European Journal of Operational Research 75 (1994) 312–331
North-Holland

Using geometric techniques to improve dynamic programming algorithms for the economic lot-sizing problem and extensions

Stan van Hoesel

Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, Netherlands

Albert Wagelmans and Bram Moerman

Econometric Institute, Erasmus University Rotterdam, P.O. Box 1738, 3000 DR Rotterdam, Netherlands

Revised January 1993

Abstract: In this paper we discuss two basic geometric techniques that can be used to speed up certain types of dynamic programs. We first present the algorithms in a general form, and then we show how these techniques can be applied to the economic lot-sizing problem and extensions. Furthermore, it is illustrated that the geometric techniques can be used to give elegant and insightful proofs of structural results, like Wagner and Whitin's planning horizon theorem. Finally, we present results of computational experiments in which new algorithms for the economic lot-sizing problem are compared with each other, as well as with other algorithms from the literature.

Keywords: Dynamic programming; Computational analysis; Lot-sizing; Inventory

1. Introduction

As argued by Denardo [4], the complexity of a dynamic programming based algorithm is usually proportional to the number of arcs in the underlying dynamic programming network. Only when special cost structures on the arcs of the network are incurred one might hope for a better running time. One of the first such improvements is due to Yao [23], who uses a monotonicity property to speed up the dynamic programming algorithm for finding optimal binary search trees. Aggarwal and Park [1] show that a similar monotonicity property holds for the dynamic programming formulation of the well-known economic lot-sizing problem (ELS), and they arrive at an $O(T \log T)$ algorithm, T being the length of the planning horizon. They also show that in some interesting special cases a linear time algorithm exists.

Correspondence to: Dr. A. Wagelmans, Econometric Institute, Erasmus University Rotterdam, P.O. Box 1738, 3000 DR Rotterdam, Netherlands.

0377-2217/94/$07.00 © 1994 – Elsevier Science B.V. All rights reserved
SSDI 0377-2217(93)E0196-5

These are significant improvements over the well-known dynamic programming algorithm proposed by Wagner and Whitin [22]. Similar improvements have been independently obtained by Federgruen and Tzur [7] and Wagelmans et al. [21]. In the latter article, the exposition is of a geometric nature.

The aim of the current paper is threefold. First, in Section 2, we generalize the approaches of [7] and [21] by discussing two basic geometric techniques that can be used to speed up certain types of dynamic programs. Our discussion includes the particular data structures needed to implement the improved algorithms. In Sections 3 and 4 we will show how, given the generalized algorithms, improved complexity results for several lot-sizing problems can easily be established. In particular, we discuss the forward and backward recursion for ELS, the extension to the model which allows backlogging, ELS with start-up costs and a generalized version of the model with learning effects in set-up costs. It suffices to show that the recursion formulas are of the form which allows the application of the geometric techniques. Because we regard the resulting algorithms as special cases of the general algorithms, it will become clear that the techniques used in [7] and [21] are essentially the same.

The second aim of this paper is to point out that the geometric techniques enable us to give elegant and insightful proofs of certain structural properties. This is done in Section 5 where two examples are given, one of which concerns a proof of Wagner and Whitin's planning horizon theorem.

The last contribution of this paper is found in Section 6, where we present results of computational experiments in which the algorithms for ELS introduced in [1], [7] and [21] are compared with each other, as well as with existing implementations of the Wagner–Whitin algorithm (cf. Evans [5], Saydam and McKnew [15]). Our experiments indicate that especially the algorithm based on the backward recursion performs very well.

The paper ends with concluding remarks in Section 7.

2. Geometric solution techniques for dynamic programs

A basic topic in computational geometry is the determination of the lower concave envelope of a given set of lines in the plane. A dual problem, in a sense, is the problem of determining the convex hull of a given set of points in the plane. For both problems efficient solution techniques exist. These solution techniques will be used to solve the following type of recursion formulas for dynamic programs.

$$G(0) := 0,$$

$$G(i) := A_i + \min_{0 \le j < i} \{G(j) + B_j + C_i D_j\}, \quad 1 \le i \le n. \tag{1}$$

Here A_i, B_j, C_i, D_j $(1 \le i \le n, 0 \le j \le n-1)$ are constants that only depend on the index. In this section, these constants are supposed to be given beforehand.

We proceed by showing how the dynamic programming recursion (1) can be viewed as a problem of finding the lower concave envelope of a set of lines. To calculate the minimum of the terms $\{G(j) + B_j + C_i D_j \,|\, 0 \le j < i\}$ for a fixed i, these terms are considered as functions of C_i. Each term corresponds to a line or a linear function m_j $(0 \le j < i)$, where

$$m_j(x) = G(j) + B_j + D_j x,$$

i.e., m_j is the line with slope D_j that passes through the point $(0, G(j) + B_j)$. Obviously, the determination of the minimum of $\{G(j) + B_j + C_i D_j \,|\, 0 \le j < i\}$ is equivalent to finding the minimum of the values $m_j(C_i)$ over j, $0 \le j < i$. This minimum is defined as $g_i(C_i)$, where g_i is the concave lower envelope of the lines m_j $(0 \le j < i)$. See Figure 1 for an example with $i = 3$. Note that g_i is piecewise linear. Given $g_i(C_i)$, one can determine the dynamic programming variable $G(i)$, which is simply $A_i + g_i(C_i)$. An algorithm based on this observation proceeds by calculating the variables $G(i)$ $(i = 1, \dots, n)$, iteratively, in forward fashion. During this process the lower envelope of the lines is updated, since a new line m_i is

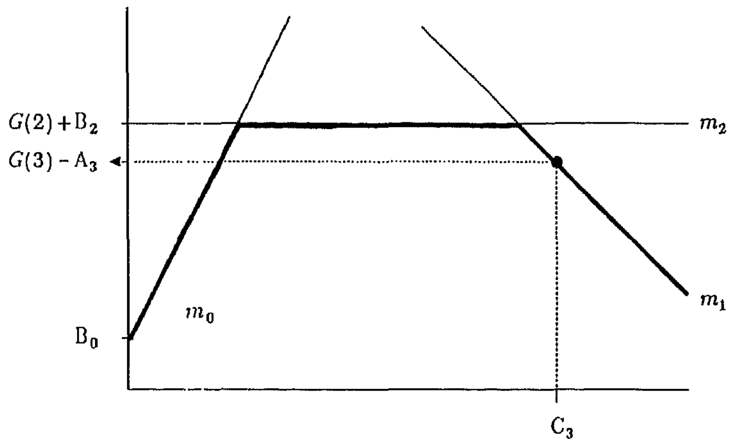

Figure 1. Determination of $G(3)$ using the lower concave envelope

added to the set of lines in iteration i. The operations in each iteration are divided in the following two steps:

Step 1. Find the value $g_i(C_i)$, the function value of the lower envelope of the lines m_j $(j < i)$ at C_i. Then $G(i)$ is calculated by adding A_i to $g_i(C_i)$.

Step 2. Add the line m_i to the set of lines that already determine the lower concave envelope g_i, where m_i is defined as

$$m_i(x) = G(i) + B_i + D_i x.$$

Recalculate the lower envelope g_{i+1}, where

$$g_{i+1}(x) := \min\{g_i(x), m_i(x)\}.$$

For an analysis of the running time of an algorithm based on this description a suitable data structure to maintain the lower envelope g_i should be chosen. In order to choose the right structure it is necessary to find out which elementary operations must be performed. Therefore, the two steps of the algorithm will be analyzed in some more detail.

In Step 1 of the iteration where $G(i)$ is determined, the function value of g_i at $x = C_i$ has to be calculated. This amounts to the search of the line segment of g_i that contains C_i.

