Master Thesis Econometrics and Management Science

# The practical running time of some lot-sizing algorithms

Charlie Ye (302759)
Supervisor: Dr. W. van den Heuvel

August 10, 2012

**Abstract**

In this paper we consider both the uncapacitated and the capacitated economic lot-sizing model, in which demand is a deterministic function of the price. We use a constant price for all periods. Van den Heuvel and Wagelmans [2006] propose an exact algorithm to determine the optimal price and lot-sizing decisions, using an heuristic algorithm from Kunreuther and Schrage [1973]. They propose a theorem on the maximum number of breakpoints dependent on $T$. However, the proof contains a flaw discovered by the authors themselves. Geunes et al. [2009] apply the algorithm and results from Van den Heuvel and Wagelmans [2006] on the capacitated variant, but with the flaw in the proof it is uncertain whether their results still hold. The goal of this paper is to determine whether the flaw in the proof has effect on the proposed theorem and the obtained results. First, we will try to disprove the theorem by looking for a contradicting result. After which we will look at whether the flaw has any effect on the results of Geunes et al. [2009].

# Contents

# Chapter 1

# Introduction

The economic lot-sizing problem (ELSP) is a well-known problem in logistics. The problem consists of multiple decisions that have to be made regarding production or ordering. Over a planning horizon ($T$), one has to determine when to produce and how much has to be produced in order to meet the demand ($D_t$) in every period $t$. This is in regard to the costs, which can be categorized as setup costs ($K_t$), production costs ($c_t$) and holding costs ($h_t$), which are time-variant.

We can distinguish two main types of this problem. The problem can be either uncapacitated or capacitated. Uncapacitated means that there are no restrictions on how many items one can produce in one period. This type of problem will only be restricted in the costs as one wants to have as much profit as possible, thus minimizing the costs. In the capacitated problem there are restrictions on the production in a period. This can be caused by for example limited storage or production capacity on a machine. This restriction makes the problem more complicated and harder to solve.

Instead of looking at the original ELSP, we will be looking at a joint pricing and lot-sizing model. In contrast to the standard lot-sizing problem, we do not have deterministic demand, but demand is a deterministic function of price. Therefore it is not sufficient to solely minimize the costs, incurred by setup, production and holding costs, but also to find an optimal price for which profit would be maximized.

In this paper we will test the practical running time of an algorithm for a joint pricing and lot-sizing model proposed by Van den Heuvel and Wagelmans [2006]. They derive an exact algorithm to determine the optimal price and lot-sizing decisions for the uncapacitated economic lot-sizing problem where demand is a function of the price. They also show that their algorithm comes down to solving a number of lot-sizing problems which can be solved in polynomial time. They claim that they can solve this problem in $O(T^3 \log T)$ time, however it turned out that there is a flaw in the proof. Geunes et al. [2009] generalize the model of Van den Heuvel and Wagelmans [2006] to allow for constant production capacities and propose an $O(T^9)$ algorithm for this problem. Because of the assumption that the original algorithm from Van den Heuvel and Wagelmans [2006] can be solved in $O(T^3 \log T)$ time, it will be interesting to see whether the $O(T^9)$ algorithm still holds and which problem size can still be solved within a reasonable amount of time.

The goal of this paper is to see whether the theorem proposed by Van den Heuvel and Wagelmans [2006] still holds and that only the proof is flawed or that the theorem by itself is faulty. We will also observe whether the practical running time corresponds to the proposed running time of the algorithm of Van den Heuvel and Wagelmans [2006]. Furthermore, we will take a look at what effect the flaw in the proof of the algorithm might have on the practical running time of the algorithm Geunes et al. [2009] proposed for the capacitated case.

# Chapter 2

# Uncapacitated lot-sizing

First we will look at the uncapacitated ELSP and the algorithm provided by Van den Heuvel and Wagelmans [2006]. They develop an exact algorithm for the joint pricing and lot-sizing model that determines the optimal price and ordering decisions simultaneously. The algorithm makes use of a heuristic algorithm proposed by Kunreuther and Schrage [1973] to determine the optimal price. For a given price, and therefore demand, the optimal production plan can be found by solving an instance of the ELSP, using the algorithm provided by Wagner and Whitin [1958]. Since the Kunreuther-Schrage procedure is essentially a local search with multiple starting points, it might not find the optimal solution, but instead give two different prices, of which one is lower and the other is higher than the optimal price. Therefore an extra procedure is needed to restart the Kunreuther-Schrage procedure. To this end they apply an idea often attributed to Eisner and Severance [1976]. The Eisner-Severance procedure finds a price inbetween the two different prices where the original Kunreuther-Schrage procedure ends with which the Kunreuther-Schrage procedure can be restarted. This continues until the Kunreuther-Schrage procedure ends up with the same prices for both starting points, which indicates all possible solutions within the interval have been found. One can then determine the optimal price from the obtained terminating prices.

