CORE DISCUSSION PAPER
2003/7

ALGORITHMS FOR SINGLE ITEM CONSTANT CAPACITY LOTSIZING PROBLEMS

Mathieu VAN VYVE[1]

January 2003

Abstract

The main result of this paper is to provide an $O(n^3)$ algorithm for the single item constant capacity lotsizing problem with backlogging and a general number of installable batches, i.e in each time period t we may install up to m_t multiples of the batch capacity, where the m_t are given and are time-dependent. This generalizes earlier results [12, 16] as we consider backlogging and a general number of installable batches.

We also give faster algorithms for three special cases of this general problem. When backlogging is not allowed and the costs satisfy the Wagner-Whitin property the problem is solvable in $O(n^2 \log n)$ time. In the discrete case it is possible to solve the problem with and without backlogging in $O(n^2)$ and $O(n \log n)$ time respectively.

Keywords: lotsizing, complexity, constant capacity, backlogging.

[1]Center of Operations Research and Econometrics (CORE), Université catholique de Louvain. 34, Voie du Roman Pays. 1348 Louvain-la-Neuve, Belgium. This text presents research results of the Belgian Program on Interuniversity Poles of Attraction initiated by the Belgian State, Prime Minister's Office, Science Policy Programming. The research was carried out with financial support of the Growth Project G1RD-1999-00034 (LISCOS) of the European Community. The scientific responsibility is assumed by the author.

1 Introduction

In the single item lotsizing problem there is a demand for a single item in n consecutive time periods. The demand in period t may be satisfied from production in period t or from stock at the end of period $t-1$. If backlogging is permitted, it may also be satisfied from backlog in period t. The production in each time period is limited by the installed capacity, which is a multiple of the batch capacity or simply the capacity. The maximum number of batches that can be installed in each period is given and is time-dependent. The *installable capacity in period t* is then the capacity that can be installed in period t. A problem is uncapacitated (resp. installation uncapacitated) if the capacity (resp. installable capacity) in any time period t is sufficient to allow production in that period to satisfy all demand that can be satisfied from t and capacitated (resp. installation capacitated) otherwise. A problem is called discrete if the production is equal to the installed capacity. The cost associated with a production plan is divided between the production cost, the holding cost and the backlogging cost in each period. There is also a cost associated with installing capacity.

Most of the recent literature considers models with cost functions (including the capacity costs) that are assumed to be proportional, leading to mixed integer linear programs. Several authors show that the uncapacitated version of the model described above is solvable in $O(n \log n)$ time for general linear cost functions and $O(n)$ time if the costs satisfy the so called Wagner-Whitin property [1, 4, 18, 17].

For capacitated problems Florian et al. [8] and Bitran and Yanasse [2] have shown that the general problem without backlogging is NP-Hard, even under various assumptions on the costs or when the problem is discrete. Several authors have considered the case when the capacity is constant over time and backlogging is not allowed. Pochet and Wolsey [12] showed that the installation uncapacitated case is solvable in $O(n^3)$ time while van Hoesel and Wagelmans [16] give a $O(n^3)$ algorithm for the installation capacitated case when only one batch can be installed in each period.

The literature on the constant capacity discrete lotsizing problem mostly focusses on the problems with startup times or costs (e.g. Fleischmann [5, 6], van Hoesel et al. [15], van Eijl [13], van Eijl and van Hoesel [14]). Lasdon and Terjung [9] consider backlogging and setup times. Recently Miller and Wolsey [10] studied formulations for the discrete lotsizing problem with backlogging and/or initial stock. In particular, they provide integral formulations of polynomial size for the single item case with either backlogging or initial stock, but not both, thereby showing that these two problems are solvable in polynomial time.

This paper is organized as follows. We begin by reviewing some standard complexity results that we use in this paper in Section 2. The notation is also introduced. The rest of the paper is divided in two main sections. Section 3 deals with problems without backlogging, while problems involving backlogging are treated in Section 4. Each of these two sections is similarly divided into three subsections. The first one introduces building blocks useful for the next two subsections. The second and the third study the discrete case and the general case respectively. Finally, we discuss the results obtained and open questions.

2 Preliminaries

In this section we introduce the notation and present some useful known results.

Let n be the length of the planning horizon. We normalize the batch capacity to 1, so that it will never appear in the models that we will consider. For each period $t \in \{1, \ldots, n\}$ the following data is given:

- d_t demand in t,
- m_t maximum number of installable batches in period t,
- p_t unit production cost in t,
- f_t unit set-up cost in t,
- h_t unit holding cost in t (defined also for $t = 0$),
- g_t unit backlogging cost in t (defined also for $t = 0$).

We suppose that all data is rational and that m_t are nonnegative integers. We also define the unit vector $e_t = \{0, 0, \ldots, 0, 1, 0, \ldots, 0\}$ where the one is in position t. For ease of presentation, we will often index d, m or e by intervals or (multi-)sets. This denotes the *sum* over the indexing interval or set. For example, $d_{kl} = \sum_{t=k}^{l} d_t$ and $e_S = \sum_{t \in S} e_t$ where $S \subseteq \{1, \ldots, n\}$. Note that S may be a multiset, i.e. it may contain the same element several times. Finally $(x)^+$ denotes $\max(x, 0)$.

We model the different problems as mixed integer programs using the following variables defined for each period $t \in \{1, \ldots, n\}$:

- x_t production in t,
- y_t number of batches that are installed in t,
- s_t stock at the end of t (defined also for $t = 0$),
- r_t backlog in t from later periods (defined also for $t = 0$).

This leads to the following formulation of the constant capacity lotsizing problem with backlogging and a general number of installable batches (denoted by LSB):

$$\min \sum_{t=1}^{n} f_t y_t + \sum_{t=1}^{n} p_t x_t + \sum_{t=0}^{n} h_t s_t + \sum_{t=0}^{n} g_t r_t, \tag{1}$$

$$s_{t-1} + r_t + x_t = d_t + r_{t-1} + s_t \qquad 1 \le t \le n, \tag{2}$$

$$(LSB) \qquad x_t \le y_t \qquad 1 \le t \le n, \tag{3}$$

$$s, r, x \ge 0, \tag{4}$$

$$y_t \in \{0, 1, \ldots, m_t\} \qquad 1 \le t \le n. \tag{5}$$

This is the problem considered in Section 4.3. All other problems will be defined as special cases. The following results are well known and will be used later.

Proposition 1 *$(x + a + b)^+ - (x + a)^+ \ge (x + b)^+ - (x)^+$ for any $a, b \ge 0$ and $x \in R$.*

Proof. It follows from the convexity of $x \mapsto (x)^+$. ∎

Proposition 2 *If a function f defined on the interval of integers $[a, b]$ is non-decreasing and can be evaluated in constant time, then, for any y, one can find the values*

$$\min_{a \le x \le b} \{x \in Z : f(x) \ge y\} \text{ and } \max_{a \le x \le b} \{x \in Z : f(x) \le y\},$$

and the corresponding values of x in $O(\log(b - a))$ time by binary search. ∎

Proposition 3 *[19] Let the cost ϕ_{ab} of the interval $[a, b]$ be given for all integers a and b satisfying $1 \leq a \leq b \leq n$. The problem of partitioning the interval of integers $[1, n]$ into subintervals with minimum cost is solvable in $O(n^2)$ time by dynamic programming.* ■

In the rest of this paper, we will denote by PARTITION-INT (ϕ) the function which computes the cost of this optimal partition.

Let $M = (N, \mathcal{F})$ be a polymatroid where $\mathcal{F}$ is the set of independant multisets on the ground set N, and let $f \in R^{|N|}$ be an objective function. Let $y(p)$ denote the lexicographically earliest optimal solution of the following associated optimization problem.

$$\begin{aligned} &\max \sum_{i \in N} f_i y_i, \\ &\sum_{i \in N} y_i = p, \\ &S(y) \in \mathcal{F}, \\ &y \in Z_+^{|N|}, \end{aligned}$$

where $S(y)$ denotes the multiset containing the element $i \in N$ y_i times. The next proposition essentially follows from the correctness of the (reverse) greedy algorithm when applied to polymatroids.

Proposition 4 *[11] $S(y(p)) \subset S(y(q))$ (i.e. $y(p) < y(q)$) for any $p < q$.* ■

As a corollary, to obtain $y(p)$ from $y(q)$, it suffices to remove from $y(p)$ the $q - p$ most expensive elements. When describing algorithms, we will denote this operation by $y(p) \leftarrow$ GREEDY-DECREASE $(y(q), q - p)$.

3 Problems without Backlogging

3.1 Building Blocks

Consider the following problem $Q(j)$ for $1 \leq j \leq n$:

$$\max \textstyle\sum_{i=1}^{j} f_i y_i, \tag{6}$$

$$\textstyle\sum_{i=1}^{t} y_i \leq \delta_t \qquad 1 \leq t \leq j-1, \tag{7}$$

$$y_t \in \{0, 1, \ldots, m_t\} \quad 1 \leq t \leq j, \tag{8}$$

where $\delta \in Z_+^n$. Let $\delta_0 = 0$ and $\eta_j = \min_{0 \leq t \leq j} [\delta_t + m_{t+1,j}] = \min(\delta_j, \eta_{j-1} + m_j)$ for all $1 \leq j \leq n$.

Proposition 5 *All solutions of (7)–(8) having maximal cardinality satisfy*

$$\sum_{i=1}^{j} y_i = \eta_{j-1} + m_j. \tag{9}$$

Proof. Let $k = \arg\min_{0 \le t \le j-1} [\delta_t + m_{t+1,j}]$. If $\sum_{i=1}^{j} y_i > \min_{0 \le t \le j-1} [\delta_t + m_{t+1,j}]$, then y does not satisfy (7) for $t = k$ or, if $k = 0$, y does not satisfy (8). So by contradiction suppose that $\sum_{i=1}^{j} y_i \le \min_{0 \le t \le j-1} [\delta_t + m_{t+1,j}] - 1$. There exists a period t which satisfies $y_t < m_t$. Let l be the last such period. Then $y' = y + e_l$ is feasible in (7)–(8) and contradicts the maximality of y. ■

Corollary 6 *The set of feasible solutions of (7)–(8) forms a polymatroid.* ■

We denote by $Q(j, p)$ the problem $Q(j)$ with the cardinality constraint $\sum_{i=1}^{j} y_i = p$. We also define Q as the collection of problems $\{Q(j, \beta_j)\}_{j=1}^{n}$ where $\beta_j \ge \delta_j$ for each $1 \le j \le n$. Problem Q is a useful building block for solving later problems.

The output of the following algorithm is a n-dimensional vector ϕ, where ϕ_j is the value of the optimal solution of $Q(j, \beta_j)$. The first seven lines constitute the initialization.