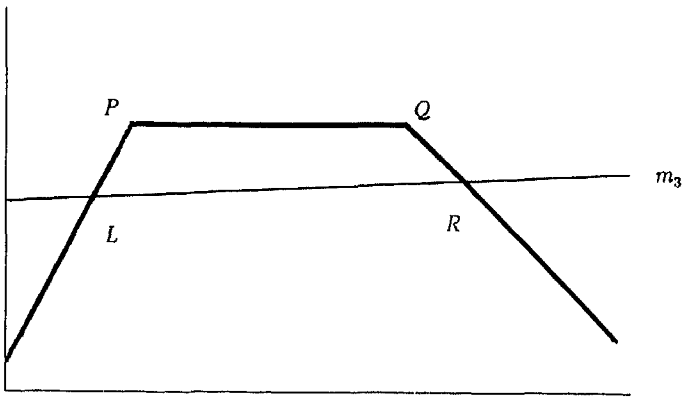

Figure 2. Calculation of $g_4(x)$

In Step 2 the line m_i is added to the concave lower envelope. In order to do so, the points where m_i and g_i meet should be calculated, if they exist, and the line segments of g_i between these points, should be replaced by m_i. This procedure is composed of the following two steps (see also Figure 2).

Step 2a. First, detect a breakpoint of g_i that lies above m_i, if one exists. Since the line segments of g_i have decreasing slopes, when viewed in order of increasing x, it suffices to search the breakpoint B, for which the left line segment incident to B, has slope larger than D_i and the right line segment has slope at most D_i. If this breakpoint lies above the line m_i, then the lower envelope should be updated. Otherwise it follows from the concavity of g_i that $g_{i+1} = g_i$ and therefore, we are finished with Step 2 in that case.

Note that there are two exceptions, namely the case in which all line segments have slopes larger than D_i, and the case in which all line segments have smaller slopes. In the first case it suffices to check whether the right-most breakpoint of g_i lies below m_i. If so, m_i and g_i will intersect only somewhere to the right of the right-most breakpoint, i.e., we will determine the intersection point of m_i with the right-most segment of g_i. Otherwise, m_i can only intersect g_i to the left of the breakpoint. How this intersection point can be found follows from Step 2b below. The second case is handled analogously.

Step 2b. Now suppose that the breakpoint P of g_i lies above m_i. Then first the line segments to the right of P are inspected. Let the first breakpoint to the right of P be denoted by Q. There are two possibilities: Q lies below m_i or Q lies above (or on) m_i. In the first case m_i intersects the line segment defined by P and Q. The intersection point, say R, is calculated, and the line segment $[P, Q]$ is deleted. Moreover, the line segment $[R, Q]$ is added. In the second case the line segment $[P, Q]$ is deleted and we proceed by inspecting the line segment to the right of Q in the same manner, until an intersection point R is found. An analogous procedure inspects the line segments left of P, until an intersection point L is found. Finally, the line segment $[L, R]$ is added to g_{i+1}.

A suitable data structure that maintains the lower envelope should be able to perform the following operations quickly. In Step 1 a line segment must be determined which contains the point corresponding to a given value of x. In Step 2a a breakpoint should be determined for which the slopes of the line segments incident to this point, s_l and s_r, satisfy $s_l \leq s < s_r$ for a given number s. In Step 2b a neighboring line segment should be detected. Moreover, one or more line segments may be deleted and one line segment may be added.

Thus the operations that have to be performed are FIND, ADD and DELETE. A balanced tree, like a 2-3 tree (or an AVL-tree) supports these operations in time proportional to the logarithm of the number of nodes, see Aho et al. [2].

The 2-3 tree is designed such that the leaves contain the line segments of the lower envelope. The information maintained in each leaf consists of the endpoints of the line segment including their value, and the slope of the line segment. The number of line segments is bounded by n, since each line segment corresponds to exactly one line m_j. The interior nodes of the tree contain the interval that is covered by the line segments in the subtree of the interior node. Moreover such a node contains information on the leftmost breakpoint and the leftmost (highest) slope, as well as the same information on the rightmost breakpoint and slope in the subtree. An example is given in Figure 3.

Theorem 1. *The dynamic programming recursion* (1) *can be solved in* $O(n \log n)$ *time.*

Proof. Due to the choice of the data structure the basic operation of determining the value $g_i(x)$ for some x, takes $O(\log n)$ time. Moreover, the deletion and addition of a line segment also takes $O(\log n)$ time. It remains to be shown, therefore, that the number of times that these operations are executed is bounded by $O(n)$.

Step 1 takes $O(\log n)$ time per iteration, since only $g_i(C_i)$ must be determined, and there are n iterations.

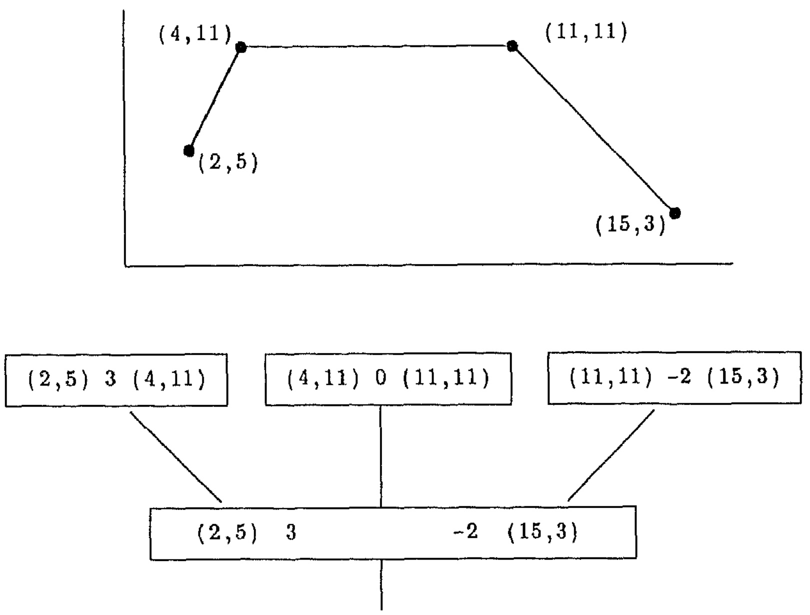

Figure 3. Partial lower envelope and corresponding nodes in 2-3 tree

Step 2a takes O(log n) time per iteration, since only a breakpoint for which the slopes of the left and right line segments are larger and smaller than D_i, respectively, has to be determined.

Step 2b contains several operations. First, at most one line segment is added, which takes O(log n) time. Second, at most two line segments are shortened, which also takes O(log n) time. So these operations take O(n log n) time overall. Third, some line segments may be deleted. However, the number of deletions throughout the algorithm is bounded by n, since the line corresponding to a deleted segment will not return in the lower envelope. Each deletion takes O(log n) time. □

In the following we show under which conditions the different steps can be implemented in linear time.

Lemma 2. *If the C_i $(1 \leq i \leq n)$ are monotone in i, then Step 1 can be performed in linear time.*

Proof. Without loss of generality we may assume that the C_i are non-decreasing in i. We will focus on the intervals on the horizontal axis which correspond to the different line segments of g_i. Given the interval that contains the point C_{i-1}, C_i can be found in this interval or in an interval to the right of C_{i-1}. Therefore we can find the interval that contains C_i, by searching in the direction to the right. To facilitate locating a neighboring right leaf in the 2-3 tree, a doubly linked list is introduced, which connects neighboring leaves. The doubly linked list is incorporated to find a neighboring interval in constant time. Each time an interval is inspected, we either pass it (if it does not contain C_i) or we stop (if C_i is contained in it). A stop happens only once per iteration, and therefore there are O(n) stops. If a pass happens, then the interval will not be inspected anymore, since the C_i are non-decreasing. Therefore, this interval will not be considered when searching for the interval containing C_j for all $j \geq i$. Because the number of intervals is O(n), passes also happen only O(n) times. Hence, given the lower envelopes in every iteration, the total computational effort spent on evaluating $g_i(C_i)$ for all $i = 1, \ldots, n$ is O(n). This completes the proof. □

Lemma 3. *If the D_j $(1 \leq j \leq n)$ are monotone in j, then Step 2 can be performed in linear time. Moreover, the 2-3 tree can be replaced by a stack.*

Proof. Without loss of generality we may suppose that the D_j are non-increasing in j. Since D_i, the slope of m_i, is smaller than or equal to the slopes of all other lines m_j $(0 \le j < i)$, addition of a line segment takes place only at the right of the lower envelope g_i. Therefore, it suffices to check whether the rightmost line segment of g_i intersects m_i or lies completely above m_i. This can simply be checked by testing whether the last breakpoint of g_i lies below or above m_i. If the latter is the case, then this line segment is deleted, and the next to last line segment will be inspected, and so on, until a line segment is found that intersects m_i. Then the intersection point of m_i and that line segment is calculated, and this point together with its corresponding line segment are added to the lower envelope.