This section will give a description of the original ELSP and the various procedures used in the exact algorithm of Van den Heuvel and Wagelmans [2006], along with the algorithm itself. The algorithm has been implemented and run for several generated test cases to determine the practical running time for the algorithm. Although Van den Heuvel and Wagelmans [2006] propose a $O(T^3 \log T)$ running time for their algorithm, we will implement the algorithm from Wagner and Whitin [1958] such that it takes $O(T^2)$ time, instead of $O(T \log T)$ as proposed by Federgruen and Tzur [1991] and Wagelmans et al. [1992]. This means the algorithm will have a running time of $O(T^4)$, to which we can compare our results.

## 2.1 Problem description

First we will give a description of the original ELSP. We will use the same notation as in Van den Heuvel and Wagelmans [2006]:

| | |
|---|---|
| $T$: | model horizon, |
| $D_t$: | demand in period $t$, |
| $K_t$: | setup costs in $t$, |
| $c_t$: | unit production costs in period $t$, |
| $h_t$: | unit holding costs in period $t$, |
| $x_t$: | production quantity in period $t$, |
| $I_t$: | ending inventory in period $t$. |

The problem can then be formulated as

$$\begin{array}{rll} C(D) = \min & \sum_{t=1}^{T} \left(K_t \delta\left(x_t\right) + c_t x_t + I_t h_t\right) & \\ \text{s.t.} & I_t = I_{t-1} - D_t + x_t & t = 1, \ldots, T, \\ & x_t, I_t \geq 0, & t = 1, \ldots, T, \\ & I_0 = 0, & \end{array}$$

where

$$\delta\left(x\right) = \begin{cases} 0 & \text{for } x = 0 \\ 1 & \text{for } x > 0 \end{cases}$$

and $D$ is the demand vector of length $T$. The objective function of the model is to minimize the total costs, which consists of the setup costs in periods $t$ where there is a production, the production costs of the amount of items made in each period and the holding costs of the amount of items in the ending inventory of each period. The first set of constraints ensures that the ending inventory of period $t$ equals the ending inventory of period $t-1$ minus the demand in period $t$ and plus the amount of items produced in period $t$. The second set of constraints ensures that there are nonnegative production quantities and ending inventory. Finally, the third constraint is to ensure that the starting inventory is empty. This original ELSP can be solved optimally with the algorithm provided by Wagner and Whitin [1958] using dynamic programming in $O(T^2)$ time.

Van den Heuvel and Wagelmans [2006] assume that demand is not deterministic as in the original ELSP, but a deterministic function of the price where the price $p$ is constant over all periods $t$. They use the demand function proposed by Kunreuther and Schrage [1973],

$$D_t\left(p\right) = \alpha_t + \beta_t d\left(p\right),$$

where $d\left(p\right)$ is a differentiable non-increasing function and $\alpha_t, \beta_t \geq 0$. This ensures that demand will not increase if the price increases. This is what you would normally expect in human behaviour, one would not buy more of a product if it becomes more expensive.

Now that we have introduced a price $p$, the objective is not to minimize the total costs as in the original ESLP, but to maximize total profit. The objective can now be formulated as

$$\Pi\left(D\left(p\right)\right) = \max\nolimits_{p>0:D(p)\geq 0}\left\{\sum_{t=1}^{T} pD_t\left(p\right) - C\left(D\left(p\right)\right)\right\},$$

where $C\left(D\left(p\right)\right)$ is the total costs with price dependent demand. Kunreuther and Schrage [1973] show that $C\left(D\left(p\right)\right)$ is a concave piecewise linear function of the so-called 'demand effect' $d\left(p\right)$. This means that each line piece corresponds to a specific production plan. We can then use the heuristic algorithm proposed by Kunreuther and Schrage [1973], together with the procedure of Eisner and Severance [1976] to determine the optimal price for which total profit is maximized.