SOLVE-Q(n, β, δ, f, m)

```
 1  η_0 ← 0
 2  for j ← 1 to n
 3      do η_j ← min(δ_j, η_{j-1} + m_j)
 4  if β_1 > m_1
 5     then φ_1 ← ∞
 6     else  φ_1 ← β_1 f_1
 7  y ← (η_1, 0, ..., 0)
 8  for j ← 2 to n
 9      do y_j ← m_j
10          if m_j + η_{j-1} < β_j
11             then φ_j ← ∞
12                  y ← GREEDY-DECREASE(y, (m_j + η_{j-1} − η_j)^+)
13             else  y ← GREEDY-DECREASE(y, m_j + η_{j-1} − β_j)
14                  φ_j ← Σ_{i=1}^{j} f_i y_i
15                  y ← GREEDY-DECREASE(y, β_j − η_j)
16  return φ
```

Example 1 *Consider the following instance of Q with $n = 6$.*

Period	1	2	3	4	5	6
f	15	15	27	19	16	42
m	2	2	1	3	4	1
δ	1	3	3	5	7	8
β	2	4	4	6	8	9

The maximal solutions have cardinality $\eta = [1, 3, 3, 5, 7, 8]$. The following tables show how the vector y is modified during the course of SOLVE-Q. *The second to last column gives the line number of Algorithm* SOLVE-Q *which lead to the new values. The cost ϕ_j is indicated in the last column.*

Iteration	y^1	y^2	y^3	y^4	y^5	y^6	*Line*	*Cost*
Init.	1							15
2	1	2					2	
	1	2					5	∞
3	1	2	1				2	
	1	2	1				7	72
	1	2	0				9	
4	1	2	0	3			2	
	1	2	0	3			7	102
	1	2	0	2			9	
5	1	2	0	2	4		2	
	1	2	0	1	4		7	128
	1	2	0	0	4		9	
6	1	2	0	0	4	1	2	
	1	2	0	0	4	1	5	∞

Proposition 7 *Algorithm* SOLVE-Q *is correct.*

Proof. We prove the following invariant at the beginning of the loop of line 8: y is an optimal solution of $Q(j-1, \eta_{j-1})$. The execution of lines 1 to 7 ensures that it is true for $j = 2$. So suppose it is true at the beginning of the loop for any j. Since $\sum_{i=1}^{j-1} y_i \leq \eta_{j-1}$ and $y_j \leq m_j$, y is an optimal solution of $Q(j, \eta_{j-1}+m_j)$ after line 9. Also, this implies that $Q(j, \beta_j)$ is infeasible if $\delta_{j-1} + m_j < \beta_j$ (Line 10). In this case, ϕ_j is assigned an infinite value and, by Proposition 4, the loop invariant is restored in Line 12. Otherwise, by Proposition 4 and Corollary 6, GREEDY-DECREASE $(y, \eta_{j-1} + m_j - \beta_j)$ outputs an optimal solution of $Q(j, \beta_j)$. Finally, the loop invariant is restored in Line 15. ■

We now discuss an efficient implemention of SOLVE-Q. We maintain a priority queue F, see, e.q. [3], wich contains the element t if $y_t > 0$. The cost f_t is the key to the element t in F. The presence of F makes it possible to find the less profitable period t with $y_t > 0$ in $O(1)$ time. In Line 9, INSERT (F, j) is performed to maintain F. The following implementation of GREEDY-DECREASE (y, p) guarantees that the priority queue F is maintained.

GREEDY-DECREASE(y, p)
1 $t \leftarrow$ MIN(F)
2 $y_t \leftarrow (y_t - p)^+$
3 **if** $y_t = 0$
4 **then** EXTRACT-MIN(F)
5 **if** $p > y_t$
6 **then** GREEDY-DECREASE$(y, p - y_t)$

Proposition 8 *Algorithm* SOLVE-Q *can be implemented in* $O(n \log n)$ *time.*

Proof. Observe that each period t is inserted in F only once, namely in Line 9. The first consequence is that the operation INSERT (F, j) is $O(n \log n)$ overall, as one call to INSERT (F, j) is $O(\log n)$. The second consequence is that each period t can be extracted from F only once also. Therefore, there are at most n calls to EXTRACT-MIN (F) overall, each one being $O(\log n)$. Note also that each time GREEDY-DECREASE is executed, if t is not removed from F (i.e. there is no call

to EXTRACT-MIN (F)), then there is no recursive call to GREEDY-DECREASE. It follows that there can be at most $2n$ calls to GREEDY-DECREASE overall, and GREEDY-DECREASE is thus $O(n \log n)$ overall.

Finally, it is easy to maintain the optimal cost associated to the solution y during the algorithm in $O(n)$ time, since there are at most $O(n)$ calls to GREEDY-DECREASE and EXTRACT-MIN. Line 14 is thus $O(n)$ overall. ■

3.2 Discrete Lotsizing without Backlogging

The discrete lotizing problem DLS is the special case of LSB where $x_t = y_t$ and $r_t = 0$ for all periods t, and $s_0 = r_0 = 0$. Because of the 2n equality constraints (2) and $x_t = y_t$, we may assume without loss of generality that $h_t = p_t = 0$ for all t. Using the same relations, we rewrite DLS as follows.

$$(DLS) \qquad \min \sum_{i=1}^{n} f_i y_i, \tag{10}$$

$$\sum_{i=1}^{t} y_i \geq \lceil d_{1t} \rceil \qquad 1 \leq t \leq n, \tag{11}$$

$$y_t \in \{0, 1, \ldots, m_t\} \qquad 1 \leq t \leq n. \tag{12}$$

We solve DLS by reducing it to an instance of Q. We may assume $f_t > 0$ for all t. Otherwise, we can set $m_t = 0$, replace d_t by $d_t - m_t$ and add $f_t m_t$ to the objective. Complementing $\overline{y}_i = m_i - y_i$ for all i, we obtain an equivalent packing formulation of DLS:

$$\sum_{i=1}^{n} f_i m_i - \qquad \max \sum_{i=1}^{n} f_i \overline{y}_i, \tag{13}$$

$$\sum_{i=1}^{t} \overline{y}_i \leq m_{1t} - \lceil d_{1t} \rceil \qquad 1 \leq t \leq n, \tag{14}$$

$$\overline{y}_t \in \{0, 1, \ldots, m_t\} \qquad 1 \leq t \leq n. \tag{15}$$

Proposition 9 *All optimal solutions of (13)–(15) satisfy*

$$\sum_{i=1}^{n} \overline{y}_i = \min_{0 \leq t \leq n} \left(m_{1n} - \lceil d_{1t} \rceil \right). \tag{16}$$

Proof. As $f_t > 0$ for all t, optimal solutions are maximal. A similar argument to that of Proposition 5 yields the desired result. ■

Therefore, replacing (14) for $t = n$ by (16) does not change the set of optimal solutions. We obtain an instance of Q by letting $\beta_j = \min_{0 \leq t \leq n} \left(m_{1n} - \lceil d_{1t} \rceil \right)$ for all j and $\delta_j = m_{1j} - d_{1j}$. Denoting by ϕ the ouput of Algoritm SOLVE-Q executed on this instance, the optimal cost of (13)–(15) is ϕ_1. We therefore have the following result.

Proposition 10 *DLS can be solved in $O(n \log n)$ time.* ■

3.3 Constant Capacity Lotsizing with Wagner-Whitin Costs

The constant capacity lotsizing problem with Wagner-Whitin (WW) costs is a special case of LSB in which $r_t = 0$ and $p_t + h_t \geq p_{t+1}$ for all t. Because of the n equality constraints (2), we may assume without loss of generality that $p_t = 0$ for all t. For notational convenience, we also suppose that $s_0 = 0$. Thus, we obtain the following formulation.

$$\min \sum_{t=1}^{n} f_t y_t + \sum_{t=1}^{n} h_t s_t, \tag{17}$$

$$\sum_{i=1}^{t} x_i = d_{1t} + s_t \qquad 1 \leq t \leq n, \tag{18}$$

(WW)

$$x_t \leq y_t \tag{19}$$

$$s, x \geq 0 \tag{20}$$

$$y_t \in \{0, 1, \ldots, m_t\}. \tag{21}$$

Furthermore, without loss of generality, we may assume that $f_t \geq 0$. Otherwise, we can add a constant term $f_t m_t$ to the objective and work with a zero capacity cost in t.

Following the approach of Zangwill [20], we characterize extreme optimal solutions. Given an extreme solution (x, y, s), we call t a *regeneration period* if $s_t = 0$ and $[a, b]$ a *regeneration interval* if $a - 1$ and b are regeneration periods with no regeneration periods in between. Because we are considering extreme solutions, there can only be at most one period j in $[a, b]$ with $0 < x_j < y_j$ which we call the *fractional period* of this regeneration interval. Let $\rho_{ab} = d_{ab} - (\lceil d_{ab} \rceil - 1)$.

From $f_t \geq 0$, it follows that there exists an extreme optimal solution such that

$$x_t = \begin{cases} y_t - 1 + \rho_{ab} & \text{if } t \text{ is the unique fractional period of the regeneration} \\ & \text{interval } [a, b], \\ y_t & \text{for all periods } t \text{ which are not fractional periods.} \end{cases}$$

Up to now, we have not used the Wagner-Whitin property. The next proposition shows why this property is interesting.

Proposition 11 *With Wagner-Whitin costs, there exists an extreme optimal solution of (17)–(21) such that in each regeneration interval $[a, b]$, the fractional period is a.*

Proof. Supppose by contradiction that for each extreme optimal solution $\theta = (x, y, s)$ there exists an interval $[a_\theta, b_\theta]$ and a period $a_\theta < j_\theta \leq b_\theta$ for which $0 < x_{j_\theta} < y_{j_\theta}$. Let us fix $\theta = (x^*, y^*, s^*)$ as the the lexicographically latest optimal extreme solution. Observe first that $y^*_{a_\theta}$ cannot be zero. Otherwise either the demand d_{a_θ} could not be satisfied or in the case of zero demand we would have $s^*_{a_\theta} = 0$. Let $\epsilon = \min[x_{a_\theta}, \min_{t=a_\theta,\ldots,j_\theta-1} s^*_t] > 0$. Shifting ϵ units of production from a_θ to j_θ yields a lexicographically better extreme solution because of the Wagner-Whitin property. This contradicts the lex-optimality of the solution. ■

Corollary 12 *With Wagner-Whitin costs, there exists an extreme optimal solution of (17)–(21) such that in each regeneration interval $[a,b]$, $s_{t-1} = d_{t,b} - \sum_{i=t}^{b} y_i > 0$ for $a < t \leq b$.* ■

We now fix a and b, and suppose that $[a,b]$ is a regeneration interval. We want to solve the problem restricted to this interval $[a,b]$; call this problem $INT(a,b)$. Using Corollary 12, we eliminate stock by substitution and write $INT(a,b)$ as

$$\phi(a,b) = \min \sum_{i=a}^{b} (f_i - \sum_{j=a}^{i-1} h_j) y_i + \sum_{i=a}^{b-1} h_i d_{i+1,b}, \tag{22}$$

$$\sum_{i=a}^{b} y_i = \lceil d_{ab} \rceil, \tag{23}$$

$(INT(a,b))$

$$\sum_{i=t}^{b} y_i \leq \lceil d_{tb} \rceil - 1 \qquad a < t \leq b, \tag{24}$$

$$y_t \in \{0,1,\ldots,m_t\} \qquad a \leq t \leq b. \tag{25}$$

Constraint (23) ensures that exactly the minimum capacity to produce d_{ab} is installed. Constraint (24) guarantees that $s_{t-1} > 0$ for $0 < t \leq b$.