We use a stack to keep track of the lower envelope. The line segments appear in order of increasing slope in the stack, i.e., the top of the stack corresponds to the right-most line segment, and so on. Each time a line segment is deleted the corresponding line does not return in the lower envelope, and therefore this happens O(n) times. A deletion itself takes constant time, since it is always performed at the right of g_i, and thus the top of the stack. An addition is performed at the top of the stack as well, and happens only once per iteration.

Finally, it should be noted that the stack can also be used to perform binary searches in order to find the value of C_i in the lower envelope. □

Because the search features of a doubly linked list are trivially implemented in a stack, combining Lemmas 2 and 3 leads to the following result.

Corollary 4. *If the C_i and D_j $(1 \le i, j \le n)$ are monotone with respect to their indices, the dynamic programming recursion* (1) *can be solved in time proportional to n by using a stack as the data structure to maintain the lower envelope.*

A final remark concerns the constants A_i, B_j, C_i and D_j $(1 \le i, j \le n)$. These constants need not be available at any time in the procedure, but it suffices that they can be evaluated in constant time at the moment they should be available.

Another geometric technique, one that is closely related to the concept of using the lower concave envelope of lines, considers the lower convex envelope of a set of points. It is used to solve the following variant of the dynamic programming recursion (1).

$$G(0) := 0,$$
$$G(i) := a_i + \min_{0 \le j < i} \{G(j) + b_j + c_i(d_i - d_j)\}, \quad 1 \le i \le n. \tag{2}$$

Dynamic programming recursions (1) and (2) are easily seen to be equivalent, by the following substitutions performed on (2):

$$A_i = a_i + c_i d_i, \qquad B_i = b_i,$$
$$C_i = -c_i, \qquad D_i = d_i,$$

for $i = 1, \dots, n$. Because of the close relation between the formulations, it will not come as a surprise that the complexity results concerning the geometric technique that we are going to describe, resemble our earlier obtained complexity results

Again, we start by describing what happens in an iteration of the algorithm. Suppose that for each j smaller than i, the corresponding variable $G(j)$ has been determined. Consider the points $(d_j, G(j) + b_j)$ in the plane. Through each of the points a line with slope c_i is drawn. For a fixed j, this line, denoted by l_j, is defined as the following function of x:

$$l_j(x) = G(j) + b_j + (x - d_j)c_i.$$

Since $l_j(d_i)$ is exactly the term between braces in (2) for a fixed j, we are interested in $\min\{l_j(d_i) \mid 0 \le j < i\}$ and therefore the line l_j $(j < i)$ lying lowest at $x = d_i$ has to be found. This line is easily determined if the

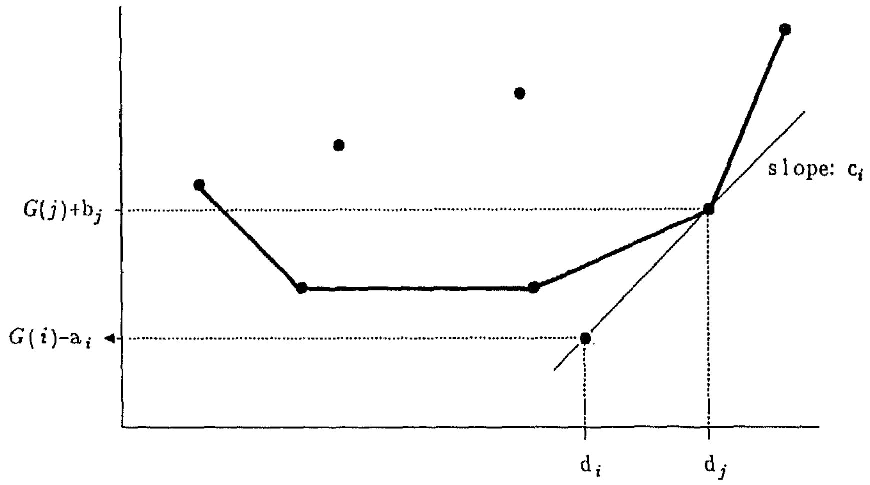

Figure 4. The determination of $G(i)$ using the convex lower envelope

convex lower envelope g_i of the set of points $\{(d_j, G(j)+b_j) \mid j<i\}$ is available. Finding the lowest lying line, then amounts to finding the breakpoint of the convex lower envelope where a line with slope c_i is tangent to this envelope (see Figure 4). This is equivalent to searching for the breakpoint for which the line segments adjacent to it change slope from smaller than c_i to greater than or equal to c_i.

The lower envelope is maintained by a 2-3 tree in which the leaves contain the information on the line segments (endpoints and slopes), in order of increasing x-coordinate. The interior nodes contain all information of the first leaf contained in their subtree, together with the length of the interval that is covered by the line segments in the subtree. In each iteration, two steps have to be performed, one to determine $G(i)$ and one to update the lower envelope.

Step 1. Search for the breakpoint, where the line segments adjacent to it, change slope from smaller than c_i to greater than or equal to c_i. G_i can now easily be determined.

Step 2. Add $(d_i, G(i)+b_i)$ to the lower envelope, and update it (see also Figure 5):

Step 2a. Determine the position of this point, i.e., search for the interval (corresponding to a line segment of g_i) that contains d_i. If it turns out that $(d_i, G(i)+b_i)$ lies above g_i, then stop. Otherwise perform Step 2b, the reconstruction of the lower envelope.

Step 2b. Consider the breakpoints corresponding to the endpoints of the interval containing d_i. The line segment between these breakpoints is deleted. Then the breakpoint to the right of d_i, say P, is

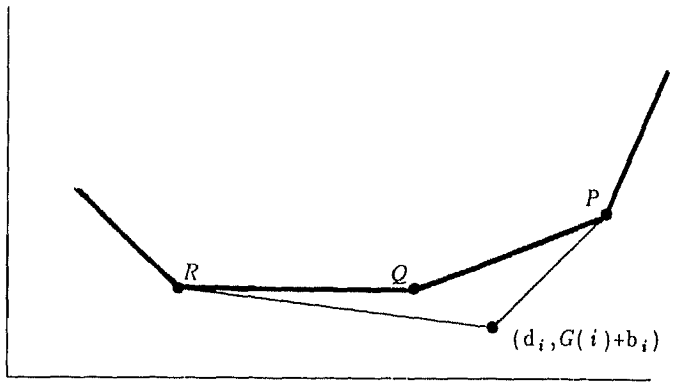

Figure 5. The addition of a point to the lower convex envelope

checked. If the slope of the line segment between $(d_i, G(i)+b_i)$ and P is greater than the slope of the line segment to the right of P, the latter is deleted from the lower envelope and this process is repeated. Otherwise, if its slope is smaller than the slope of the segment to the right of P, then the process is stopped, since the part of the lower envelope to the right of $(d_i, G(i)+b_i)$ is convex again. Subsequently, a similar procedure is performed for the part of g_i to the left of $(d_i, G(i)+b_i)$.

The operations that are performed in Steps 1 and 2 are easily seen to take O(log n) time using a 2-3 tree, since they amount to searching, addition and deletion of line segments. Moreover, analogous to the reasoning with respect to the previous geometric algorithm in this section, it is fairly easy to see that the operations are performed O(n) times. This provides the proof for the following theorem.

Theorem 5. *The geometric implementation of dynamic programming recursion* (2) *takes* O(n log n) *time.*

Analogous to the implementation of (1), we can speed up the algorithm if the c_i and/or d_i are monotone in the index i. The proofs of the following two lemmas are left to the reader.

Lemma 6. *Step* 1 *can be implemented in linear time if the* c_i *are monotone in* i.

Lemma 7. *Step* 2 *can be implemented in linear time if the* d_j *are monotone in* j. *Moreover, the* 2-3 *tree can be replaced by a stack.*

Combining these lemmas we obtain the following result.