## 2.2 Methods

The exact algorithm proposed by Van den Heuvel and Wagelmans [2006] makes use of many already existing algorithms. First we have the well-known algorithm from Wagner and Whitin [1958] for obtaining an optimal solution of the uncapacitated ELSP. This algorithm, which uses dynamic programming, runs in $O(T^2)$ time. However, Federgruen and Tzur [1991], Wagelmans et al. [1992] and Aggarwal and Park [1993] have all independently improved this result to $O(T \log T)$. We will take a look at the original algorithm proposed by Wagner and Whitin [1958] instead of the improved algorithms. We will also take a closer look at the procedures from Kunreuther and Schrage [1973] and Eisner and Severance [1976] which are used to determine the optimal price. Finally a pseudo-code of the exact algorithm will be provided.

### 2.2.1 Wagner-Whitin algorithm

For the uncapacitated ELSP, Wagner and Whitin [1958] introduced the dynamic lot-size model, which is a generalization of the economic order quantity model. They provided an algorithm for finding the optimal solution by using dynamic programming. The algorithm basically looks at all periods and determines the total costs if one would decide to start a new production in a period, incurring setup cost $K_t$ and production costs $c_t$, or satisfy demand with current stock in inventory, incurring holding costs $h_t$ from the period when the item was produced until when it is removed from inventory. A new production can only be started when the inventory is empty (zero-inventory property) and therefore we also assume the starting inventory to be equal to zero ($I_0 = 0$).

We can calculate the possible costs, which we will denote by $Z_{kt}$, consisting of the combined production and holding costs of starting a new production in period $k$ that supplies demand from periods $k$ through $t$, by using the expression

$$Z_{kt} = K_k \delta\left(x_k\right) + c_k x_k + \sum_{j=k+1}^{t} D_j \sum_{i=k}^{j} h_i.$$

Using these costs, we can obtain the minimum possible cost for each period $t$

through

$$F_t = \min_{k=1,\dots,t} (Z_{kt} + F_{k-1}) \quad \text{for } t = 1, \dots, T$$

where $F_0 = 0$. One can then use backward induction to determine the optimal production plan. This is done by determining the period $k$ for which the cumulative costs $Z_{kT} + F_{k-1}$ is the smallest, or in other words, find the period $k$ accompanying $F_T$. Then continue finding the production period accompanying $F_{k-1}$ and continue until the production period is equal to the first period in the horizon. It is well known that the Wagner-Whitin algorithm has a worst case running time of $O(T^2)$. Even though it has been shown that this can be reduced to $O(T \log T)$, we implement the algorithm the old-fashioned way.

**Example** Consider a four-period problem with $K = (10, 7, 4, 1)$, $c = (1.2, 0.6, 0.6, 0.4)$, $h_t = h = 0$ and $D = (4, 6, 9, 2)$. If we would start production in period $t = 1$ and only produce the amount for that period, which is $D_1 = 4$, this would incur the costs $Z_{11} = 10 + 1.2 \cdot 4 = 14.8$. However, if we would not only produce the amount demanded in period $t = 1$, but rather the amount in both $t = 1$ and $t = 2$, we would have $Z_{12} = 10 + 1.2 \cdot (4 + 6) = 22.0$. If the demanded amount of period 2 was to be produced in the same period, the cost would be $Z_{22} = 7 + 0.6 \cdot 6 = 10.6$. The cumulative cost however would be $Z_{22} + F_1 = Z_{22} + Z_{11} = 25.4$, as demand in the first period needs to be satisfied as well and $Z_{11}$ is the only candidate for $t = 1$ and therefore immediately the minimum. One can see that this is actually less cost efficient as if we would produce the amount for both periods in $t = 1$. We can calculate these cumulative costs for each period and the results are shown in Table 2.1. Now backward induction can be applied to determine the optimal production plan. We can see that the minimum possible cost for period $t = T$ is $F_4 = Z^*_{24} = 32.0$. Now we need to find $F_{2-1} = F_1$, which is $Z_{11} = 14.8$ and we can terminate the procedure as we have arrived at the first period. The optimal production plan thus consists of producing in two periods, namely $t = 1$ and $t = 2$, producing $D_1 = 4$ in $t = 1$ and $D_2 + D_3 + D_4 = 17$ in $t = 2$. The total cost of this production plan is equal to $F_4 = 32.0$.