Using the equality constraint (23), we can add $\sum_{j=a}^{b-1} h_j \left(\sum_{i=a}^{b} y_i - \lceil d_{ab} \rceil \right)$ to the objective and work with the following equivalent expression.

$$\sum_{i=a}^{b} (f_i + \sum_{j=i}^{b-1} h_j) y_i + \sum_{i=a}^{b-1} h_i d_{i+1,b} - \sum_{j=a}^{b-1} h_j \lceil d_{ab} \rceil \tag{26}$$

The coefficient of y_i is now independent of a. Hence $\{INT(a,b)\}_{a=1}^{b}$ is an instance of Q, for fixed b. This suggests the following algorithm for solving WW.

SOLVE-WW(n,h,f,m,d)
1 **for** $t \leftarrow 1$ **to** $n, b \leftarrow t$ **to** n
2 **do** $f_t(b) \leftarrow f_t + \sum_{j=t}^{b-1} h_j$
3 $\delta_t(b) \leftarrow \lceil d_{tb} \rceil - 1$
4 $\beta_t(b) \leftarrow \lceil d_{tb} \rceil$
5 **for** $b \leftarrow 1$ **to** n
6 **do** $\phi(b) \leftarrow$ SOLVE-Q$(b-a+1, \beta(b), \delta(b), f(b), m)$
7 **for** $a \leftarrow 1$ **to** $n, b \leftarrow a$ **to** n
8 **do** $\phi_a(b) \leftarrow \phi_a(b) + \sum_{i=a}^{b-1} h_i d_{i+1b} - \sum_{j=a}^{b-1} h_j \lceil d_{ab} \rceil$
9 **return** PARTITION-INT(ϕ).

Example 2 *Consider the following instance of WW with $n = 6$.*

Period	1	2	3	4	5	6
m	1	4	3	1	2	2
f	31	8	13	22	12	15
h	3	2	1	2	3	
d	0.6	1.9	2.4	0.3	1.8	1.5

We show how to compute the values $\phi_a(6)$ *for* $1 \leq a \leq 6$. *The first step is to compute the coefficient of* y *appearing in the modified objective (26). We obtain* $f(b) = (42, 16, 19, 27, 15, 15)$. *This yields an instance of* Q *identical to that of Example 1. We thus get the same objective values* $\phi(6) = (\infty, 128, 102, 72, \infty, 30)$. *To compute the true costs* $\phi_a(6)$, *we add the constant terms of (26). This is illustrated in the following table.*

Period a	1	2	3	4	5	6
	∞	128	102	72	∞	30
$\sum_{i=a}^{b-1} h_i d_{i+1,b}$	50.4	26.7	14.7	11.1	4.5	0
$\sum_{i=a}^{b-1} h_i \lceil d_{ab} \rceil$	99	64	36	20	12	0
$\phi(a, 6)$	∞	90.7	80.7	63.1	∞	30

Observe that $\phi_1(6)$ *is assigned an infinite cost even though there is enough capacity in periods* 1 *to* t *to satisfy the demand of periods* 1 *to* t *for any* t. *The reason is that it is not possible to produce the fractional batch of* 0.5 *and satisfy the demand in period* 1 *of* 0.6. *Hence* $[1, 6]$ *cannot be a regeneration interval with Wagner-Whitin costs.*

The solution output by Algorithm Solve-WW *for* $INT(2, 6)$ *is* $(y_2, \ldots, y_6) = (4, 1, 0, 2, 1)$ *and* $(s_2, \ldots, s_5) = (2, 0.6, 0.3, 0.5)$. *We compute* $\phi(2, 6) =$ *production costs + holding costs* $= (4{*}8{+}13{+}2{*}12{+}15){+}(2{*}2{+}1{*}0.6{+}2{*}0.3{+}3{*}0.5) = 84 + 6.7 = 90.7$ *which is identical to the value found in the table.*

Proposition 13 *The algorithm* Solve-WW *is correct and runs in* $O(n^2 \log n)$ *time.*

Proof. It follows from previous discussion that, after Line 2, $\phi(a, b)$ is the optimal cost of the regeneration interval $[a, b]$. The cost of an extreme solution of WW is equal to the sum of the costs of the regeneration intervals composing it. Therefore, by Proposition 3, Partition-Int(ϕ) outputs the optimal solution of WW.

The computations carried out during the initialization and in Line 2 can be implemented in $O(n^2)$ operations. by Proposition 8, Solve-Q is $O(n \log n)$ and there are n calls to it. Hence, Line 1 is $O(n^2 \log n)$. Finally, Partition-Int is $O(n^2)$ by Proposition 3. ■

4 Problems with Backlogging

4.1 Building Blocks

Allowing backlogging in a lotsizing problem amounts to allow the violation of the demand satisfaction constraints, but penalizing such violation. Through proper normalization, it can be shown that it is equivalent to penalize the slack (i.e. the stock), the violation (i.e. the backlog), or both. The problem P that we introduce below is a generalization of the problem Q of the previous section, in which the violation of constraint (7) is allowed and the slack in (7) is penalized.

Let Problem P be defined as the collection of problems $\{\{P(k, p)\}_{p=0}^{m_{1k}}\}_{k=1}^{n}$,

where $P(k,p)$ is formulated as

$$p^*(k,p) = \min \sum_{i=1}^{k} (f_i y_i + h_i (\sum_{j=1}^{i} y_j - d_{1i})^+), \tag{27}$$

$$\sum_{i=1}^{k} y_i = p, \tag{28}$$

$$y_t \in \{0,1,\ldots,m_t\} \qquad 1 \le t \le k. \tag{29}$$

We make the assumption that $h_i \ge 0$ for all $1 \le i \le n$. Letting $s_t = (\sum_{j=1}^{i} y_j - d_{1i})^+$ and $r_t = (d_{1i} - \sum_{j=1}^{i} y_j)^+$, we see that $P(k,p)$ is a discrete lostizing problem with backlogging in which the backlogging costs have been normalized to zero. In this section, we will sometimes refer to these quantities as the "stock" or the "backlog" even though they do not appear formally in Problem $P(k,p)$.

Unlike Problem $Q(k,p)$, Problem $P(k,p)$ is not equivalent to the optimization of a linear function over a matroid. Yet we will see in the rest of this section that a natural greedy algorithm keeps most of the properties that it possesses when applied to matroid optimization problems. We use the following notation to denote the cost function appearing in (27).

$$C^k(y) = \sum_{i=1}^{k} \left(f_i y_i + g_i (\sum_{j=1}^{i} y_j - d_{1i})^+ \right)$$

To compare solutions of (27)–(29), we use the total order $C^k(y) <_{lex} C^k(z)$ in which ties are broken lexicographically, i.e. y is lexicographically earlier than z when $C^k(y) = C^k(z)$.

Lemma 14 *If $y = (y_1,\ldots,y_k)$ is an optimal solution of $P(k,p)$ and $\sum_{i=1}^{l} y_i = q$ for some $l < k$, then $y' = (y_1,\ldots,y_l)$ is an optimal solution of $P(l,q)$.*

Proof. Suppose by contradiction that $z' = (z_1,\ldots,z_l)$ is a strictly better solution of $P(l,q)$ than y'. Consider the point $z = (z_1,\ldots,z_l,y_{l+1},\ldots,y_k)$. We have

$$\begin{aligned} C^k(z) &= C^l(z') + \sum_{i=l+1}^{k} \left(f_i y_i + h_i (q + \sum_{j=l+1}^{i} y_j - d_{1i})^+ \right), \\ &< C^l(y') + \sum_{i=l+1}^{k} \left(f_i y_i + h_i (q + \sum_{j=l+1}^{i} y_j - d_{1i})^+ \right), \\ &= C^k(y), \end{aligned}$$

which contradicts the optimality of y. ∎

Lemma 14 suggests the following representation of the solutions of Problem P.

Definition 1 *Let $y(p)$ be the lexicographically earliest solution vector of $P(n,p)$. The solution matrix pair of Problem P is a pair (Z,A) of matrices of size $m_{1n} \times n$ such that $z_{pk} = \sum_{i=1}^{k} y_i(p)$ and a_{pk} is the value of an optimal solution of $P(k,p)$.*

The following result will be repeatedly used in the rest of this section.

Lemma 15 *If $C(y - e_{k_1} + e_{k_2}) <_{lex} C(y)$ (i.e. moving one batch from period k_1 to period k_2 is profitable) and one of the following conditions hold, then $C(z - e_{k_1} + e_{k_2}) <_{lex} C(z)$ (i.e. the same modification in solution z is also profitable). The conditions are that either*

(a) $k_1 < k_2$ and $\sum_{i=1}^{l} z_i \geq \sum_{i=1}^{l} y_i$ for $k_1 \leq l < k_2$ (i.e. moving a batch from k_1 to k_2 decreases the stock and the stock is higher in z than in y), or

(b) $k_2 < k_1$ and $\sum_{i=1}^{l} y_i \geq \sum_{i=1}^{l} z_i$ for $k_2 \leq l < k_1$ (i.e. moving a batch from k_1 to k_2 increases the stock and the stock is lower in z than in y).

Proof. Condition *(a)*.

$$\begin{aligned} C(z - e_{k_1} + e_{k_2}) - C(z) &= f_{k_2} - f_{k_1} + \sum_{t=k_1}^{k_2-1} h_t \left[(\sum_{i=1}^{t} z_i - 1 - d_{1i})^+ - (\sum_{i=1}^{t} z_i - d_{1i})^+ \right] \\ &\leq f_{k_2} - f_{k_1} + \sum_{t=k_1}^{k_2-1} h_t \left[(\sum_{i=1}^{t} y_i - 1 - d_{1i})^+ - (\sum_{i=1}^{t} y_i - d_{1i})^+ \right] \\ &= C(y - e_{k_1} + e_{k_2}) - C(y) \\ &\leq 0, \end{aligned}$$

where the first inequality is by Proposition 1. The result follows.
Condition *(b)*. This case is similar to *(a)*. ■

Note that the proof only relies on a property of the continuous function C. Hence, Lemma 15 is still true when considering fractional solutions or infeasible solutions.

Proposition 16 *If y is the lexicographically earliest (resp. lastest) optimal solution of $P(k, p-1)$, then there exists a period j such that $y + e_j$ is the lexicographically earliest (resp. latest) optimal solution of $P(k, p)$.*

Proof. We only prove it for the "earliest" case, since the reasoning is identical in the "latest" case. Let z be the lexicographically earliest optimal solution of $P(k, p)$. Let $k_1 = \min\{1 \leq t \leq k | y_t \neq z_t\}$ be the earliest period where z and y are different.
Case *1*. Suppose $z_{k_1} < y_{k_1}$. Because $\sum_{i=1}^{k} z_i = \sum_{i=1}^{k} y_i + 1$, there must exist $t > k_1$ such that $z_t > y_t$. Let k_2 be the earliest such t and observe that $\sum_{i=1}^{j} y_i - \sum_{i=1}^{j} z_i > 0$ for $k_1 \leq j < k_2$. We have that $y + e_{k_2} - e_{k_1}$ and $z + e_{k_1} - e_{k_2}$ are feasible in $P(k, p-1)$ and $P(k, p)$ respectively. Thus $C(y) <_{lex} C(y + e_{k_2} - e_{k_1})$. By Lemma 15, this implies that $C(z + e_{k_1} - e_{k_2}) <_{lex} C(z)$, which contradicts the optimality of z in $P(k, p)$.