Corollary 8. *If the* c_i *and* d_j *are monotone, then dynamic programming recursion* (2) *is solvable in linear time.*

The relation between the two geometric techniques discussed in this section can be pointed out now. Looking at the substitutions that show the equivalence between dynamic programming recursions (1) and (2), we immediately see that the conditions under which it suffices to use a stack (Lemmas 3 and 7), as well as the conditions under which a linear time algorithm exists (Corollaries 4 and 8) are the same. In fact, the two geometric techniques can be viewed as two different ways of presenting the same underlying algorithm. The techniques are dual in the sense that the roles that points and lines play in the first technique are exchanged in the second one. In the first technique each index corresponds to a line, and in the second each index corresponds to a point. Moreover, the lower envelope of lines corresponds to the lower envelope of points. More about the duality of both concepts can be found in Preparata and Shamos [14].

The dynamic programs (1) and (2) are both forward recursions in the sense that the dynamic programming variables are expressed as functions of lower indexed dynamic programming variables. For the applications that will be presented in Sections 3 and 4 it is sometimes more convenient to consider dynamic programs formulated in a backward fashion. For instance, instead of (2) we may consider the following recursion:

$$\begin{aligned} &G'(n+1) := 0, \\ &G'(i) := a_i' + \min_{i<j\le n+1} \{G'(j) + b_j' + c_i'(d_i' - d_j')\}, \quad 1 \le i \le n. \end{aligned} \tag{3}$$

It is obvious that this program can be solved analogously to (2). The only difference is that the dynamic programming variables are now computed in order of decreasing index. A similar remark holds for the backward version of (1).

Finally, we should mention that the results of the Corollaries 4 and 8 can be further generalized as has been done, among others, by Klawe [11], who solves the following dynamic programming recursion:

$$G(i) := \min_{j>i} \{G(j) + w_{ij}\} \tag{4}$$

where the w_{ij} $(1 \le i < j \le n)$ satisfy a monotonicity condition. This condition states that for each i and j $(i < j)$:

$$w_{i,j} + w_{i+1,j+1} \ge w_{i+1,j} + w_{i,j+1}.$$

In fact, it suffices to require a monotonicity with respect to the column minima in the matrix of elements $G(j) + w_{ij}$. Note that if the C_i and D_j are monotone, then the dynamic programming recursions (1) and (2) satisfy the monotonicity condition (take $w_{ij} := A_i + B_j + C_i D_j$). The solution method of Klawe can therefore be used to solve the linear case of the dynamic programming recursion. However, Aggarwal and Park [1] show that a divide-and-conquer strategy, applied when the C_i and D_j are not monotone, provides an $O(n \log n)$ algorithm for the general case as well.

3. Improved dynamic programming algorithms for ELS

We consider the economic lot-sizing problem (ELS) which was introduced, in a slightly restricted version, by Wagner and Whitin [22]. The length of the planning horizon is denoted by T, and we let d_t $(t = 1,\dots,T)$ denote the non-negative demand in period t. The set-up costs are denoted by f_t $(t = 1,\dots,T)$, and they are incurred whenever production takes place in period t. We denote the unit production costs by p_t $(t = 1,\dots,T)$ and the unit inventory costs by h_t $(t = 1,\dots,T)$. For notational convenience we define for every parameter vector α:

$$\alpha_{ij} := \textstyle\sum_{t=i}^{j} \alpha_t.$$

Note that it takes linear time to calculate α_{1t} for all $t = 1,\dots,T$.

The objective is to satisfy all demand at minimum cost, such that inventories are never negative and the starting and ending inventory are zero. An equivalent problem results if we take all inventory costs equal to 0 and replace p_t by

$$r_t \equiv p_t + h_{tT}, \quad t = 1,\dots,T$$

(see Wagelmans et al. [21]). In this section, we will consider this transformed model. Moreover, for convenience we will assume in the remainder of the paper that demand is positive in every period. The modifications needed to allow for zero demand are straightforward.

We will show that the well-known forward and backward recursion for ELS, which are both readily seen to be solvable in $O(T^2)$ time, fit into the descriptions of the dynamic programming recursions formulated in the preceding section. Surprisingly, there is an important difference in the implementation of the geometric techniques when applied to the two recursions. To show this difference we will view both recursions as special cases of the dynamic programs (1) and (2), and we will also give a brief description of the algorithms.

3.1. Forward recursion

The forward recursion uses variables $F(t)$ which denote the value of the optimal solution for the problem restricted to the planning horizon $1,\dots,t$. The usual forward recursion can be rewritten as

$$F(0) := 0,$$
$$F(t) := \min_{0<\tau\le t} \{F(\tau-1) + (f_\tau - r_\tau d_{1,\tau-1}) + d_{1t} r_\tau\}, \quad 1 \le t \le T. \tag{5}$$

Recall recursion (1) which is

$$G(0) := 0,$$
$$G(i) := A_i + \min_{0\le j<i} \{G(j) + B_j + C_i D_j\}, \quad 1 \le i \le n.$$

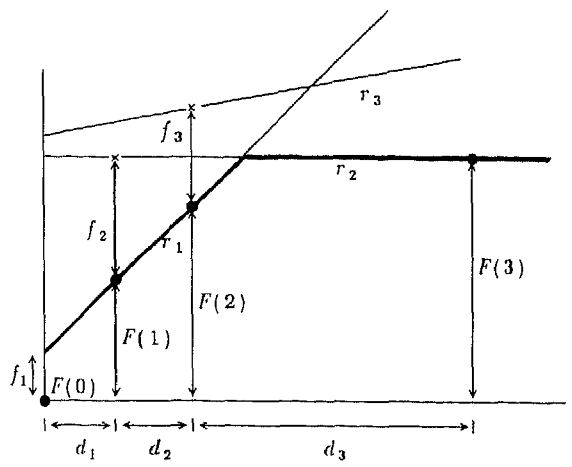

Figure 6. The lower envelope for the forward recursion

It follows that t plays the role of i in (1) and that $\tau - 1$ corresponds to j. Moreover, A_i is replaced by zero, B_j is replaced by $f_\tau - r_\tau d_{1,\tau-1}$, C_i is replaced by d_{1t} and D_j is replaced by r_τ. Especially the last two replacements turn out to be important, because d_{1t} is monotone in t, while r_τ is (by definition) non-increasing in τ in the important case that there are no speculative motives to hold inventory.

The forward algorithm is illustrated in Figure 6. It is essentially the algorithm that was introduced by Federgruen and Tzur [7], although they did not use geometric arguments in their exposition.

Since d_{1t} is non-decreasing in t, evaluating the concave lower envelope in these values take O(T) time in total (cf. Lemma 2). However, since r_τ need not be monotone in τ, a 2–3 tree is needed to keep track of the lower envelope. Therefore, the latter takes O(T log T) time. In case r_τ is monotone, this part of the algorithm can also be performed in linear time (cf. Lemma 3). As already mentioned above, r_τ is non-increasing in the model without speculative motives. Thus, the following theorem is immediate.

Theorem 9. *The forward recursion for ELS can be solved in O(T log T) time, using a 2–3 tree to store the lower envelope of lines. If there are no speculative motives to hold inventory, then the forward recursion can be solved in O(T) time.*

3.2. Backward recursion

The backward recursion can be rewritten as follows.

$$B(T+1) := 0;$$
$$B(t) := f_t + \min_{t<\tau\le T+1}\{B(\tau) - r_t(d_{1,t-1} - d_{1,\tau-1})\}, \quad 1 \le t \le T. \tag{6}$$

It is easy to see that $B(t)$ actually denotes the value of the optimal solution for the problem restricted to the planning horizon $t,\ldots,T$.

Recall recursion (3): $G'(n+1) := 0$; $G'(i) := a'_i + \min_{i<j\le n+1}\{G'(j) + b'_j + c'_i(d'_i - d'_j)\}$ $(1 \le i \le n)$. It follows that t plays the role of i in (3), and that τ corresponds to j. Moreover, a'_i is replaced by f_t, b'_j is replaced by zero, c'_i is replaced by $-r_t$ and $d'_i - d'_j$ is replaced by $d_{1,t-1} - d_{1,\tau-1}$.

The backward algorithm is illustrated in Figure 7. Note that the point (0,0) corresponds to $(d_{T+1,T}, B(T+1))$.