| $k \backslash t$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 14.8 | 22.0 | 32.8 | 35.2 |
| 2 | | 10.6 | 16.0 | 17.2 |
| 3 | | | 9.4 | 10.6 |
| 4 | | | | 1.8 |

| $k \backslash t$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 14.8 | 22.0 | 32.8 | 35.2 |
| 2 | | 25.4 | 30.8 | 32.0 |
| 3 | | | 31.4 | 32.6 |
| 4 | | | | 32.6 |

Table 2.1: Wagner-Whitin example costs and cumulative costs

### 2.2.2 Kunreuther-Schrage algorithm

Kunreuther and Schrage [1973] provide a heuristic algorithm which determines the optimal price for which the total profit gets maximized when the demand function of the ELSP is a deterministic function of the price. The algorithm works as follows:

1. Set $i := 0$, start with some initial price $p_i$ and set $\Pi_i := -\infty$.
2. Set $i := i + 1$.
3. Calculate the demand vector $D(p_{i-1})$ and let $S_i$ be the optimal production plan corresponding to problem $C(D(p_{i-1}))$.
4. Calculate the price $p_i$ which maximizes the net profit $\Pi_i := \sum_{t=1}^{T} R_t(p_i) - S_i(d(p_i))$.
5. If $\Pi_i - \Pi_{i-1} > 0$, then go to step 2.
6. The terminating price is $p_i$ with corresponding net profit $\Pi_i$.

In step 3 of the algorithm, we can use the aforementioned Wagner-Whitin algorithm to find an optimal production plan given the demand $D(p_{i-1})$. In step 4 we want to determine the optimal price given a fixed production plan, which we obtain from step 3. We assume that this requires less time than solving the ELSP.

Kunreuther and Schrage [1973] show that the algorithm does not skip over any optimum while determining the terminating price. This means that of two consecutive prices generated by the Kunreuther-Schrage algorithm, the latter one will always generate more profit than any price inbetween the two prices. Therefore we can provide the algorithm with a lower and an upper bound on the optimal price $p^*$, $p_L$ and $p_U$ respectively. This will result in terminating prices $p_L^*$ and $p_U^*$, for which the inequality $p_L^* \leq p^* \leq p_U^*$ will hold. Thus if the terminating prices $p_L^*$ and $p_U^*$ are equal, we have found our optimal price $p^*$. Since this algorithm is a local search, it may occur that $p_L^*$ and $p_U^*$ are not equal. This means that the optimal price $p^*$has not been found yet and has to lie in between the terminating prices. Therefore we need to restart the KS-procedure with a suitable price $\bar{p}$ that satisfies $p_L^* \leq \bar{p} \leq p_U^*$. To determine this new price, we make use of an idea from Eisner and Severance.

### 2.2.3 Eisner-Severance method

The method from Eisner and Severance [1976] provides a way for us to find a suitable $\bar{p}$ that satisfies $p_L^* \leq \bar{p} \leq p_U^*$. This method works on a piecewise linear concave function and since our objective function $C\left(D\left(p\right)\right)$ satisfies these conditions, we can apply this method to our problem. The method starts out with two instances of the dependent variable of the function, which are $p_L^*$ and $p_U^*$ in our case. Assume we have some algorithm that solves the optimization problem for a fixed $p$, which is the Wagner-Whitin algorithm in our case. Assume that we want to find all breakpoints in the interval $[p_L^*, p_U^*]$, because if we know there are breakpoints within this interval, it means that there exists at least one line piece which the KS-procedure has not reached yet due to termination. The Eisner-Severance method can now be described as follows:

1. Apply the algorithm on the optimization problem with $p = p_L^*$ and $p = p_U^*$.
2. Calculate the intersection point $\bar{p}$ of the lines corresponding to $p_L^*$ and $p_U^*$, say $l_L^*\left(p\right)$ and $l_U^*\left(p\right)$.
3. Perform the algorithm on the optimization problem with $p = \bar{p}$.
4. If the objective value is equal to $l_L^*\left(p\right)$ and $l_U^*\left(p\right)$, there is no new line piece between $p_L^*$ and $p_U^*$. Otherwise, repeat the procedure recursively (go to step 1) with $p_L^* := p_L^*$ and $p_U^* := \bar{p}$ and with $p_L^* := \bar{p}$ and $p_U^* := p_U^*$.

