution.

The only non-trivial part of the recursion is the evaluation of $F''_k(s)$, which resembles (18). Therefore, we immediately obtain the following complexity result, which also holds when backlogging is allowed.

**Theorem 4** *The Capacitated Economic Lot Size Problem with production cost functions each consisting of a start-up cost, a set-up cost and a linear part, can be solved in $O(n^2\bar{d})$ time.*

## 4.3 Piecewise Linear Production Cost Functions

Now suppose that the interval $[0, c_k] = I_k^1 \cup I_k^2 \cup \ldots \cup I_k^{q_k}$, where $I_k^i$ is a subinterval and $I_k^i \cap I_k^j = \emptyset$. Let $p_k(x) = f_k^i + p_k^i x$, for $x \in I_k^i$, $i = 1, 2, \ldots, q_k$. Then the recursive equation for $F_k(s)$ becomes

$$\begin{aligned} F_k(s) &= \min_{0 \le x \le c_k} \{\bar{p}_k(x) + h_k(s - d_k + x) + F_{k+1}(s - d_k + x)\} && (23) \\ &= \min_{1 \le i \le q_k} \min_{x \in I_k^i} \{f_k^i - p_k^i x + h_k(s - d_k + x) + F_{k+1}(s - d_k + x)\} && (24) \\ &= \min_{1 \le i \le q_k} \{f_k^i - p_k^i(s - d_k) + G_k^i(s - d_k)\} && (25) \end{aligned}$$

where

$$\begin{aligned} g_k^i(\tau) &= p_k^i \tau + h_k(\tau) + F_{k+1}(\tau) && (26) \\ G_k^i(s') &= \min\{g_k^i(s' + x) |\ 0 < s' + x \le D_{k+1},\ x \in I_k^i\ \} && (27) \end{aligned}$$

It is now easy to see that $G_k^i(\cdot)$ can be computed in $O(D_k)$ time from $F_{k+1}(\cdot)$, for fixed $k$ and $i$. Hence, $F_k(\cdot)$ can be computed in $O(q_k D_k) = O(n q_k \bar{d})$ time, for fixed $k$.

If $\bar{q} = \sum_{k=1}^n q_k / n$, i.e., the average number of pieces of the production cost functions, then we have just shown the following.

**Theorem 5** *The Capacitated Economic Lot Size Problem with piecewise linear production cost functions can be solved in $O(n^2 \bar{q} \bar{d})$ time.*

It is not difficult to verify that this result also holds if we allow backlogging and include start-up costs.

Finally, we note that the dynamic programming algorithm can trivially be adapted for bounds on the inventory levels. Although incorporating such bounds does not affect the complexity of the algorithm, it will actually lower the practical running time.

## 5 Computational Results

We have implemented our algorithm in a C code, which runs on a SUN SPARC 5 workstation. Although our dynamic programming procedure is only a pseudo-polynomial time algorithm, our computational experiments indicate that it is capable of solving quite large problem instances within a reasonable amount of time. This can be explained by the fact that the dependence on the magnitude of the data is only linear.

Our experiments are very similar to those described in Chen *et al.* (1994b), who modified the test problem design of Baker *et al.* (1978) to incorporate piecewise linear production costs. We have created problem instances with $n \in \{12, 24, 48, 96, 192\}$. Demand in period $t$ is randomly generated according to the formula

$$d_t \ = \ \lfloor \ \mu \ + \ \sigma \, z_t \ + a \ \sin( \tfrac{2\pi t}{b} \ + \ \tfrac{\pi}{2} \ ) \ + \ 0.5 \ \rfloor$$

where $\mu$ is approximately the mean demand, $\sigma$ is approximately the standard error of demand, $a$ is the amplitude of the seasonality component, $b$ denotes the number of periods in one seasonal cycle, and the $z_t$ are i.i.d. standard normal random variables. In our experiments we consider $\mu = 20, 100$ and 200. For each of these, problem instances were generated using the following four combinations of parameters: (1) $\sigma = 0.335\,\mu$, $a = 0$, (2) $\sigma = 1.185\,\mu$, $a = 0$, (3) $\sigma = 0.335\,\mu$, $a = 0.625\,\mu$, $b = n$ and (4) $\sigma = 0.335\,\mu$, $a = 0.625\,\mu$, $b = 12$.