The complexity of the backward algorithm is similar to the complexity of the forward algorithm. However, there is a slight difference in the implementation of both. Due to the fact that d_{1t} is monotone in t, updating the convex lower envelope in the backward algorithm can be performed in O(T) time (cf.

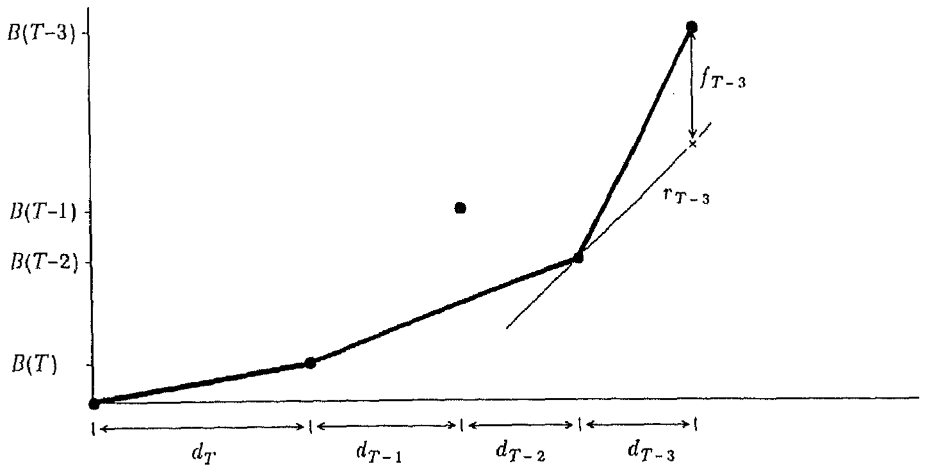

Figure 7. The lower envelope for the backward recursion

Lemma 7). This implies that there is no need to implement the lower envelope with a 2-3 tree, in the backward algorithm. A simple stack always suffices. Finally, if the production costs r_t are monotone, then it follows from Lemma 6 that the entire algorithm can also be implemented in linear time. This implies the following theorem.

Theorem 10. *The backward algorithm can be implemented with a running time of* $O(T \log T)$ *using a stack to keep track of the lower envelope. If there are no speculative motives to hold inventory, then the backward algorithm runs in* $O(T)$ *time.*

The algorithm described in this subsection is essentially the one that was introduced in Wagelmans et al. [21]. From the preceding subsection and our remarks about the relationship between the two basic geometric techniques in Section 2, it follows that the algorithms presented in [7] and [21] are based on the same technique. The difference lies solely in the recursion formulas used.

4. Applications to extensions of ELS

The methods developed in Section 2 can also be applied to extensions of ELS. We will consider the model with backlogging, the model with start-up costs and a generalized version of the model with learning effects in the set-up costs. This section also serves to illustrate the general scheme one has to follow in order to apply the geometric techniques. In most cases, first a preprocessing phase is needed to obtain dynamic programming formulations of the desired form. Often this boils down to calculating the constants that were assumed to be given beforehand in Section 2. Of course, this phase should not be too time consuming. Typically, the successful applications have a linear time preprocessing phase.

4.1. ELS with backlogging

In the economic lot-sizing problem with backlogging (ELSB) inventory is allowed to be negative. In case of a negative inventory in period t, a cost of h_t^- per unit is incurred, while h_t^+ now denotes the usual unit inventory cost. The problem can be solved using the following double recursion due to Zangwill [24]:

$$H(s) := \min_{s \le t \le T+1} \left\{ p_t d_{s,t-1} + \sum_{\tau=s}^{t-1} h_\tau^- d_{s\tau} + H'(t) \right\}, \quad 1 \le s \le T, \tag{7a}$$

$$H'(t) := \min_{t < u \le T+1} \left\{ f_t + p_t d_{t,u-1} + \sum_{\tau=t}^{u-1} h_\tau^+ d_{\tau-1,u-1} + H(u) \right\}, \quad 1 \le t \le T, \tag{7b}$$

with boundary conditions

$$H(T+1) := H'(T+1) := 0.$$

In this double recursion the variables $H(t)$ denote the optimal costs for the problem restricted to the planning horizon $t,\dots,T$. The variables $H'(t)$ denote the cost of an optimal solution of ELSB restricted to the planning horizon $t,\dots,T$, with the additional condition that t is a production period. The order in which the variables are determined is the following: if the variables $H'(t)$ and $H(t)$ have been calculated for $t = s+1,\dots,T$, then first $H'(s)$ is calculated, and then $H(s)$.

The double recursion is readily seen to be solvable in $O(T^2)$ time. In order to show that both recursion formulas actually are of the same type as the recursions (1) and (2), the following quantities are defined for $s \le t < u$:

$$S(s,t) = \sum_{\tau=s}^{t-1} h_\tau^- d_{s\tau}$$

and

$$S'(t,u) = \sum_{\tau=t}^{u-1} h_\tau^+ d_{\tau+1,u-1}.$$

Now the following formulas hold:

$$S(s,t) = S(1,t) - S(1,s) - h_{s,t-1}^- d_{1,s-1}$$

and

$$S'(t,u) = S'(1,u) - S'(1,t) - h_{1,t-1}^+ d_{t,u-1}.$$

Note that these values can be calculated in constant time, if a linear time preprocessing is performed in which the quantities h_{1t}^-, h_{1t}^+, d_{1t}, $S(1,t)$ and $S'(1,t)$ are calculated.

The dynamic programming recursions (7a) and (7b) can now be reformulated as follows:

$$\begin{aligned} &H(T+1) := 0, && \text{(8a)}\\ &H(s) := -S(1,s) + h_{1,s-1}^- d_{1,s-1}\\ &\qquad + \min_{s \le t \le T+1} \{H'(t) + p_t d_{1,t-1} + S(1,t) + (-h_{1,t-1}^- - p_t) d_{1,s-1}\}, \quad 1 \le s \le T,\\ &H'(T+1) := 0,\\ &H'(t) := f_t - S'(1,t)\\ &\qquad + \min_{t < u \le T+1} \{H(u) + S'(1,u) + (p_t - h_{1,t-1}^+) d_{t,u-1}\}, \quad 1 \le t \le T. && \text{(8b)}\end{aligned}$$

These reformulations can be solved using the geometric techniques of Section 2. Recursion formula (8a) resembles the backward version of (1). It calculates $H(s)$ from the values $H'(t)$ $(t = s+1,\dots,T)$ by use of the lower envelope of the lines through $(0, H'(t) + p_t d_{1,t-1} + S(1,t))$ with slope $-h_{1,t-1}^- - p_t$. Recursion (8b) resembles (3), the backward version of (2), and it calculates $H'(s)$ from the values $H(t)$ by use of the lower envelope of the points $(d_{tT}, H(t) + S'(1,t))$. Informally speaking, we keep track of two figures, one with a concave lower envelope and the other one with a convex lower envelope, between which we alternate.

The following theorem is a consequence of the results in Section 2 and the preceding exposition.

Theorem 11. *ELSB is solvable in* $O(T \log T)$ *time.*

As argued in Section 2, monotonicity conditions on the costs may speed up the algorithm, resulting in a running time which grows linearly in T. Since d_{1t} is already monotone in t, the conditions in Corollaries 4 and 8 are translated into the conditions given in the following theorem.

Theorem 12. *ELSB is solvable in* $O(T)$ *time, if* $p_t - h_t^- \le p_{t+1} \le p_t + h_t^+$, *for* $t \in \{1, \ldots, T-1\}$.

Proof. Updating and searching on the lower envelopes can be done in linear time if $p_t - h_{1,t-1}^+$ and $-h_{1,t-1}^- - p_t$ are monotone in t. Taking $p_t - h_{1,t-1}^+$ non-increasing and $h_{1,t-1}^- + p_t$ non-decreasing gives the condition mentioned in this theorem. □

We have presented a backward algorithm to solve ELSB. However, an algorithm based on a forward recursion acts essentially the same, since the forward double recursion is similar to the backward recursion. Furthermore, it should be mentioned that the results of Theorem 11 and 12 have independently been obtained by Aggarwal and Park [1] and Federgruen and Tzur [6].