A graphical representation of this method is shown in Figure 2.2.1. One can see that the piecewise linear concave function consists of three line pieces, namely

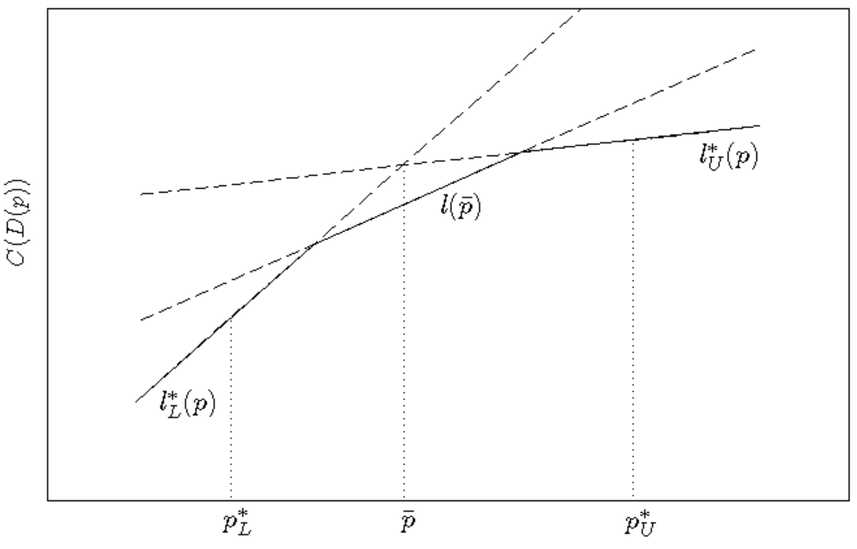

Figure 2.2.1: The Eisner-Severance method

$l_L^*(p)$, $l(\bar{p})$ and $l_U^*(p)$. If we apply the Wagner-Whitin algorithm on $p_L^*$ and $p_U^*$, we obtain the optimal production plans $S_L^*$ and $S_U^*$ with the corresponding lines $l_L^*(p)$ and $l_U^*(p)$ respectively. Using these lines, we can calculate the intersection point $\bar{p}$ and apply the Wagner-Whitin algorithm on this price to find line $l(\bar{p})$. Since the objective value will not be equal to $l_L^*\ (p)$ and $l_U^*\ (p)$, we can repeat this procedure with $p_L^* := p_L^*$ and $p_U^* := \bar{p}$ and with $p_L^* := \bar{p}$ and $p_U^* := p_U^*$. However, we will not find new line pieces and the Eisner-Severance procedure terminates. If there would be $nBP$ breakpoints, the Wagner-Whitin algorithm needs to be applied $2nBP - 1$ times.

### 2.2.4 Exact algorithm

Van den Heuvel and Wagelmans [2006] propose an exact algorithm making use of the aforementioned procedures to determine an optimal price and ordering decisions simultaneously. The algorithm is as follows:

– Start the KS-procedure with $p_L < p_U$ and let $p_L^*$and $p_U^*$ be the terminating prices

```
– If $p_L^* = p_U^*$ then
    – Stop; the optimal price is $p_L^*$
– Else
    – Call Optim($p_L^*, p_U^*$)
– End if
– Function Optim($p_L, p_U$)
    – $\bar{p} = ES\left(S_L, S_U\right)$
    – If $S_L\left(d\left(\bar{p}\right)\right) = S_U\left(d\left(\bar{p}\right)\right)$ then
        ∗ End function
    – Else
        ∗ Perform the KS-procedure with $p = \bar{p}$ and let $\bar{p}^*$ be the terminating price
        ∗ Check whether the found optimum is larger than the previous found optimum and store the corresponding price and production plan.
        ∗ If $\bar{p}^* > \bar{p}$, then
            · Call Optim($p_L, \bar{p}$); Call Optim($\bar{p}^*, p_U$)
        ∗ Else
            · Call Optim($p_L, \bar{p}^*$); Call Optim($\bar{p}, p_U$)
        ∗ End if
    – End if
– End function
```