We have taken $\bar{q} \in \{1, 2, 4, 8, 16\}$. Actually, in all our tests the number of pieces of the production cost function are equal for every period. The slope of every piece is uniformly distributed in $[2000/\mu, \ 6000/\mu]$.

The capacity of each period and each piece is generated by drawing a number from the uniform distribution on $[0.5\, c/\bar{q}\ , \ 1.5\, c/\bar{q}]$, where $c \in \{2\mu, 4\mu, 6\mu, 8\mu\}$, and then rounding it to the nearest integer. During the problem generation a simple check ensures that only feasible test instances are created.

The set-up cost for each period is uniformly distributed in $[0.5\ f/\bar{q}, \ 1.5\ f/\bar{q}]$, where $f \in \{400, 1600, 3600, 6400\}$. There are no start-up costs. The holding cost function of each period is assumed to be linear, with a slope which is uniformly distributed in $[100/\mu, \ 300/\mu]$. Backlogging is not allowed.

For a given value of $\mu$, we can create in this way a total of 64 problem instances with different characteristics. For $\mu = 200$, our problem generation is almost identical to the way Chen *et al.* (1994b) created their test problems. The only difference is that we require demands and capacities

| pattern | average | maximum |
|---|---|---|
| 1 | 18.64 | 20.63 |
| 2 | 20.61 | 24.45 |
| 3 | 19.02 | 20.70 |
| 4 | 18.98 | 21.13 |

$n = 48$, $\bar{q} = 8$, $\mu = 200$

Table 1: CPU times (in seconds) for different demand patterns

to be integer, whereas Chen *et al.* allow demands and capacities to be real numbers.

Because the complexity of our algorithm depends on the average demand, we have considered different values of $\mu$. By making the unit holding and production costs dependent on $\mu$, changing this parameter can be viewed as rescaling the problem instance.

From the analyses in the preceding sections, we see that the running time of our algorithm is not only bounded by a constant times $n^2\bar{q}\bar{d}$, but that it is actually almost proportional to this expression. Therefore, we expect that the running time of our algorithm will be quite robust with respect to changes of data other than the values of $n$, $\bar{q}$ and $\mu$. Our computational experiments confirm this. For example, Tables 1 and 2 present results for 48 periods, 8 pieces in the production cost functions, and mean demand equal to 200. Tabel 1 distinguishes between the four demand patterns and the averages and maxima are taken over 16 problem instances with different capacity and set-up cost characteristics, whereas in Table 2 the averages and maxima are taken over the demand patterns and the distinction is made with respect to the other characteristics. We see that the results are hardly affected by changes in the problem characteristics. For other values of $n$, $\bar{q}$ and $\bar{d}$, similar results were found.

On the other hand, when increasing the number of periods while keeping all other characteristics the same, the average time grows more or less quadratically, as expected. This is illustrated in Table 3. The averages and maxima are taken over all 64 combinations of demand patterns, set-up costs and capacities. Similarly, changing only $\mu$ or $\bar{q}$ results in almost proportional changes of the CPU times. Typical results are shown in Tables 4 and 5, respectively. Again, all 64 combinations are considered.

We refrain from giving computational results for all problem instances in our test set, because – as should be clear from our observations – it is easy to give a good estimate of the average and maximum running times of our algorithm for problem instances which are not listed in the tables.

| $f$ | $c$ | average | maximum |
|---|---|---|---|
| 400 | 400 | 18.03 | 20.62 |
| 400 | 800 | 19.65 | 21.83 |
| 400 | 1200 | 19.28 | 20.60 |
| 400 | 1600 | 18.93 | 21.13 |
| 1600 | 400 | 17.85 | 19.43 |
| 1600 | 800 | 19.87 | 22.10 |
| 1600 | 1200 | 19.72 | 21.85 |
| 1600 | 1600 | 20.68 | 23.52 |
| 3600 | 400 | 19.95 | 22.47 |
| 3600 | 800 | 19.43 | 23.60 |
| 3600 | 1200 | 17.84 | 18.37 |
| 3600 | 1600 | 19.03 | 19.72 |
| 6400 | 400 | 18.79 | 19.20 |
| 6400 | 800 | 20.36 | 24.45 |
| 6400 | 1200 | 19.65 | 22.37 |
| 6400 | 1600 | 20.09 | 22.12 |