4.2. ELS with start-up costs

In the economic lot-sizing problem with start-up costs (ELSS) a start-up is incurred in period t if there is a set-up in period t and no set-up in period $t-1$. Hence, start-up costs are incurred is some periods in addition to the regular set-up costs. The fixed cost associated with a start-up in period t is denoted by g_t. This type of fixed cost structure has been introduced by Schrage [16], and was later also studied by Karmarkar and Schrage [10] and Karmarkar, Kekre and Kekre [9]. The model for ELSS has first been studied by Van Wassenhove and Vanderhenst [20]. It is important to note that we allow start-ups and set-ups in periods in which no production takes place, and there may be an optimal solution in which this actually occurs. For instance, if set-ups consist mainly of maintenance activities, then it may be cheaper to maintain the ready-to-produce state of a machine for a few periods (without actually producing), than to perform a relatively expensive start-up. Because start-up costs need not be stationary, it may even be optimal to perform a start-up followed by a sequence of set-ups in consecutive periods before production actually starts.

One can show (see Van Hoesel [17]) that the well-known zero-inventory property of ELS also holds for ELSS, i.e., we may restrict ourselves to schedules, where the quantity produced in any production period t equals the cumulative demand of the periods $t, \ldots, \tau$ for some $\tau \ge t$, and no production period occurs in the periods $t+1, \ldots, \tau$. To include the start-up costs in a dynamic program, two types of dynamic programming variables are defined.

$K(t)$: The minimum cost of satisfying the demands of periods $t, \ldots T$ through production in these periods. Note that the possible costs of a start-up and set-ups in period t and earlier are included.

The following variables do not take into account the start-up and set-up costs that are incurred in the periods $1, \ldots, t$. Note that the production costs in t are included.

$K'(t)$: The minimum cost of satisfying the demands of periods $t, \ldots, T$ through production in these periods, where the start-up and set-up costs of the periods $1, \ldots, t$ are ignored.

Note that neither $K(t)$ nor $K'(t)$ denotes the total cost of the subproblem consisting of $t, \ldots, T$. The following double recursion can be used to calculate the dynamic programming variables.

$$K(T+1) := 0,$$

$$K(t) := \min_{1 \le \tau \le t} \{g_\tau + f_{\tau t}\} + K'(t), \quad 1 \le t \le T, \tag{9a}$$

$$K'(T+1) := 0,$$

$$K'(t) := \min_{t < \tau \le T+1} \{r_t d_{t,\tau-1} + K(\tau),\ r_t d_{t,\tau-1} + f_{t+1,\tau} + K'(\tau)\}, \quad 1 \le t \le T. \tag{9b}$$

Let us first explain recursion (9a). Given $K'(t)$ we only need to determine in which period $\tau \le t$ the first start-up is performed. If $\tau < t$, then the start-up is followed by set-ups in the periods τ to t. Of course, we choose τ such that the sum of the start-up cost and the set-up costs is minimal. Determining the optimal first start-up periods for all periods can be done in linear time. It suffices to show that these start-up periods may be assumed to form a non-decreasing sequence. To see this, suppose that s is the optimal start-up period for period $t-1$. Thus,

$$g_s + f_{s,t-1} = \min_{1 \le \tau \le t-1} \{g_\tau + f_{\tau,t-1}\}.$$

Adding f_t to both sides of the equation gives

$$g_s + f_{s,t} = \min_{1 \le \tau \le t-1} \{g_\tau + f_{\tau,t}\}.$$

Therefore, to determine the optimal start-up period for t, we only have to compare $g_t + f_t$ with $g_s + f_{st}$. If

$$g_t + f_t \le g_s + f_{st},$$

then t becomes the optimal start-up period, otherwise s is also the optimal start-up period for t.

Let us now consider recursion (9b). In this formula τ has the interpretation of the first production period after t. The choice to be made is, whether it is cheaper to start-up production after t, in which case $\min_{t<\tau\le T+1}\{r_t d_{t,\tau-1} + K(\tau)\}$ should be determined, or to perform set-ups in all periods between t and τ, in which case $\min_{t<\tau\le T+1}\{r_t d_{t,\tau-1} + f_{t+1,\tau} + K'(\tau)\}$ should be determined. The recursion formula for $K'(t)$ is written as the minimum of these two terms. To see that this is correct, note that if

$$K(\tau) = g_s + f_{s\tau} + K'(\tau)$$

for some $s \le t$, then the term $r_t d_{t,\tau-1} + K(\tau)$ can never (uniquely) determine the overall minimum in (9b) because for $s \le t$ it holds that

$$g_s + f_{s\tau} + K'(\tau) \ge f_{t+1,\tau} + K'(\tau).$$

Hence

$$r_t d_{t,\tau-1} + K(\tau) \ge r_t d_{t,\tau-1} + f_{t+1,\tau} + K'(\tau).$$

Since $K'(t)$ is the minimum over two terms, and because $\min_{t<\tau\le T+1}\{r_t d_{t,\tau-1} + f_{t+1,\tau} + K'(\tau)\}$ follows immediately from $\min_{t<\tau\le T+1}\{r_t d_{t,\tau-1} + f_{1\tau} + K'(\tau)\}$, evaluating (9b) can be done by using two lower envelopes: one of the points $(d_{\tau T}, K(\tau))$ and one of the points $(d_{\tau T}, f_{1\tau} + K'(\tau))$. It follows that $K'(t)$ can be determined for all $t \in \{1,\dots,T\}$ in $O(T \log T)$ time, and in $O(T)$ time if the production costs r_t are monotone. The following theorem is evident.

Theorem 13. *ELSS is solvable in* $O(T \log T)$ *time in general and in* $O(T)$ *time if there are no speculative motives to hold inventory.*

It should be noted here that the techniques used by Klawe [11] and Aggarwal and Park [1] can also be used to generate the results of Theorem 13, since dynamic program (9) belongs to the class discussed at the end of Section 2.

Finally, we note that it is possible to show that the model which combines backlogging and start-up costs is solvable in $O(T \log T)$ time (cf. Van Hoesel [17]). The conditions that are sufficient to let the algorithm run in linear time, are the same as those mentioned in Theorem 12.

4.3. ELS with generalized learning effects in set-ups

It is often the case that a repetitive task is performed more and more efficiently due to so-called learning effects. This was the motivation for Chand and Sethi [3] to study an economic lot-sizing model which captures those effects with respect to the set-ups. They assumed that in case of k set-ups the total set-up costs are equal to $\mathrm{sc}(k)$, with $\mathrm{sc}(k)$ a non-decreasing concave function in $k \in \{1,\dots,T\}$. Furthermore, the unit production costs were assumed to be stationary, i.e., $p_t = p$ for all $t = 1,\dots,T$. Chand and

Sethi proposed an $O(T^3)$ algorithm to solve this model. Using the concavity assumption, they also provided results that may be useful in speeding up this algorithm, although the worst case complexity is not affected. Exploiting the concavity assumption extensively, Malik and Wang [13] arrived at an $O(T^2)$ algorithm for the same problem. In Van Hoesel and Wagelmans [19] it is shown that the latter time bound can actually be achieved for the more general model in which the concavity assumption is dropped, and the assumption of stationary unit production costs is replaced by the assumption that speculative motives to hold inventory are not present.

In this subsection we present an algorithm for a model, to which we will refer as ELSL, that is more general than both models discussed above. In ELSL the set-up costs are assumed to be dependent on both the total number of set-ups and the periods in which the set-ups occur. In the case that the k-th set-up takes place in period t, we incur a non-negative set-up cost equal to $f_t(k)$. For all other costs we assume the usual structure. Under these assumptions it is easy to show that again the zero-inventory property holds. As in Section 3, we will use the unit production costs r_t and no explicit holding costs.