Case *2*. Suppose $z_{k_1} > y_{k_1}$ and that there exists a $k_2 > k_1$ such that $z_{k_2} < y_{k_2}$. Because $y + e_{k_1} - e_{k_2}$ is feasible in $P(k, p-1)$, $C(y) <_{lex} C(y + e_{k_1} - e_{k_2})$. Observe that $z - e_{k_1} + e_{k_2}$ is feasible in $P(k, p)$ and has higher stock between k_1 and k_2 than y. By Lemma 15, this implies that $C(z - e_{k_1} + e_{k_2}) <_{lex} C(z)$, which contradicts the optimality of z in $P(k, p)$.

Case *3*. Suppose $z_{k_1} > y_{k_1}$ and there does not exist a $t > k_1$ such that $z_t < y_t$. Because $\sum_{i=1}^{n} z_i = \sum_{i=1}^{n} y_i + 1$, k_1 is the only period where y and z differ. Thus, the proposition is true. ■

These two results lead the following greedy algorithm.

GENERIC-P(n, h, f, m, d)

```
1   y ← (0, ..., 0)
2   for p ← 1 to m_{1n}
3       do for j ← 1 to n
4              do c(j) ← C^n(y + e_j) − C^n(y)
5          k ← min arg min_{j:y_j ≠ m_j} c(j)
6          y_k ← y_k + 1
7          for j ← 1 to n
8              do z_{pj} ← Σ_{i=1}^{j} y_i
9          for j ← 1 to n
10             do a_{pj} ← C^j(y)
```

Definition 2 *The solution sequence of Problem P is the sequence of periods selected by Algorithm* GENERIC-P *in Line 3.*

Example 3 *Consider the following instance of P.*

	1	*2*	*3*	*4*	*5*
d	*0.3*	*1.2*	*2.1*	*0.7*	*1.6*
m	*2*	*1*	*4*	*1*	*2*
f	*-4*	*-1*	*-2*	*-1*	*2*
h	*2*	*1*	*3*	*1*	*2*
d_{1t}	*0.3*	*1.5*	*3.6*	*4.3*	*5.9*

We detail the first six iterations. Algorithm GENERIC-P *will compute the following values $c(j) = C^n(y + e_j) - C^n(y)$. The smallest cost chosen at each iteration is highlighted. A star means that $y_t = m_t$ so that this period cannot be choosen anymore.*

Iteration	$c(1)$	$c(2)$	$c(3)$	$c(4)$	$c(5)$	$C^5(y)$
1	$\boxed{-2.6}$	-1	-2	-1	2	-2.6
2	-1.5	-0.5	$\boxed{-2}$	-1	2	-4.6
3	-1.5	-0.5	$\boxed{-2}$	-1	2	-6.6
4	-0.3	0.7	-0.8	$\boxed{-1}$	2	-7.6
5	0.4	1.4	$\boxed{-0.1}$	-0.3^*	2	-7.7
6	2.7	3.7	$\boxed{2.2}$	0.2^*	2.2	-5.5

The solution sequence is $\langle 1, 3, 3, 4, 3, 3\rangle$. After the sixth iteration, the matrices Z and A output by the algorithm contain:

$$Z = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 2 & 2 & 2 \\ 1 & 1 & 3 & 3 & 3 \\ 1 & 1 & 3 & 4 & 4 \\ 1 & 1 & 4 & 5 & 5 \\ 1 & 1 & 5 & 6 & 6 \end{pmatrix}, A = \begin{pmatrix} -2.6 & -2.6 & -2.6 & -2.6 & -2.6 \\ -2.6 & -2.6 & -4.6 & -4.6 & -4.6 \\ -2.6 & -2.6 & -6.6 & -6.6 & -6.6 \\ -2.6 & -2.6 & -6.6 & -7.6 & -7.6 \\ -2.6 & -2.6 & -7.4 & -7.7 & -7.7 \\ -2.6 & -2.6 & -6.4 & -5.7 & -5.5 \end{pmatrix}$$

Corollary 17 *The solution sequence S' of $P(n', p')$ with $n' \leq n$ and $p' \leq p$ is the subsequence of the solution sequence S of $P(n, p)$ restricted to the first p' periods less than or equal to n' in S.*

The correctness of Algorithm GENERIC-P follows immediately from Proposition 16 and Lemma 14. However, the algorithm's complexity is unsatisfactory as it is proportional to m_{1n}. Given a solution y, we partition the time periods into the sets $S_y^j = \{t | \lceil \sum_{i=1}^{t} y_i - d_{1t} \rceil = j\}$ for $j \leq 0$ and $S_y^+ = \{1, \ldots, n\} \setminus \bigcup_{j \leq 0} S_y^j$.

Proposition 18 *During the execution of Algorithm* GENERIC-P*, the vector c needs updating in Line 2 only if $\{k, \ldots, n\} \cap (S_y^{-1} \cup S_y^0) \neq \emptyset$, where k is the period chosen in Line 3 in the previous iteration. This happens at most $2n$ times.*

Proof. Expanding $c(j)$ yields (we use the notation $c_y(j)$ to emphasize the dependance on y)

$$c_y(j) = C^n(y + e_j) - C^n(y) = f_j + \sum_{\substack{i \in S_y^+ \\ i \geq j}} h_i + \sum_{\substack{i \in S_y^0 \\ i \geq j}} h_i \gamma_i,$$

where $\gamma_i = \lceil d_{1i} \rceil - d_{1i}$. Thus, when y is updated to $y + e_k$, we can update $c(j)$ as follows.

$$\Delta c_k(j) = c_{y+e_k}(j) - c_y(j) = \sum_{\substack{i \in S_y^0 \\ i \geq \max\{j,k\}}} (1 - \gamma_i) h_i + \sum_{\substack{i \in S_y^{-1} \\ i \geq \max\{j,k\}}} h_i \gamma_i. \qquad (30)$$

Hence, c_{y+e_k} differs from c_y only $\{k, \ldots, n\} \cap (S_y^{-1} \cup S_y^0) \neq \emptyset$. Furthermore, if $t \in \{k, \ldots, n\}$ and $t \in S_y^i$, then $t \in S_{y+e_k}^{i+1}$. Hence, if $c_{y+e_k} \neq c_y$, there exists a period t such that either $t \in S_y^{-1}$ and $t \in S_{y+e_k}^0$ or $t \in S_y^0$ and $t \in S_{y+e_k}^1$. Moreover, the index i of the set S_y^i to which belongs a given period always increases during the execution of the algorithm. Consequently, there can be at most $2n$ iterations such that $c_{y+e_k} \neq c_y$. ■

Observe that Proposition 18 applies to the second and third iteration of Example 3: $C^n(y + e_j) - C^n(y)$ remains the same, and iteration 3 "repeats" the calculations performed in iteration 2. Indeed, the second row of Z and the second row of A are exactly the mean between the first and the third rows. More generally, suppose that $c(y)$ is not modified during iterations $p, p+1, \ldots, p+k$ of GENERIC-P. Then, rows p to $p+k$ of the corresponding matrix solution pair (Z, A) can be computed by interpolating the rows $p-1$ and $p+k+1$. Identifying such "redundant" iterations is key to the efficient implementation that we discuss now. The following definition formalizes this idea.

Definition 3 *Let $y(p)$ be the lexicographically earliest solution vector of $P(n, p)$ and (Z', A') the solution matrix pair of P. A compact solution matrix pair (Z, A) of Problem P is a pair of matrices of size $O(n) \times n$ such that Z and A are submatrices of Z' and A' respectively, and satisfy the following conditions for any $p \geq 2$.*

(i) $(z_{p1}, \ldots, z_{pn}) - (z_{p-1,1}, \ldots, z_{p-1,n}) = k e_{in}$ *for some $i \in [1, n]$ and k a nonnegative integer.*

(ii) For any q such that $z_{p-1,n} < q < z_{pn}$,

$$p^*(n,q) = a_{p-1,n} + (q - z_{p-1,n})\frac{a_{pn} - a_{p-1,n}}{z_{pn} - z_{p-1,n}}.$$

A compact solution matrix is thus a small submatrix of a solution matrix, from which it is easy to recompute the missing rows. Proposition 18 implies that there always exists a compact solution matrix pair. In the following algorithm, the sets S_y^i are implemented as a vector $\sigma(t) = i$ if $t \in S_y^i$. Given an instance of P, SOLVE-P outputs the compact solution matrix pair (Z, A) and the number of rows q.

```
SOLVE-P(n, h, f, m, d)
 1  y ← (0, ..., 0)
 2  for t ← 1 to n
 3      do σ(t) ← ⌈−d_{1t}⌉
 4  for t ← 1 to n
 5      do if σ(t) = 0
 6            then c(t) ← f_t + h_t γ_t
 7            else c(t) ← f_t
 8  repeat
 9          k ← min arg min_j {c(j) : y_j < m_j}
10          p' ← max {1, min_{t≥k:σ(t)≤0} [−σ(t)]}
11          p ← min{p', m_k − y_k}
12          y_k ← y_k + p
13          q ← q + 1
14          for t ← 1 to n
15              do Z(q,t) ← Z(q−1,t)
16          Z(q,k) ← Z(q−1,k) + p
17          for t ← 1 to k − 1
18              do A(t,q) ← A(t,q−1)
19          for t ← 1 to k − 1
20              do A(t,q) ← A(t,q−1) + p(c(k) − c(t+1) + f_{t+1})
21          Δc ← 0
22          for t ← n to k by −1
23              do switch
24                    case σ(t) = −p' : Δc ← Δc + h_t γ_t
25                    case σ(t) ≥ 1 − p' : Δc ← Δc + h_t
26                 c(t) ← c(t) + Δc
27                 σ(t) ← σ(t) + p
28          for t ← k − 1 to 1
29              do c(t) ← c(t) + Δc
30    until Σ_{i=1}^n y_i = Σ_{i=1}^n m_i
31  return q, Z, A
```

Proposition 19 *Algorithm* SOLVE-P *is correct.*

Proof. We prove the following invariant.

i. The vector y is the lexicographically earliest optimal solution of the problem $P(n, \sum_{i=1}^n y_i)$.

ii. The vector σ satisfies $\sigma(t) = i$ for all $t \in S_y^i$.

iii. $c(j) = C^n(y + e_j) - C^n(y)$.

The invariant is true following the initialization. By Proposition 16, the algorithm chooses the correct period k. We increase y_k until there is a change in $c(j)$. This will happen after $\max\left\{1, \min_{t \geq k:\sigma(t) \leq 0}[-\sigma(t)]\right\}$ increments by Proposition 18. In Line 11, we prevent y_k from exceeding its upper bound m_k.

In lines 13 to 20, the output is recorded. The expression $c(k) - c(t+1) + f_{t+1}$ has a value of

$$\begin{aligned} & C^n(y + e_k) - C^n(y) - (C^n(y + e_{t+1}) - C^n(y)) + f_{t+1} \\ & \quad = f_k - f_{t+1} + \sum_{i=k}^{t} h_i \left((\sum_{j=1}^{i} y_j + 1 - d_{1i})^+ - (\sum_{j=1}^{i} y_j - d_{1i})^+ \right) + f_{t+1} \\ & \quad = C^t(y + e_k) - C^t(y). \end{aligned}$$

This justifies Line 20.