$n = 48$, $\bar{q} = 8$, $\mu = 200$

Table 2: CPU times (in seconds) for different set-up cost and capacity characteristics

| $n$ | average | maximum |
|---|---|---|
| 12 | 0.06 | 0.08 |
| 24 | 0.23 | 0.27 |
| 48 | 0.97 | 1.33 |
| 96 | 3.92 | 5.33 |
| 192 | 15.83 | 19.08 |

$\bar{q} = 4$, $\mu = 20$

Table 3: CPU times (in seconds) for different numbers of periods

| $\mu$ | average | maximum |
|---|---|---|
| 20 | 7.68 | 10.30 |
| 100 | 40.00 | 52.58 |
| 200 | 79.89 | 105.23 |

$n = 96$, $\bar{q} = 8$

Table 4: CPU times (in seconds) for different values of mean demand

| $\bar{q}$ | average | maximum |
|---|---|---|
| 1 | 1.06 | 1.35 |
| 2 | 2.01 | 2.78 |
| 4 | 3.92 | 5.33 |
| 8 | 7.68 | 10.30 |
| 16 | 15.36 | 19.83 |

$n = 96$, $\mu = 20$

Table 5: CPU times (in seconds) for different numbers of pieces

For instance, based on Table 5, we expect the average (maximum) CPU time for $n = 96$, $\bar{q} = 4$ and $\mu = 200$ to be around 40 (55) seconds. The actual result is 40.23 (54.42) seconds.

We think that these results indicate that our dynamic programming algorithm is a computationally feasible approach. We have compared our algorithm with the branch-and-cut approach developed by Leung *et al.* (1989), and the variable-redefinition approach proposed by Eppen and Martin (1987). Both methods can be extended to multi-item problems, while our algorithm is a special purpose algorithm for single-item problems. The three 12-period problems reported in Leung *et al.* require 14 to 18 seconds on a Prime 850, and another six 12-period problems used by Eppen and Martin require 12.35 to 123.02 seconds on a VAX 11/750. Our algorithm can solve all these problems within 0.35 seconds on a SUN SPARC 5 workstation.

Now, it is also interesting to see how the running time of our algorithm compares to the running time of the effective and elegant algorithm presented in a recent paper by Chen *et al.* (1994b). Although the latter algorithm is designed for a more restricted class of problems, we expected that our approach could be competitive for some problem instances with high values of $n$ and $\bar{q}$. The reason is that Chen *et al.* reported CPU times which increase at a rate that is approximately cubic in each of these parameters, whereas our CPU times increase at a quadratic and linear rate, respectively.

A FORTRAN code of Chen *et al.*'s algorithm was made available to us by the authors themselves. After adapting the code to make it run on a SUN SPARC 5 workstation, we used it to solve several of our test problems. The results of the computational comparison indicate that our algorithm is worth considering as an alternative to Chen *et al.*'s for problem instances with high values of $n$ and $\bar{q}$, tight capacity restrictions and average demand which is not too high (say, two digits). This is illustrated in Table 6, which shows CPU times of both algorithms for problem instances with 96 periods, 8 pieces in the production cost functions, and mean demand equal to 20. For

|  |  | our method |  | Chen *et al.* |  |
|---|---|---|---|---|---|
| $f$ | $c$ | average | (maximum) | average | (maximum) |
| 400 | 40 | 7.79 | (8.58) | 5.48 | (6.22) |
| 400 | 80 | 7.83 | (8.81) | 3.67 | (4.61) |
| 400 | 120 | 7.47 | (7.68) | 2.40 | (3.21) |
| 400 | 160 | 7.78 | (8.06) | 1.77 | (2.24) |
| 1600 | 40 | 7.62 | (8.15) | 11.04 | (14.46) |
| 1600 | 80 | 7.31 | (7.68) | 6.46 | (7.83) |
| 1600 | 120 | 7.89 | (9.15) | 3.63 | (4.68) |
| 1600 | 160 | 7.91 | (9.08) | 2.58 | (3.39) |
| 3600 | 40 | 7.78 | (8.10) | 15.81 | (27.87) |
| 3600 | 80 | 8.27 | (10.30) | 11.94 | (15.62) |
| 3600 | 120 | 7.67 | (8.65) | 6.08 | (8.39) |
| 3600 | 160 | 7.98 | (9.38) | 4.18 | (5.57) |
| 6400 | 40 | 7.70 | (8.03) | 13.54 | (19.57) |
| 6400 | 80 | 7.77 | (8.60) | 17.69 | (24.13) |
| 6400 | 120 | 7.88 | (9.17) | 10.59 | (16.40) |
| 6400 | 160 | 7.98 | (9.12) | 6.19 | (7.81) |