We can formulate ELSL as a dynamic program as follows. Let for $1 \le k \le t \le T$ the variable $M(t, k)$ denote the value of an optimal solution for the planning horizon $1,\dots,t$ if the number of set-ups is restricted to be exactly k. Clearly, the value of the optimal solution to the problem is $\min_{1\le k \le T}\{M(T, k)\}$. If we define $M(0, 0)$ to be 0, then the following recursion holds:

$$M(t, k) := \min_{k \le \tau \le t} \{M(\tau-1, k-1) + f_\tau(k) - r_\tau d_{1,\tau-1} + r_\tau d_{1t}\}, \quad 1 \le k \le t \le T. \tag{10}$$

The algorithm for solving ELSL is based on the forward algorithm of Section 3.1. However, the optimal production decisions can no longer be represented by a single (concave) envelope. This is due to the interdependency of the set-up costs, which implies that partial solutions for every number of set-ups need to be available (cf. (10)). Therefore, we keep track of (eventually) T lower envelopes, one for every value of k. Initially, we only construct the lower envelope for $k = 1$, i.e., we draw the line $f_1(1) + r_1 x$.

There are $T-1$ iterations, each corresponding to a value of $t \le T-1$. At the beginning of the t-th iteration we have determined a partial lower envelope for every $k \in \{1,\dots,t\}$. These envelopes are partial in the sense that they correspond to solutions with all production in the first t periods. For every $k \in \{1,\dots,t\}$ we can find the value of $M(t, k)$ by evaluating the envelope for k in x-coordinate d_{1t}. We add subsequently for every $k < t$ the line with slope r_{t+1} that passes through the point $(d_{1t}, M(t, k) + f_{t+1}(k+1))$ to the lower envelope for $k+1$. For $k = t$ such an envelope does not exist yet, so we create a new envelope consisting of only one line.

For a fixed k it can easily be checked that the total number of operations involving the corresponding envelope can be bounded by $O(T \log T)$. Moreover, a linear time bound is possible if we do not allow speculative motives. This implies the following result.

Theorem 14. *ELSL is solvable in* $O(T^2 \log T)$ *time in general and in* $O(T^2)$ *time if there are no speculative motives to hold inventory.*

Note that in the case of no speculative motives, the time bound is the best possible.

5. Proving structural properties

The geometric nature of the backward and forward algorithms is not only useful to get faster algorithms. The techniques also enable us to prove some structural properties of ELS in a very elegant way. By way of illustration two examples of such properties are given in this section. The first example is the planning horizon theorem of Wagner and Whitin, which is proved using the technique for the forward algorithm. The second example is a theorem on the structure of multiple optimal solutions, which is proved using the technique for the backward algorithm. In both examples we assume that there are no speculative motives to hold inventory.

Table 1
Data of Example 1

	t				
q	1	2	3	4	5
d_t	1	1	1	2	2
f_t	1	8	2	10	10
r_t	2	0	1	10	10
$s(t)$	1	1	1	3	2

Theorem 15 (Planning horizon theorem of Wagner and Whitin). *Suppose that there are no speculative motives to hold inventory. If s is the last production period in an optimal schedule for the planning horizon $1, \dots, t$, then there exists an optimal last production period for planning horizon $1, \dots, t+1$ that does not occur before s, i.e., the set $\{s, \dots, t+1\}$ contains a production period.*

Proof. Consider the lower concave envelope, after t iterations of the forward algorithm. The slopes of the line segments of g_t are in decreasing order because of the concavity of g_t. Since these slopes correspond to production costs of periods, the corresponding periods are in increasing order.

$F(t)$ is determined as the value of g_t at $x = d_{1t}$. Since s is the last production period in an optimal solution for the planning horizon $1, \dots, t$, the line segment of g_t at $x = d_{1t}$ has slope r_s. Now the line m_t is added which has slope $r_t \leq r_s$. To determine $F(t+1)$ we need to evaluate g_{t+1} at $x = d_{1,t+1}$, i.e., in a point equal to or to the right of d_{1t}. Due to the concavity of g_{t+1}, the slope of the line segment corresponding to $d_{1,t+1}$ is at most r_s, and thus by the monotonicity of the production costs the corresponding production period cannot precede s. □

In case that speculative motives are allowed, the following counter-example shows that Theorem 15 does not hold.

Example 1.

In Table 1, $s(t)$ denotes the (unique) optimal last production period for planning horizon $1, \dots, t$. It is left to the reader to verify the last row of the table. Note that $s(4) > s(5)$.

Another structural property concerns the relation between multiple optimal solutions.

Theorem 16. *Suppose that for an instance of ELS without speculative motives there exist at least two optimal production schedules. Let the production periods of one optimal schedule be $1 = t_1, \dots, t_K$. Then the second optimal schedule has a production period in each set $\{t_k, \dots, t_{k+1}\}$ for $k = 1, \dots, K-1$.*

Proof. Consider the final convex lower envelope g of the backward algorithm. If t is the last production period before t_k in the second schedule, then the place where its successor lies on g is the point, where the slopes change from greater than r_t to smaller than or equal to r_t. For t_k the same applies with

Table 2
Data of Example 2

	t			
	1	2	3	4
d_t	1	1	1	1
f_t	2	12	1	1
r_t	6	0	3	1

respect to its production cost r_{t_k}. However, its successor is t_{k+1}, and therefore the change of slope occurs at the point $(d_{t_{k+1},T}, B(t_{k+1}))$. By the ordering of the production costs we have $r_{t_k} \leq r_t$, and therefore the slope change for t occurs to the right of $(d_{t_{k+1},T}, B(t_{k+1}))$, and thus at a point belonging to a period smaller than or equal to t_{k+1}. By our assumption that t was the last production period before t_k, this proves the theorem. □

Again, the assumption that there are no speculative motives to hold inventory is crucial for the validity of the theorem, as is shown in the following example.

Example 2.
It is left to the reader to verify that the following schedules are both optimal (with costs 20):
Schedule 1 with production periods 1, 3 and 4.
Schedule 2 with production periods 1 and 2.
Clearly, these schedules do not satisfy the structural property of Theorem 16.

6. Computational experiments

Some limited testing has been carried out to get an indication of the performance of the algorithms for ELS, which were developed in Section 3. We compared these algorithms to the other algorithms that are known from the literature. From the complexity results in Section 3, it is obvious that a distinction has to be made between instances without speculative motives to hold inventory and general instances.

6.1. *Algorithms for instances without speculative motives*

The following algorithms have been tested:
BL: the linear time backward dynamic programming algorithm;
FL: the linear time forward dynamic programming algorithm;
K: the recursive matrix searching algorithm developed by Klawe;
WW1: the original dynamic programming algorithm of Wagner and Whitin, which uses the planning horizon theorem.

The instances on which the various programs were run, have been generated as follows. The set-up, holding and production costs were taken equal to respectively 450, 5 and 0. The demands were uniformly drawn between 0 and 10. These choices result in an average production run of six periods.

The algorithms have been implemented in Turbo Pascal, and were run on an Olivetti M280 PC without co-processor. The results (running-time in seconds) are summarized in Table 3.

The performance of all algorithms shows a linear relationship with the length of the planning horizon. This is not surprising for the algorithms BL, FL and K, since these are linear time algorithms. Also for algorithm WW1 this is not surprising, since in every iteration it only considers the periods from the last production period on. As already mentioned above, the average production run is equal to six periods. Because this is a small number compared to the length of the planning horizons, the maximal number of

Table 3
Running-time (sec) of the algorithms for instances without speculative motives

T	BL	FL	K	WW1
500	0.2	0.3	0.5	0.2
1000	0.4	0.6	1.0	0.4
2000	0.8	1.2	2.0	0.9
4000	1.5	2.5	4.2	1.8
8000	3.0	5.0	8.5	3.6

Table 4
Effect of variations in the set-up costs

f	run length	BL	FL	K	WW1
50	2	1.5	2.5	2.6	0.9
200	4	1.5	2.5	3.8	1.3
450	6	1.5	2.5	4.2	1.8
800	8	1.5	2.5	4.4	2.2
1250	10	1.5	2.5	4.5	2.6
5000	20	1.5	2.5	4.5	4.8
20000	40	1.5	2.5	5.2	9.1

periods considered in an iteration can be viewed as a constant. However, this seems to imply that especially algorithm WW1 is sensitive to variations in the average length of the production runs. We have investigated this and the results are presented in Table 4. The variation was created by varying the set-up costs. The instances on which the algorithms were tested had production costs equal to 0, holding costs equal to 5 and demand fixed to 5 units each period. The planning horizon was fixed at 4000 periods.