The block of lines 21–29 updates the vectors c and σ for the new incumbent. Using (30), we can calculate $\Delta c_k(j)$ recursively:

$$\Delta c_k(j) = \Delta c_k(j+1) + \begin{cases} h_j(1 - \gamma_j) & \text{if } j \geq k,\ j \in S_y^0, \\ h_j \gamma_j & \text{if } j \geq k,\ j \in S_y^{-1}, \\ 0 & \text{otherwise.} \end{cases} \tag{31}$$

This is implemented by the **switch** in line 23. ■

Example 4 *We detail the two first iterations of Algorithm* Solve-P *on the same instance than that of Example 3). After the initialization, the situation is the following.*

	1	2	3	4	5
y	0	0	0	0	0
σ	0	-1	-3	-4	-5
c	$\boxed{-2.6}$	-1	-2	-1	2

Iteration 1. We compute $k = 1$, $p' = 1$ *and* $p = 1$. *Then* $\Delta c_1(5) = \Delta c_1(4) = \Delta c_1(3) = 0$, $\Delta c_1(2) = 0 + 1 * 0.5 = 0.5$ *and* $\Delta c_1(1) = 0.5 + 2 * (1 - 0.7) = 1.1$. *The updated values are as follows.*

	1	2	3	4	5
y	1	0	0	0	0
σ	1	0	-2	-3	-4
c	-1.5	-0.5	$\boxed{-2}$	-1	2

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ -2.6 & -2.6 & -2.6 & -2.6 & -2.6 \end{pmatrix}$$

Iteration 2. We compute $k = 3$, $p' = 2$ *and* $p = 2$. *Then* $\Delta c_3(5) = \Delta c_3(4) = 0$, $\Delta c_3(3) = 3 * 0.4 = 1.2$ *and* $\Delta c_3(2) = \Delta c_3(1) = 1.2$. *The updated values are as follows.*

	1	2	3	4	5
y	1	0	2	0	0
σ	1	0	0	-1	-2
c	-0.3	0.7	-0.8	$\boxed{-1}$	2

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ -2.6 & -2.6 & -6.6 & -6.6 & -6.6 \end{pmatrix}$$

Note that this second iteration of Algorithm SOLVE-P *removes the redundant third iteration of Algorithm* GENERIC-P *(see the beginning of Example 3).* ■

Proposition 20 *Algorithm* SOLVE-P *terminates after $O(n^2)$ operations.*

Proof. At each iteration, either y_k reaches its bound m_k or an update of c is necessary. There can be at most n events of the first type and, by Proposition 18, at most $2n$ events of the second type. Hence there can be at most $3n$ iterations. The initialization and one iteration are both $O(n)$. ■

Lemma 21 *The optimal cost of $P(n,p)$ is a convex function of p.*

Proof. The relation (30) and the fact that $h_t \geq 0$ imply that the value of $\min_j c(j)$ increases monotonically during the course of SOLVE-P. The result follows. ■

Lemma 22 *Two instances P^1 and P^2 of $P(n,q)$ differing only by the demand in period n have the same optimal solutions.*

Proof. The cost functions of the two instances differ by the constant term $h_n\left((q-d^1_{1n})^+ - (q-d^2_{1n})^+\right)$ where d^i is the vector of demands of P^i. Moreover, the two problems have the same feasible set. The result follows. ■

Proposition 23 *Let P^1 and P^2 be two identical instances of $P(n,q)$ except that $m^2_j = m^1_j - 1$ for some $1 \leq j \leq n$. Let y^i denote the lexicographically earliest optimal solution of P^i. Then $y^2 = y^1 - e_j + e_l$ for some $1 \leq l \leq n$.*

Proof. As P^1 is a relaxation of P^2, if $y^1_j \leq m^1_j - 1 = m^2_j$, then y^1 is feasible in P^2 and thus optimal in P^2. In this case, the proposition is true with $l = j$. Therefore, we can suppose that $y^1_j = m^1_j$.

Let $\alpha^i = \sum_{t=1}^{j-1} y^i_t$ for $i = 1, 2$. By Lemma 14, $[y^i_1, \ldots, y^i_{j-1}]$ is the optimal solution of $P^i(j-1, \alpha^i)$. Since P^1 and P^2 are identical in the interval $[1, j-1]$, Proposition 16 implies that $(y^2_1, \ldots, y^2_{j-1}) = (y^1_1, \ldots, y^1_{j-1}) - e_S$ or $(y^2_1, \ldots, y^2_{j-1}) = (y^1_1, \ldots, y^1_{j-1}) + e_S$ where S is a multiset on the ground set $\{1, \ldots, j-1\}$.

Without loss of generality, we may assume that $d_1 n = q = \sum_{i=1}^n y_j$ since by Lemma 22, changing d_n does not affect the optimal solution. This assumption allows us to consider time in reverse order, and make a similar statement regarding $(y^2_{j+1}, \ldots, y^2_n)$ and $(y^1_{j+1}, \ldots, y^1_n)$ over the interval $[j+1, n]$. Therefore, at least one of the following case is true.

(i) $y^2 = y^1 - \lambda e_j - e_S + e_T$, $|T| = \lambda + |S|$,

(ii) $y^2 = y^1 - \lambda e_j + e_S - e_T$, $|S| = \lambda + |T|$,

(iii) $y^2 = y^1 - \lambda e_j + e_S + e_T$, $\lambda = |S| + |T|$,

where S and T are multisets on the ground set $\{1, \ldots, j-1\}$ and $\{j+1, \ldots, n\}$ respectively, and $\lambda \geq 1$. We show that in case (i) the only possibility is $|T| = 1$ and $S = \emptyset$, that in case (ii) the only possibility is $|S| = 1$ and $T = \emptyset$, and that in case (iii) the only possibility is that $\lambda = 1$ and either S or T is empty.

Case (i). Suppose by contradiction that $|T| \geq 2$. Let $k = j$ if $\lambda > 1$ and $k = \max_{t \in S} t$ otherwise. Let l_1 and l_2 be two periods in T. Then $C(y^2) <_{lex} C(y^2 - e_{l_2} + e_k)$ since y^2 is optimal in P^2. Observe that $y^1 - e_k + e_{l_2}$ is feasible in P^1 and has more stock between periods k and l_2 than y^2. Hence, by Lemma 15, $C(y^1 - e_k + e_{l_2}) <_{lex} C(y^1 - e_k + e_{l_2} - e_{l_2} + e_k) = C(y^1)$. This contradicts the optimality of y^1 in P^1.

Case (ii). This case is identical to case (i) when considering time in reverse order. Hence the same arguments shows that $|S| = 1$.

Case (iii). If S or T is an empty set, then this case reduces to case (i) or case (ii). Therefore $k = \max_{t \in S} t$ and $l = \min_{t \in T} t$ are well-defined. Then $C(y^2) <_{lex} C(y^2 - e_l + e_j)$ since y^2 is optimal for P^2. Observe that both $y^1 - e_j + e_l$ and $y^1 - e_j + e_k$ are feasible in P^1. Moreover, $y^1 - e_j + e_l$ has more stock between periods j and l than y^2. Hence, by Lemma 15, $C(y^1 - e_j + e_k) <_{lex} C(y^1)$. This contradicts the optimality of y^1 in P^1. ■

Proposition 24 *Let P^1 and P^2 be two identical instances of $P(n, q)$ except that $0 < \rho = d^1_{j+1} - d^2_{j+1} = d^2_j - d^1_j \leq 1$ for some $1 \leq j < n$ (i.e. ρ units of demand are shifted from $j+1$ to j). Let y^i denote the lexicographically earliest optimal solution of P^i. Then one of the following is true:*

(i) $y^2 = y^1$,

(ii) $y^2 = y^1 + e_{k_1} - e_{k_2}$, *where* $1 \leq k_1 \leq j$ *and* $j + 1 \leq k_2 \leq n$.

Proof. Let C^{ni} be the cost function associated with problem P^i. The following relation holds

$$C^2(y) = C^1(y - \rho e_j + \rho e_{j+1}), \tag{32}$$

for any vector y. Let $\alpha^i = \sum_{t=1}^{j} y^i_t$. By Lemma 14, $(y^i_1, \ldots, y^i_j)$ is the optimal solution of $P^i(j, \alpha_i)$. Moreover, by Lemma 22, $P^1(j, q)$ and $P^2(j, q)$ have the same optimal solution for any nonnegative integer q. Hence, by Proposition 16, $(y^1_1, \ldots, y^1_j) = (y^2_1, \ldots, y^2_j) \pm S$ where $S \subseteq \{1, \ldots, j\}$ is a multiset.

Without loss of generality, we may assume that $d_1 n = q = \sum_{i=1}^{n} y_j$ since by Lemma 22, changing d_n does not affect the optimal solution. This assumption allows us to consider time in reverse order, and make a similar statement regarding $(y^2_{j+1}, \ldots, y^2_n)$ and $(y^1_{j+1}, \ldots, y^1_n)$ over the interval $[j+1, n]$. Therefore, exactly one of the following case is true.

(i) $[y^2_1, \ldots, y^2_n] = [y^1_1, \ldots, y^1_n] - e_S + e_T$ and $|S| = |T| \geq 0$ or

(ii) $[y^2_1, \ldots, y^2_n] = [y^1_1, \ldots, y^1_n] + e_S - e_T$ and $|S| = |T| \geq 1$,

where S and T are multisets on the ground sets $\{1,\ldots,j\}$ and $\{j+1,\ldots,n\}$ respectively. We analyse each case.

Case (i). Suppose by contradiction that $|S| = |T| \geq 1$. Let $k = \max_{t\in S} t$ and $l = \min_{t\in T} t$. By definition, $C^2(y^2) < C^2(y^2 - e_l + e_k)$. Because the stock between k and l in $y^1 - e_k + e_l - \rho e_{j+1} + \rho e_j$ is higher than in y^2, Lemma 15 implies that $C^2(y^1 - e_k + e_l - \rho e_{j+1} + \rho e_j) < C^2(y^1 - \rho e_{j+1} + \rho e_j)$. By (32) it follows that $C^1(y^1 - e_k + e_l) < C^1(y^1)$, which contradicts the optimality of y^1 in P^1. Hence $S = T = \emptyset$.

Case (ii). Suppose by contradiction that $|S| = |T| \geq 2$. Let $k \leq j$ be the latest period such that $y^2_k > y^1_k$. Similarly, let $l \geq j+1$ be the earliest such period such that $y^2_l < y^1_l$. By definition, $C^2(y^2) < C^2(y^2 - e_k + e_l)$. Because the stock between k and l in $y^1 - e_l + e_k - \rho e_{j+1} + \rho e_j$ is lower than in y^2, Lemma 15 implies that $C^2(y^1 - e_l + e_k - \rho e_{j+1} + \rho e_j) < C^2(y^1 - \rho e_{j+1} + \rho e_j)$. By (32) it follows that $C^1(y^1 - e_l + e_k) < C^1(y^1)$, which contradicts the optimality of y^1 in P^1. Hence $|S| = |T| = 1$. ■

4.2 Discrete Lotsizing with Backlogging

The discrete lotizing problem with backlogging $DLSB$ is the special case of LSB where $x_t = y_t$ for all periods t, and $s_0 = r_0 = 0$. Because of the $2n$ equality constraints (2) and $x_t = y_t$, we may assume without loss of generality that $g_t = p_t = 0$ for all t. Eliminating x_t and r_t leads to the following formulation.

$$\min \sum_{i=1}^{n} f_i y_i + \sum_{i=1}^{n} h_i s_i, \tag{33}$$

$$s_t \geq \sum_{i=1}^{t} y_i - d_{1t} \qquad 1 \leq t \leq n, \tag{34}$$

$$s_t \geq 0, y_t \in \{0,1,\ldots,m_t\} \qquad 1 \leq t \leq n, \tag{35}$$

We assume that $h_i \geq 0$ to avoid unboundedness. $DLSB$ is thus a problem of type $P(n,\cdot)$ without the cardinality constraint (28). Therefore, to determine the optimal solution of $DLSB$, it suffices to solve the corresponding problem $\{\{P(k,p)\}_{p=0}^{m_{1k}}\}_{k=1}^{n}$, and identify the optimal cardinality p. As the compact matrix solution pair is of size $O(n) \times O(n)$, by Lemma 21, this optimal p can be determined in $O(\log n)$ by binary search. The dominating operation to solve $DLSB$ is thus Solve-P and we have the following result.