$n = 96$, $\bar{q} = 8$, $\mu = 20$

Table 6: CPU times (in seconds) for different set-up cost and capacity characteristics

our method, the averages and maxima are taken over 4 problem instances with different demand patterns. Because Chen *et al.*'s algorithm may have a less stable behavior, a total of 20 instances (5 for each demand pattern) were generated for each capacity and set-up cost combination.

Our results for Chen *et al.*'s algorithm are consistent with those reported by the authors themselves. In general, the CPU times increase when the capacity restrictions become tighter as a result of lower capacities and/or higher set-up costs. Again we observe the stable behavior of our algorithm.

As we already know, the running time of our algorithm is very sensitive to scaling, whereas the running time of Chen *et al.*'s algorithm turned out to be much more robust in this respect. For instance, when $\mu$ is taken equal to 200 instead of 20, then our algorithm is only competitive with Chen *et al.*'s for the most capacity constrained type of test problems, whereas for the least constrained problems it was actually slower.

In view of the above discussion, we consider Chen *et al.*'s and our method as being complementary, rather than competing. We would like to stress, however, that the computational comparison between the two methods has only limited value, because only a certain type of problems are considered. For instance, our approach is applicable to problems with general holding cost functions. This means that if the linear holding cost functions of the test problems would be replaced by arbitrary functions which can be evaluated in constant time, then the CPU times would increase by at most a constant factor. Chen *et al.*'s method can only be applied to such problem instances after the holding cost functions are first approximated by piecewise linear functions. If one chooses to do this, one should keep in mind that Chen *et al.* have observed that the CPU time of their method increases at a rate which is approximately cubic in the number of pieces of the cost functions. Hence, our method will become relatively more attractive when the holding cost functions are non-linear. Moreover, features such as backlogging, start-up costs and bounds on inventory levels can easily be incorporated into our approach, and – as can be seen from the analysis in Sections 3 and 4 – such extensions will increase the CPU times by at most a constant factor, and sometimes even decrease the CPU times. On the other hand, incorporating these features into Chen *et al.*'s approach is likely to increase the CPU times more dramatically, because it boils down to introducing extra pieces into the cost functions. Again we expect that our method will become relatively more attractive. (We did not carry out these additional computational comparison, because Chen *et al.*'s FORTRAN code does not handle such problem features.)

Finally, we would also like to mention that our method has the advantage that it is easier to understand and implement than Chen *et al.*'s.

## 6 Conclusion

We have presented a dynamic programming algorithm for the Capacitated Economic Lot Size problem with general holding costs and piecewise linear production costs. Although our method is non-trivial, it has a simple and elegant structure, and is capable of solving a broad class of problems encountered in practice, including features such as start-up costs, backlogging and bounds on the inventory levels. The simple structure also makes it easy to implement. Moreover, we have shown that the algorithm is capable of solving quite large problem instances within a reasonable amount of time.

**Acknowledgement**

The authors wish to thank Professors Hsin-Der Chen, Donald W. Hearn and Chung-Yee Lee for making their code available to us, and Professors Awi Federgruen and Guillermo Gallego for their comments on an earlier draft of this paper. The authors are also grateful to Mr. Hung-Chang Wu for his work on coding the computer program. The first author also thanks the anonymous referees for their comments on an earlier draft of the paper.

## References

[1] A. Aggarwal and J.K. Park, 1994. More Applications of Monge-Array Techniques to Economic Lot-Size Problems, Research Memorandum, IBM T.J. Watson Research Center, Yorktown Heights, New York

[2] A.V. Aho, J.E. Hopcroft and J.D. Ullman, 1983. *Data Structures and Algorithms*, Addison-Wesley Publishing Company, Reading, Massachusetts

[3] K.R. Baker, P. Dixon, M.J. Magazine and E.A. Silver, 1978. An Algorithm for the Dynamic Lot-Size Problem with Time-Varying Production Capacity Constraints, *Management Science* **24**, 1710-1720.