Although an effect on algorithm WW1 is according to our expectations, it is somewhat surprising that the matrix searching algorithm K is also affected by variations in the set-up costs. However, this effect is much less than the effect on algorithm WW1.

6.2. *Algorithms for instances with general production costs*

The following algorithms have been tested:
B: the backward dynamic programming algorithm;
F: the forward dynamic programming algorithm;
AP: the recursive matrix searching algorithm developed by Aggarwal and Park;
WW2: the original dynamic programming algorithm of Wagner and Whitin (without the planning horizon theorem).

The instances on which the various programs were run have been generated as follows. All parameters were generated uniformly, within prespecified non-negative bounds, and independently of each other. Demand varied between 1 and 10 units, set-up costs between 100 and 500, and both production and holding costs ranged from 1 to 5.

The results here are much more indicative on which choice is preferable (see Table 5). The bad performance of algorithm WW2 is no surprise, because it is a quadratic algorithm. However, the results of algorithm AP are rather disappointing. This may be due to the complicated data manipulations that have been used to implement algorithm AP. A similar effect, but less dramatic, shows up when algorithm F is compared with algorithm B. Note that algorithm F uses 2-3 trees to maintain the lower envelope of lines, whereas B is implemented with a stack. This points out another advantage of algorithm B, i.e., the simplicity with which it can be implemented. The only non-trivial part concerns a binary search.

Table 5
Running-time (sec) of the algorithms for general instances

T	B	F	AP	WW2
500	0.2	0.8	1.8	9.1
1000	0.5	1.8	4.8	36.8
2000	1.0	3.8	14.1	141.0
4000	2.1	7.9	45.0	563.8
8000	4.4	16.9	151.9	–

7. Summary and concluding remarks

We have presented two basic geometric techniques that can be used to solve certain types of dynamic programs. We showed that the economic lot-sizing problem can be solved by applying these basic techniques straightforwardly. Solving the lot-sizing models with backlogging, start-up costs and learning effects in set-up costs involved applying the same basic techniques to more complex recursion formula. The reformulations carried out in Section 4 may be helpful to the reader in recognizing whether a certain dynamic programming problem can be solved using the geometric techniques.

We also showed that the geometric techniques can be powerful tools in proving structural properties of ELS. Such properties may be useful in designing algorithms for related problems. For instance, the property stated in Theorem 16 is used in Van Hoesel and Wagelmans [19] in order to design a linear time algorithm to determine the maximal amount by which all set-up costs can be increased simultaneously, without affecting the optimality of the current optimal solution for the economic lot-sizing model without speculative motives. (This result immediately implies a parametric approach to solve the model with learning effects in the set-ups costs, in which these costs only depend on the number of set-ups and not on the actual periods in which the set-ups take place. However, as we already mentioned before, the algorithm presented in Section 4.3. has the same complexity and can be applied to a more general model.) The geometric techniques are also quite useful as a basis for studying other issues of sensitivity analysis (cf. Van Hoesel and Wagelmans [18]).

Our computational experiments indicate that especially the backward dynamic programming algorithm performs very well in comparison to the other algorithms (although forward algorithms may be favored in a rolling horizon environment). Of course, this can be explained by the very simple data structure needed to implement that particular algorithm. Furthermore, on the basis of the computational results, we would favor the geometric techniques over the more general techniques developed in [1] and [11].

Finally, we would like to point out that the geometric techniques have also been applied to other problems that are closely related to ELS. We refer to the paper by Kuik et al. [12] on the discrete lot-sizing and scheduling problem, and the paper by Hassin and Tamir [8] on location problems on the real line. Given the quite general form of (1) and (2), we expect that other interesting application areas exist.

Acknowledgements

We thank Refael Hassin and Arie Tamir for pointing out to us the applicability of the geometric method to solving the economic lot-sizing problem with backlogging. We are also thankful to two anonymous referees for their detailed comments on earlier drafts of this paper.

References

[1] Aggarwal, A., and Park, J.K., "Improved algorithms for economic lot-size problems", *Operations Research* 41 (1993) 549–571.
[2] Aho, A.V., Hopcroft, J.E., and Ullman, J.D., *Data Structures and Algorithms*, Addison-Wesley, London, 1983.
[3] Chand, S., and Sethi, S.P., "A dynamic lot sizing model with learning in setups", *Operations Research* 38 (1990) 644–655.
[4] Denardo, E.V., *Dynamic Programming: Models and Applications*, Prentice-Hall, Englewood Cliffs, NY, 1982.
[5] Evans, J.R., "An efficient implementation of the Wagner–Whitin algorithm for dynamic lot-sizing", *Journal of Operations Management* 5 (1985) 229–235.
[6] Federgruen, A., and Tzur, M., "The dynamic lot sizing model with backlogging: A simple O(n log n) algorithm", Working Paper, Graduate School of Business, Columbia University, New York, 1990.
[7] Federgruen, A., and Tzur, M., "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 (1991) 909–925.
[8] Hassin, R., and Tamir, A., "Improved complexity bounds for location problems on the real line", *Operations Research Letters* 10 (1991) 395–402.

[9] Karmarkar, U.S., Kekre, S., and Kekre, S., "The dynamic lot-sizing problem with startup and reservation costs", *Operations Research* 35 (1987) 389–398.

[10] Karmarkar, U.S., and Schrage, L., "The deterministic dynamic cycling problem", *Operations Research* 33 (1985) 326–345.

[11] Klawe, M.M., "A simple linear time algorithm for concave one-dimensional dynamic programming", Technical Report 89-16, University of British Columbia, Vancouver, Canada, 1989.

[12] Kuik, R., Salomon, M., Van Hoesel, S., and Van Wassenhove, L.N., "The single item discrete lot sizing and scheduling problem: Linear description and optimization", Management Report Series No. 53, Erasmus University Rotterdam, Rotterdam, Netherlands, 1989.

[13] Malik, K., and Wang, Y., "An improved algorithm for dynamic lot sizing problem with learning effect in setups", Working Paper 90-08, Johnson Graduate School of Management, Cornell University, Ithaca, NY, 1990.

[14] Preparata, F.P., and Shamos, M.I., *Computational Geometry – An Introduction*, Springer-Verlag, New York, 1985.

[15] Saydam, C., and McKnew, M., "A fast microcomputer program for ordering using the Wagner–Whitin algorithm", *Production and Inventory Management Journal* 28 (1987) 15–19.

[16] Schrage, L., "The multiproduct lot scheduling problem", in: M.A.H. Dempster, J.K. Lenstra and A.H.G. Rinnooy Kan (eds.), *Deterministic and Stochastic Scheduling (Nato Advanced Study Institutes Series)*, Reidel, Dordrecht, 1982.

[17] Van Hoesel, C.P.M., "Models and algorithms for single-item lot sizing problems", Ph.D. Dissertation, Erasmus University Rotterdam, Rotterdam, Netherlands, 1991.

[18] Van Hoesel, C.P.M., and Wagelmans, A.P.M., "Sensitivity analysis of the economic lot-sizing problem", *Discrete Applied Mathematics*, 45 (1993) 291–312.

[19] Van Hoesel, C.P.M., and Wagelmans, A.P.M., "On setup cost reduction in the economic lot-sizing model without speculative motives", Working Paper ORC 256-91, Operations Research Center, Massachusetts Institute of Technology, Cambridge, MA, 1991.

[20] Van Wassenhove, L.N., and Vanderhenst, P., "Planning production in a bottleneck department", *European Journal of Operational Research* 12 (1983) 127–137.

[21] Wagelmans, A.P.M., Van Hoesel, C.P.M., and Kolen, A.W.J., "Economic lot-sizing: An O(n log n) algorithm that runs in linear time in the Wagner–Whitin case", *Operations Research* 40, *Supplement* 1 (1992) S145–S156.

[22] Wagner, H.M., and Whitin, T.M., "A dynamic version of the economic lot size model", *Management Science* 5 (1958) 89–96.

[23] Yao, F. "Speed-up in dynamic programming", *SIAM Journal on Algebraic and Discrete Methods* 3 (1982) 532–540.

[24] Zangwill, W.I., "A backlogging and a multi-echelon model of a dynamic economic lot-sizing production system: A network approach", *Management Science* 15 (1969) 506–527.