Proposition 25 *Problem DSLB can be solved in $O(n^2)$ time.*

4.3 Constant Capacity Lotsizing with Backlogging

This is the most general problem already introduced in Section 2. We restate it here for the sake of clarity. For notational convenience, we will suppose that

there is no initial and final stock or backlogging.

$$\min \sum_{t=1}^{n} f_t y_t + \sum_{t=1}^{n} p_t x_t + \sum_{t=0}^{n} h_t s_t + \sum_{t=0}^{n} g_t r_t, \tag{36}$$

$$s_{t-1} + r_t + x_t = d_t + r_{t-1} + s_t \qquad 1 \le t \le n, \tag{37}$$

$$(LSB) \qquad x_t \le y_t \qquad 1 \le t \le n, \tag{38}$$

$$s, r, x \ge 0, \tag{39}$$

$$y_t \in \{0, 1, \ldots, m_t\} \qquad 1 \le t \le n. \tag{40}$$

where we assume without loss of generality that $g_t + h_t \ge 0$ for all t (otherwise the objective is unbounded) and $f_t \ge 0$ (otherwise we can set $f_t = 0$ and add the constant $f_t m_t$ to the objective). Because of the flow balance constraints we may assume that $p_t = 0$ for all t.

Given an extreme solution (x, s, y, r), we call t a *regeneration period* if $s_t + r_t = 0$ and $[a, b]$ a *regeneration interval* if $a - 1$ and b are two regeneration periods with none in between.

Within a regeneration interval of an extreme solution there can only be at most one period $j \in \{a, \ldots, b\}$ with $0 < x_j < y_j$. We call this period the *fractional period* of this regeneration interval. Let $0 < \rho_{ab} = d_{ab} - (\lceil d_{ab} \rceil - 1) \le 1$. If $\rho_{ab} = 1$, there is no fractional period, and one is arbitrarily chosen. Since $f_t \ge 0$, it follows that there exists an extreme optimal solution such that $x_t = y_t$ in all periods $t \in \{a, \ldots, b\}$, except for the fractional period where $x_j = y_j - 1 + \rho_{ab}$.

Let ϕ_{ab} be the cost of an optimal production schedule for the $[a, b]$ regeneration interval. Once these costs are known for all intervals, the optimal solution of the problem can be computed in $O(n^2)$ calculations by dynamic programming. We now show how to compute all ϕ_{ab} efficiently.

Consider a fixed regeneration interval $[a, b]$. Let INT_{ab} denote the problem of finding an optimal production plan within $[a, b]$. Let $\epsilon_t = 1$ if the fractional batch is produced in period t and 0 otherwise and let υ_t denote the number of full batches produced in period t. We formulate INT_{ab} as follows.

$$\phi_{ab} = \min \sum_{t=a}^{b} f_t y_t + \sum_{t=a}^{b-1} (h_t s_t + g_t r_t) \tag{41}$$

$$\sum_{t=a}^{b} \upsilon_t = \lceil d_{ab} \rceil - 1, \tag{42}$$

$$\sum_{t=a}^{b} \epsilon_t = 1, \tag{43}$$

$$s_t - r_t = \sum_{i=a}^{t} (\upsilon_i + \rho_{ab} \epsilon_i) - d_{at} \qquad a \le t \le b - 1, \tag{44}$$

$$y_t = \upsilon_t + \epsilon_t \qquad a \le t \le b \tag{45}$$

$$s_t, r_t \ge 0, y_t \in \{0, 1, \ldots, m_t\} \qquad a \le t \le b. \tag{46}$$

Let $INT_{ab}(j)$ be the problem INT_{ab} with the additional constraint $\epsilon_j = 1$ (i.e. the fractional period is fixed to be j). Furthermore, let us fix the number of full batches before and after period j to subdivide problem $INT_{ab}(j)$ into

subproblems that we can solve efficiently as follows.

$$\phi_{ab}(j) = \min_{q^l,q^r,q^m} l_a^*(j,q^l) + r_b^*(j,q^r) + (q^m+1)f_j, \tag{47}$$

$$q^l + q^r + q^m = \lceil d_{ab} \rceil - 1, \tag{48}$$

$$0 \leq q^m \leq m_j - 1, \tag{49}$$

$$q^l, q^r, q^m \text{ integer}, \tag{50}$$

where $l_a^*(j,q^l)$ is the value of the optimal solution of problem $P(j-a+1,q^l)$ with data $f = (f_a,\ldots,f_{j-1})$, $d = (d_a,\ldots,d_{j-1})$, $h = (h_a,\ldots,h_{j-1})$, $g = (g_a,\ldots,g_{j-1})$ and $m = (m_a,\ldots,m_{j-1})$. Similarly, $r_b^*(j,q^r)$ is the value of the optimal solution of the problem $P(b-j,q^r)$ with data $f = (f_b,\ldots,f_{j+1})$, $d = (d_b,\ldots,d_{j+1})$, $h = (h_{b-1},\ldots,h_j)$, $h = (g_{b-1},\ldots,g_j)$ and $m = (m_b,\ldots,m_{j+1})$.

We will use the results of Section 4.1 and in particular Algorithm GENERIC-P (or its efficient implementation SOLVE-P) to compute the values $l_a^*(j,q^l)$ and $r_a^*(j,q^r)$. A generic algorithm is as follows.

GENERIC-LSB(n,h,g,f,m,d)

```
 1  for a ← 1 to n, b ← a to n, j ← a to b
 2      do φ_ab(j) ← ∞
 3  for a ← 1 to n
 4      do f^la ← (f_a,...,f_n)
 5         d^la ← (d_a,...,d_n)
 6         h^la ← (h_a,...,h_n)
 7         g^la ← (g_a,...,g_n)
 8         m^la ← (m_a,...,m_n)
 9  for b ← 1 to n
10      do f^rb ← (f_1,...,f_b)
11         d^rb ← (d_1,...,d_b)
12         h^rb ← (h_0,...,h_{b-1})
13         g^rb ← (g_0,...,g_{b-1})
14         m^rb ← (m_1,...,m_b)
15  for a ← 1 to n
16      do (ZA_a, AA_a, qa_a) ← SOLVE-P(h^la, f^la, m^la, d^la)
17  for a ← 1 to n, j ← a to n, q ← 1 to m_{a,j-1}
18      do l_a^*(j,q) ← AA_a(q, j-a)
19  for b ← 1 to n
20      do (ZB_b, AB_b, qb_b) ← SOLVE-P(h^rb, f^rb, m^rb, d^rb)
21  for b ← 1 to n, j ← 1 to b, q ← 1 to m_{j+1,b}
22      do r_b^*(j,q) ← AB_b(q, j-b)
23  for a ← 1 to n, b ← a to n, j ← a to b
24      do Solve the program (47)–(50) .
25  for a ← 1 to n, b ← a to n
26      do φ_ab ← min_{j=a..b} φ_ab(j)
27  return PARTITION-INT(φ).
```

The first 14 lines of the algorithm format the data for the calls to SOLVE-P of lines 16 and 20. After these calls to SOLVE-P, all the values $l_a^*(j,q)$ and $r_b^*(j,q)$ appearing in (47) are known. Thus, the programs (47)–(50) can be solved to compute $\phi_{ab}(j)$ for all $1 \leq a \leq j \leq b \leq n$. It then remains to select the best possible j for each interval $[a,b]$ and to compute the optimal partition of the interval $[1,n]$ into regeneration intervals. Thus we have the following result.

Proposition 26 *Algorithm* GENERIC-LSB *is correct.* ■

However, the complexity of GENERIC-LSB is not polynomial in n. It depends on the value of the vector m. Observe that in the simple where case $m_t = 1$ for each t, the running time becomes $O(n^4)$. The algorithm GENERIC-LSB may be seen as the natural extension of the $O(n^4)$ Florian and Klein algorithm [7] designed for the same problem but without backlogging. This algorithm also divides the original problem into regeneration intervals problems in which the fractional period is fixed. The arguments are similar, and the complexity of this type of algorithm is unsensitive to the presence of backlogging.

The next proposition shows how to reduce the number of potentially optimal values of q^l, q^m, q^r to obtain a running time of $O(n^3)$. A similar proposition appears in van Hoesel and Wagelmans [16], where the problem without backlogging and $m_t = 1$ for all t is studied. We define a surrogate problem $INT^s_{ab}(j)$ which is identical to $INT_{ab}(j)$ except that we neglect the capacity used up by the fractional batch in period j. Formally, constraint (49) is relaxed to be

$$q^m \leq m_j.$$

By eliminating all variables r_t from the formulation (41)–(46), we see that $INT^s_{ab}(j)$ is equivalent to an instance of Problem $P(b-a+1, \lceil d_{ab} \rceil - 1)$ with demand $(d_a, \ldots, d_{j-1}, d_j - \rho_{ab}, d_{j+1}, \ldots, d_b)$.

Proposition 27 *Exactly one of the following is true:*

(i) $\upsilon^s_{ab}(j+1) - \upsilon^s_{ab}(j) = 0$

(ii) $\upsilon^s_{ab}(j+1) - \upsilon^s_{ab}(j) = e_{k_1} - e_{k_2}$,

where $k_1 \leq j$ *and* $k_2 \geq j+1$.

Proof Proposition 24 applies to the pair of problems $INT^s_{ab}(j)$ and $INT^s_{ab}(j+1)$ and the result follows. ■

The line 24 of Algorithm GENERIC-P essentially computes the optimal distribution (q^l, q^m, q^r) (production before period j, in period j and after period j) for each interval $[a, b]$ and each fractional period j. The algorithm does not need the precise values $\upsilon^s_{ab}(j)$. The following corollary is a translation of Proposition 27 in a vocabulary better suited for describing an algorithm.

We will need the following notation. Let $\upsilon_{ab}(j)$ (resp. $\upsilon^s_{ab}(j)$) denote the lexicographically optimal solution of $INT_{ab}(j)$ (resp. $INT^s_{ab}(j)$). Let $q^l_{ab}(j) = \sum_{i=a}^{j-1} \upsilon_{ab,i}(j)$, $q^m_{ab}(j) = \upsilon_{ab,j}(j)$ and $q^r_{ab}(j) = \sum_{i=j+1}^{b} \upsilon_{ab,i}(j)$ be the optimal distribution of the $\lceil d_{ab} \rceil - 1$ full batches in $INT_{ab}(j)$. Similarly, let $q^{sl}_{ab}(j) = \sum_{i=a}^{j-1} \upsilon^s_{ab,i}(j)$, $q^{sm}_{ab}(j) = \upsilon^s_{ab,j}(j)$ and $q^{sr}_{ab}(j) = \sum_{i=j+1}^{b} \upsilon^s_{ab,i}(j)$ be the optimal distribution of the $\lceil d_{ab} \rceil - 1$ full batches in $INT^s_{ab}(j)$.