[4] G.R. Bitran and H.H. Yanasse, 1982. Computational Complexity of the Capacitated Lot Size Problem, *Management Science* **28**, 1174-1186.

[5] H.-D. Chen, D. Hearn and C.-Y. Lee, 1994a. A New Dynamic Programming Algorithm for the Single Item Capacitated Dynamic Lot Size Model, *Journal of Global Optimization* **4**, 285-300.

[6] H.-D. Chen, D. Hearn and C.-Y. Lee, 1994b. A Dynamic Programming Algorithm for Dynamic Lot Size Models with Piecewise Linear Costs, *Journal of Global Optimization* **4**, 397-413.

[7] C.-S. Chung and C.-H. M. Lin, 1988. An $O(T^2)$ Algorithm for the *NI/G/NI/ND* Capacitated Lot Size Problem, *Management Science* **34**, 420-426.

[8] C.-S. Chung, J. Flynn and C.-H. M. Lin, 1994. An Effective Algorithm for the Capacitated Single Item Lot Size Problem, *European Journal of Operational Research* **75**, 427-440.

[9] G.D. Eppen and R.K. Martin, 1987. Solving Multi-item Capacitated Lot-sizing Problems using Variable Redefinition, *Operations Research* **35**, 832-848.

[10] S.S. Erenguc and Y. Aksoy, 1990. A Branch and Bound Algorithm for a Single Item Nonconvex Dynamic Lot Sizing Problem with Capacity Constraints, *Computers and Operations Research* **17**, 199-210.

[11] A. Federgruen and M. Tzur, 1991. 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**, 909-925.

[12] M. Florian and M. Klein, 1971. Deterministic Production Planning with Concave Costs and Capacity Constraints, *Management Science* **18**, 12-20.

[13] M. Florian, J.K. Lenstra and A.H.G. Rinnooy Kan, 1980. Deterministic Production Planning: Algorithms and Complexity, *Management Science* **26**, 669-679.

[14] J.M.Y. Leung, T.L. Magnanti and R. Vachani, 1989. Facets and Algorithms for Capacitated Lot Sizing, *Mathematical Programming* **45**, 331-359.

[15] Ö. Kirca, 1988. An Efficient Algorithm for the Capacitated Single Item Dynamic Lot Size Problem, *European Journal of Operational Research* **45**, 15-24.

[16] V. Lofti and Y.-S. Yoon, 1994. An Algorithm for the Single Item Capacitated Lot- sizing Problem with Concave Production and Holding Costs, *Journal of the Operational Research Society* **45**, 934-941.

[17] S.F. Love, 1973. Bounded Production and Inventory Models with Piecewise Concave Costs, *Management Science* **20**, 313-318.

[18] Y. Pochet, 1988. Valid Inequalities and Separation for Capacitated Economic Lot Sizing, *Operations Research Letters* **7**, 109-116.

[19] L. Schrage, 1984. The multiproduct lot scheduling problem, *Deterministic and Stochastic Scheduling*, M.A.H. Dempster, J.K. Lenstra and A.H.G. Rinnooy Kan, eds., Nato Advanced Study Institutes Series, D. Riedel, Holland.

[20] A.F. Veinott, Jr., 1964. Production Planning with Convex Costs: A Parametric Study, *Management Science* **10**, 441-460.

[21] C.P.M. Van Hoesel and A.P.M. Wagelmans, 1995. An $O(T^3)$ Algorithm for the Economic Lot-Sizing Problem with Constant Capacities, *Management Science* (to appear).

[22] S. Van Hoesel, A. Wagelmans and B. Moerman, 1994. Using Geometric Techniques to Improve Dynamic Programming Algorithms for the Economic Lot–Sizing Problem and Extensions, *European Journal of Operational Research* **75**, 312–331.

[23] H.M. Wagner and T.M. Whitin, 1958. Dynamic Version of the Economic Lot Size Model, *Management Science* **5**, 89-96.

[24] W. Zangwill, 1966. A Deterministic Multi-Period Production Scheduling Model with Backlogging, *Management Science* **13**, 105-119.