In the next corollary, the case *(i)* corresponds to the case *(i)* of Proposition 27. The case *(ii)* (resp. case *(iii)*) corresponds to the case *(ii)* of Proposition 27 and $k_2 \geq j+1$ (resp. $k_2 = j+1$).

Corollary 28 *Exactly one of the following is true:*

(i) $q^{sl}_{ab}(j) = q^{sl}_{ab}(j-1) + q^{sm}_{ab}(j-1)$, $q^{sm}_{ab}(j) = \upsilon^s_{ab,j}(j-1)$ *and* $q^{sr}_{ab}(j) = q^{sr}_{ab}(j-1) - \upsilon^s_{ab,j}(j-1)$,

(ii) $q^{sl}_{ab}(j) = q^{sl}_{ab}(j-1) + q^{sm}_{ab}(j-1) + 1$, $q^{sm}_{ab}(j) = \upsilon^{s}_{ab,j}(j-1)$ *and* $q^{sr}_{ab}(j) = q^{sr}_{ab}(j-1) - \upsilon^{s}_{ab,j}(j-1) - 1$,

(iii) $q^{sl}_{ab}(j) = q^{sl}_{ab}(j-1) + q^{sm}_{ab}(j-1) + 1$, $q^{sm}_{ab}(j) = \upsilon^{s}_{ab,j}(j-1) - 1$ *and* $q^{sr}_{ab}(j) = q^{sr}_{ab}(j-1) - \upsilon^{s}_{ab,j}(j-1)$.

■

Proposition 27 and Corollary 28 relate the two surrogate problems $INT^{s}_{ab}(j)$ and $INT^{s}_{ab}(j+1)$. What we really need is to relate the optimal solutions of the real problems $INT_{ab}(j)$ and $INT_{ab}(j+1)$. The next proposition provides this link indirectly by relating the optimal solutions of $INT^{s}_{ab}(j)$ and $INT_{ab}(j)$ for any j.

Proposition 29 $\upsilon_{ab}(j) - \upsilon^{s}_{ab}(j) = e_l - e_j$, *where* $a \leq l \leq b$.

Proof. $INT^{s}_{ab}(j)$ and $INT_{ab}(j)$ satisfy the conditions of Proposition 23. The result follows. ■

Again, we need to translate this result into the language of (q^l, q^m, q^r) in order to describe an algorithm. If $\upsilon^{s}_{ab,j}(j) < m_j$ (case *(i)* of Corollary 30), the optimal solution $\upsilon^{s}_{ab}(j)$ of the relaxed problem $INT^{s}_{ab}(j)$ is feasible in $INT_{ab}(j)$. Hence it is also optimal in $INT_{ab}(j)$. If $\upsilon^{s}_{ab,j}(j) = m_j$ (case *(ii)* of Corollary 30), the optimal solution $\upsilon^{s}_{ab}(j)$ of the relaxed problem $INT^{s}_{ab}(j)$ is infeasible in $INT_{ab}(j)$. Thus, in Proposition 29, we have $l < j$ or $l > j$. This corresponds to the case *(iia)* and *(iib)* of the Corollary 30 respectively.

Corollary 30

(i) *If* $\upsilon^{s}_{ab,j}(j) < m_j$ *then* $q^{l}_{ab}(j) = q^{sl}_{ab}(j)$, $q^{m}_{ab}(j) = q^{sm}_{ab}(j)$ *and* $q^{r}_{ab}(j) = q^{sr}_{ab}(j)$.

(ii) *If* $\upsilon^{s}_{ab,j}(j) = m_j$ *then either*

a. $q^{l}_{ab}(j) = q^{sl}_{ab}(j)$, $q^{m}_{ab}(j) = q^{sm}_{ab}(j) - 1$ *and* $q^{r}_{ab}(j) = q^{sr}_{ab}(j) + 1$ *or*

b. $q^{l}_{ab}(j) = q^{sl}_{ab}(j) + 1$, $q^{m}_{ab}(j) = q^{sm}_{ab}(j) - 1$ *and* $q^{r}_{ab}(j) = q^{sr}_{ab}(j)$.

■

The two corollaries show how to compute each $\phi_{ab}(j)$ efficiently. The procedure has two steps. In the first step, we compute the optimal values $q^{sl}(j)$, $q^{sm}(j)$ and $q^{sr}(j)$ of the surrogate problem $INT^{s}_{ab}(j)$. It follows from Corollary 28 that, knowing $q^{sl}(j-1), q^{sm}(j-1), q^{sr}(j-1)$ and $\upsilon^{s}_{ab,j}(j-1)$, there are only three potentially optimal triples. Computing the cost of each one and selecting the cheapest is possible in $O(1)$ time if we can compute the values $l^{*}_{a}(q^{sl}_{ab}(j-1) + q^{sm}_{ab}(j-1), j)$, $l^{*}_{a}(q^{sl}_{ab}(j-1) + q^{sm}_{ab}(j-1) + 1, j)$, $r^{*}_{b}(q^{sr}_{ab}(j-1) - \upsilon^{s}_{ab,j}(j-1), j)$ and $r^{*}_{b}(q^{sr}_{ab}(j-1) - \upsilon^{s}_{ab,j}(j-1) - 1, j)$ in $O(1)$ time.

In the second step, we compute the optimal distribution $q^l(j), q^m(j), q^r(j)$ of the original problem $INT_{ab}(j)$. Corollary 30 shows that this is possible in $O(1)$ if we can compute the desired values $l^{*}_{a}(q^{sl}_{ab}(j), j)$, $l^{*}_{a}(q^{sl}_{ab}(j)+1, j)$ and $r^{*}_{b}(q^{sr}_{ab}(j), j)$, $r^{*}_{b}(q^{sr}_{ab}(j)+1, j)$ in $O(1)$ time. Before showing that achieving this complexity bound is possible, we describe such an efficient implementation of Generic-LSB.

SOLVE-LSB(n, h, g, f, m, d)

1 Same as GENERIC-LSB up to line 22.
2 **for** $a \leftarrow 1$ **to** $n, b \leftarrow a$ **to** n
3 **do** $q^{sm} \leftarrow \arg\min_{q=0..m_a}\{r_b^*(a+1, \lceil d_{ab}\rceil - 1 - q) + f_a q\}$
4 $q^{sl} \leftarrow 0$
5 $q^{sr} \leftarrow \lceil d_{ab}\rceil - 1 - q^{sm}$
6 $q^m \leftarrow \min\{m_a - 1, q^{ms}\}$
7 $\phi_{ab} \leftarrow r_b^*(a+1, \lceil d_{ab}\rceil - 1 - q^m) + (q^m + 1)f_a$
8 **for** $j \leftarrow a+1$ **to** b
9 **do** $(q^{sl}, q^{sm}, q^{sr}) \leftarrow \arg\min_{(p^s,p^m,p^r)\in T_{28}}\{l_a^*(j, p^s) + r_b^*(j, p^r) + f_j(p^m + 1)\}$
10 $(q^l, q^m, q^r) \leftarrow \arg\min_{(p^s,p^m,p^r)\in T_{30}}\{l_a^*(j, p^s) + r_b^*(j, p^r) + f_j(p^m + 1)\}$
11 $\phi_{ab} \leftarrow \min\{\phi_{ab}, l_a^*(j, q^l) + r_b^*(j, q^r) + (q^m + 1)f_j\}$
12 **return** PARTITION-INT(ϕ)

The sets T_{28} and T_{30} denote the possible values of the triples listed in Corollaries 28 and 30 respectively.

Example 5 *We illustrate the Algorithm* SOLVE-LSB *on a 4 period regeneration interval. Let* $d = [39.6, 11.9, 17.5, 13.8]$, $m = [13, 42, 25, 16]$, $f = [4, 2, 4, 2]$, $h = [1, 2, 3]$ *and* $g = [2, 1, 3]$. *The fractional batch size* ρ *is equal to* 0.8 *and the compact solution matrices computed in the first two lines of* SOLVE-LSB *are the following.*

$$AA_1 = \begin{pmatrix} 79.2 & 130.7 & 337.7 \\ 105.2 & 143.7 & 311.7 \\ 105.2 & 181.7 & 235.7 \\ 105.2 & 184.2 & 235.2 \\ 105.2 & 184.2 & 252.2 \\ 105.2 & 184.2 & 308.2 \\ 105.2 & 196.2 & 329.2 \end{pmatrix} \quad ZA_1 = \begin{pmatrix} 0 & 0 & 0 \\ \boxed{13} & 13 & 13 \\ 13 & \boxed{51} & 51 \\ 13 & 52 & \boxed{52} \\ 13 & 52 & 69 \\ 13 & 52 & 77 \\ 13 & 55 & 80 \end{pmatrix}$$

$$AB_4 = \begin{pmatrix} 41.4 & 104 & 147.2 \\ 28.4 & 65 & 95.2 \\ 28.6 & 63.2 & 92.4 \\ 28.6 & 97.2 & 109.4 \\ 28.6 & 97.2 & 121.4 \\ 28.6 & 97.2 & 124.8 \\ 28.6 & 97.2 & 240.8 \\ 28.6 & 101.3 & 246.9 \\ 28.6 & 136.3 & 295.9 \\ 38.6 & 148.3 & 311.9 \end{pmatrix} \quad ZB_4 = \begin{pmatrix} 0 & 0 & 0 \\ 13 & 13 & 13 \\ 14 & 14 & 14 \\ 14 & 31 & 31 \\ 14 & 31 & 43 \\ 14 & 31 & 44 \\ \boxed{14} & \boxed{31} & \boxed{73} \\ 14 & 32 & 74 \\ 14 & 39 & 81 \\ 16 & 41 & 83 \end{pmatrix}$$

The following tables describe the optimal solution of $INT^s(j)$ *and* $INT(j)$.

j	$\upsilon_1^s(j)$	$\upsilon_2^s(j)$	$\upsilon_3^s(j)$	$\upsilon_4^s(j)$	$\upsilon_1(j)$	$\upsilon_2(j)$	$\upsilon_3(j)$	$\upsilon_4(j)$
1	13	38	17	14	13	39	17	14
2	13	38	17	14	13	39	17	14
3	13	39	16	14	13	39	17	14
4	13	39	17	13	13	39	17	14

j	$q^{sl}(j)$	$q^{sm}(j)$	$q^{sr}(j)$	$q^{l}(j)$	$q^{m}(j)$	$q^{r}(j)$
1	–	13	69	–	12	70
2	13	38	31	13	38	31
3	52	16	14	52	16	14
4	69	13	–	69	13	–

In Lines 4–6, we determine $q^{sm}(1) = 13$ *and* $q^{sr}(1) = 69$. *This is done by minimizing* $\{r_b^*(a+1, \lceil d_{ab} \rceil - 1 - q) + f_a q\}$ *where* $r_b^*(a+1, \lceil d_{ab} \rceil - 1 - q)$ *is computed from the third column of* ZB_4. *As will be showed later, this can be done by binary search since* $\{r_b^*(a+1, \lceil d_{ab} \rceil - 1 - q) + f_a q\}$ *is convex in* q *(see the values in any column of* ZA_1 *or* ZB_4*).*

Because $q^{sm}(1) = m_1 = 13$, *and* $q^{m}(1) = q^{sm}(1) - 1 = 12$, *the only possibility is to produce this batch later and thus* $q^{r}(1) = q^{sr}(1) + 1 = 70$. *Observe that there is no row in* ZB_4 *containing a solution with a cardinality of* 70. *However, from the definition of a compact solution matrix pair, the desired value can be computed by taking the weighted sum of* 124.8 *and* 240.8 *corresponding to* $q^{sr}(1) = 44$ *and* $q^{sr}(1) = 73$*:* $r_b^*(a+1, \lceil d_{ab} \rceil - 1 - q) = 124.8 + (240.8 - 124.8) * (70 - 44)/(73 - 44) = 228.8$. *Hence* $\phi(1) = 228.8 + 13 * 4 = 280.8$.

It follows from Corollary 28 that $(q^{sl}(2), q^{sm}(2), q^{sr}(2))$ *can only take the values* $(13, 38, 31)$, $(14, 38, 30)$ *or* $(14, 37, 30)$. *The last two possibilities are infeasible because* $m_1 = 13$. *Thus, using Corollary 30,* $(q^{l}(2), q^{m}(2), q^{r}(2)) = (13, 38, 31)$ *because* $q^{sm}(2) = 38 < m_2 = 42$. *Hence* $\phi(2) = 105.2 + 97.2 + 39 * 2 = 280.4$.

Using again Corollary 28, $(q^{sl}(3), q^{sm}(3), q^{sr}(3))$ *can only take the values* $(51, 17, 14)$, $(52, 17, 13)$ *or* $(52, 16, 14)$. *The corresponding costs are* $181.7 + 28.6 + 17 * 4 = 278.3$, $184.2 + 28.4 + 17 * 4 = 280.6$ *and* $184.2 + 28.6 + 16 * 4 = 276.8$. *Hence* $(q^{sl}(3), q^{sm}(3), q^{sr}(3)) = (52, 16, 14)$. *From Corollary 30, it follows that* $(q^{l}(3), q^{m}(3), q^{r}(3)) = (52, 16, 14)$ *since* $q^{sm}(2) = 16 < m_2 = 25$. *Hence* $\phi(3) = 276.8 + 4 = 280.8$.

Similarly, one can compute that $(q^{l}(3), q^{m}(3), q^{r}(3)) = (69, 13, -)$ *to find* $\phi(4) = 252.2 + 14 * 2 = 280.2$. *This is the cheapest solution.*

Before proving that SOLVE-LSB can be implemented in $O(n^3)$ operations, we need a few more definitions. Let $\alpha_{ab}(j) = \min\{p | ZA_a(p, j-a) \geq q_{ab}^{sl}(j-1) + q_{ab}^{sm}(j-1)\}$. An equivalent definition is that $\alpha_{ab}(j)$ is the last row p in the matrix ZA_a such that any previous row holds a solution which contains fewer batches than $q_{ab}^{sl}(j-1) + q_{ab}^{sm}(j-1)$ in the interval $[a, j-1]$. Proposition 27 implies that $\alpha_{ab}(a+1) \leq \ldots \leq \alpha_{ab}(b)$. In the previous example, for each column of the matrix ZA_a, the element $ZA_a(p, j-a)$ is highlighted.

Similarly, let $\beta_{ab}(j) = \max\{p | ZB_b(p, b-j) \geq q_{ab}^{sr}(j-1) - \upsilon_{ab,j}(j-1)\}$. Proposition 27 implies that $\beta_{ab}(b-1) \geq \beta_{ab}(b-2) \geq \ldots \geq \beta_{ab}(a)$.

Observation 1 *Let* $a \leq b$ *be fixed. To find* $\alpha_{ab}(j+1)$*, we consider the elements* $\{ZA_a(\alpha_{ab}(j), j+1-a), ZA_a(\alpha_{ab}(j)+1, j+1-a), \ldots\}$ *until the considered entry reaches the value* $q_{ab}^{sl}(j) + q_{ab}^{sm}(j)$. *Thus, we need to read at most the following elements* $ZA_a(\alpha_{ab}(j), j+1-a), \ldots, ZA_a(\alpha_{ab}(j+1), j+1-a)$. *Since the matrix* ZA_a *is* $O(n) \times n$, *we need to read* $O(n)$ *entries of the matrix* ZA_a *to determine* $\alpha_{ab}(j)$ *for all* $a \leq j \leq b$. *Hence, for fixed* $a \leq b$ *and given matrices* ZA_a, ZB_b, *the values* $\alpha_{ab}(j)$ *for* $a < j \leq b$ *can be computed in* $O(n)$ *overall. A similar argument shows that* $\beta_{ab}(j)$ *for* $a \leq j < b$ *can be computed in* $O(n)$ *operations as well.*

Observation 2 *Given a matrix A of size $m \times n$ and an element a_{ij}, one can compute the next non identical entry in any direction from position (i, j) in constant time. Indeed, this information can be computed beforehand for each element of the matrix in an amount of time proportional to the size of the matrix.*

Proposition 31 *Algorithm* Solve-LSB *is correct and can be implemented in $O(n^3)$ operations.*

Proof. The correctness of the algorithm follows from Proposition 26 and the fact that Solve-LSB implements Generic-LSB.

Consider the first 22 lines of Algorithm Generic-LSB. The first 14 lines take no more than $O(n^3)$ time. From Proposition 20, it follows that the calls to Solve-P are $O(n^3)$ overall. Hence the first 22 lines of Algorithm Generic-LSB are $O(n^3)$.

Let a, b be fixed. We show that one iteration of loop starting at line 2 of Solve-LSB can be implemented in $O(n)$ time. Consider first lines 4 and 5 that initialize the loop. By Lemma 21, the function $r_b^*(a+1, \lceil d_{ab} \rceil - q)$ is a convex function of q. This function is stored as a column of the matrix AB_b. This column gives the values of the function at the breakpoints, with possible duplications. Moreover, this column is a vector of size $O(n)$. Since $r_b^*(a+1, \lceil d_{ab} \rceil - q) + f_a q$ is also a convex function of q, the row containing the optimal value can be computed in $O(\log n)$ by binary search. Once this row is determined, we can compute $y_{a+1}(a)$ from ZB_b in $O(1)$. Hence lines 4 and 5 are $O(\log n)$.

It follows from Observation 1 that we can compute $\alpha_{ab}(j)$ and $\beta_{ab}(j)$ for $a \leq j \leq b$ in $O(n)$ time. Also, $l_a^*(q_{ab}^{sl}(j-1) + q_{ab}^{sm}(j-1) + 1, j)$ can be found in the $j - a^{th}$ column of the matrix ZA_a which is not further than two non identical entries away from $(\alpha_{ab}(j), j-a)$. Thus, it follows from Observation 2 that $l_a^*(q_{ab}^{sl}(j-1) + q_{ab}^{sm}(j-1), j)$ can be computed in constant time. A similar argument applies to the other values $l_a^*(\cdot, j)$ and $r_b^*(\cdot, j)$ that are needed in lines 10–12.

Hence, the loop starting at line 2 is $O(n^3)$ overall, which dominates the dynamic program of line 15. ■

5 Perspectives

Based upon the complexity bounds obtained for lotsizing problems in the literature, it is often the case that backlogging does not make the problem more complex. We can cite as examples the uncapacitated and the constant capacity lotsizing problems. In the light of this conjecture, the results obtained in this work suggest two questions regarding the best complexity of the solution algorithm.

The first question concerns the discrete lotsizing problems with and without backlogging. The bounds obtained are $O(n^2)$ and $O(n \log n)$ with and without backlogging respectively. The $O(n^2)$ bound for the greedy algorithm Solve-P comes from the fact that there are $O(n)$ greedy iterations needed and that each one requires $O(n)$ time. It is difficult to imagine such an algorithm taking less than $O(n)$ iterations to terminate. Decreasing the complexity of one iteration

would at least require to implement line 9 of SOLVE-P in $O(\log n)$ time on average. Another possible approach to obtain an $O(n \log n)$ algorithm for Problem $DLSB$, would be to work period-wise, as does SOLVE-Q.

The second open question concerns the constant capacity lotsizing problem with Wagner-Whitin costs and backlogging. Can it be solved in $O(n^2 \log n)$ time? We are currently investigating this question.

Finally, most of the literature on discrete lotsizing considers multi item problems which are linked through cardinality constraints or changeovers. Developing efficient algorithms for the polynomially solvable cases of this type of problem is another possible direction for future research.

Acknowledgement. The author is very grateful to Laurence Wolsey, Yves Pochet and especially Hamish Waterer who spent much time to improve the readibility of this paper.

References

[1] A. Aggarwal and J.K. Park. Improved algorithms for economic lot size problems. *Operations Research*, 41:549–571, 1990.

[2] G. Bitran and H. Yanasse. Computational complexity of the capacitated lot size problem. *Management Science*, 28:1174–1185, 1982.

[3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Electrical and Computer Science Series. MIT Press, 2002.

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

[5] B. Fleischmann. The discrete lot-sizing and scheduling problem. *European J. Oper. Res.*, 44:337–348, 1990.

[6] B. Fleischmann. The discrete lot-sizing and scheduling problem with sequence-dependent setup costs. *European J. Oper. Res.*, 75:395–404, 1994.

[7] M. Florian and M. Klein. Deterministic production planning with concave costs and capacity constraints. *Management Science*, 18:12–20, 1971.

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

[9] L.S. Lasdon and R.C. Terjung. An efficient algorithm for multi-item scheduling. *Operations Research*, 19:946–969, 1971.

[10] A.J. Miller and L. Wolsey. Tight mip formulations for multi-item discrete lot-sizing problems. To appear in *Operations Research*.

[11] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & sons, Chichester, 1988. Wiley Interscience Series in Discrete Mathematics and Optimization.

[12] Y. Pochet and L.A. Wolsey. Lot-sizing with constant batches: Formulations and valid inequalities. *Mathematics of Operations Research*, 18:767–785, 1993.

[13] C.A. van Eijl. *A Polyhedral Approach to the Discrete Lot-sizing and Scheduling Problem*. PhD thesis, Technische Universiteit Eindhoven, June 1996.

[14] C.A. van Eijl and C.P.M. van Hoesel. On the discrete lot-sizing and scheduling problem. *Operations Research Letters*, 20:7–13, 1997.

[15] C.P.M. van Hoesel, R. Kuik, M. Salomon, and L.N. van Wassenhove. The discrete lot-sizing and scheduling problem. *Discrete Applied Mathematics*, 48:289–303, 1994.

[16] C.P.M. van Hoesel and A. Wagelmans. An $O(T^3)$ algorithm for the economic lotsizing problem with constant capacities. *Management Science*, 42:142–150, 1996.

[17] S. van Hoesel, A. Wagelmans, and B. Moerman. Using geometric techniques to improve dynamic programming algorithms for the economic lot-sizing problem and extensions. *European J. Oper. Res.*, 75:312–331, 1994.

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

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

[20] W.I. Zangwill. Minimum concave cost flows in certain networks. *Management Science*, 14:429–450, 1968.