The production plans $S_L$ and $S_U$ correspond to the prices $p_L$ and $p_U$ respectively. The function ES applies one iteration of the ES method and returns a price $\bar{p}$ that satisfies $p_L \leq \bar{p} \leq p_U$. When it cannot find new production plans between $S_L$ and $S_U$, the Optim function terminates. If it does find one, the KS-procedure gets applied on the price $\bar{p}$. The algorithm then calls the Optim function recursively for the new price bounds. If the Optim function terminates while it has found a new production plan, the corresponding $\bar{p}$ will be noted down as a breakpoint in the cost function. If the main Optim function terminates, we will have all breakpoints between $p_L$ and $p_U$.

## 2.3 Results

According to Van den Heuvel and Wagelmans [2006] the maximum number of breakpoints one can find is equal to $\frac{1}{2}T(T-1)$. However, because of a flaw in the proof, it is unsure whether only the proof is incorrect or also the theorem itself. For this purpose, we will generate a vast amount of test cases according to different criteria. We want to look for difficult instances where the amount of breakpoints will be large. If the number of breakpoints for even one of the cases exceeds the expected maximum number of breakpoints, we can be sure that the theorem is flawed. If we are unable to do so, we cannot say anything conclusive about the theorem, until the proof gets corrected, if possible. In this section we will first observe the computation time of the algorithm as we let $T$ vary, comparing the practical running time with the theory. We will discuss how we choose our parameters for our test cases, after which we will concentrate on the case of $T = 5$ and apply the algorithm to the test cases to look for possible instances where the amount of breakpoints exceeds the theoretical amount. We choose this particular $T$ because the proof is supposedly flawed for $T \geq 5$. All of the results have been obtained by using the 64-bit version of MATLAB on a desktop computer with a quad-core processor at 3.2 GHz and 4GB RAM.

### 2.3.1 Computation time for various $T$

First we will look at the development of the amount of breakpoints and the computation time of one case for various $T$. For this purpose we will use an example that generates a large number of breakpoints. We will use the same parameters as in the example proposed by Van den Heuvel and Wagelmans [2006]:

$$\begin{aligned} &D_t(p) = D + p && \text{for } t = 1, \dots, T, \\ &K_t = (T - t + 1)K && \text{for } t = 1, \dots, T, \\ &c_t = T^{T-t+1} && \text{for } t = 1, \dots, T, \end{aligned}$$

where we will use $D = \alpha_t = 0$, $\beta_t = 1$ and $K = 100$. We will use $d(p) = p$ rather than $-p$ to mirror the objective function such that the breakpoints will occur more often at smaller $p$ than at larger $p$, which happens when $d(p)$ is a non-increasing function. We run the Optim-function using these parameters and $T = 4, \dots, 20$. The results are shown in Table 2.2.

We can see that the number of breakpoints found by the function is equal to the expected number of breakpoints according to Van den Heuvel and Wagelmans [2006] up until $T = 11$. We can see for $T \geq 11$ that the number of breakpoints found is lower than the expected number of breakpoints. This is because the breakpoints will lie closer to each other as $T$ becomes larger. The algorithm will eventually falter due to numerical imprecision as breakpoints lie too close to each other. In order to let the function continue without ending up in an endless recursion, a condition is implemented such that the recursion will only continue when the calculated intersection point is different from both the lower and upper bound on $p$. However, this does have a disadvantage as some breakpoints will not be calculated, resulting in the aforementioned loss of breakpoints. From $T = 16$ onwards the loss of breakpoints gets immense and the number of breakpoints found even decreases as $T$ increases.

If we look at the computation time, we can see that it behaves as one would expect. The computation time increases as $T$ increases and the problem gets more complex. However, at $T = 16$ this trend halts and the computation time does not increase that rapidly anymore. This is due to the numerical imprecision of MATLAB and the computer which results in premature termination of the function and therefore a loss of breakpoints. If the algorithm actually runs in $O(T^4)$ time, we should see a multiplication of the computation time by $2^4 = 16$ if we double $T$. If we compare the computation times of $T = (4, 5, 6, 7)$ with the times of $T = (8, 10, 12, 14)$ we can see a multiplication of $(3.2, 7.6, 11.0, 12.2)$ respectively. This stays well within the multiplication factor of 16. A plot of the computation time against $T$ can be seen in Figure 2.3.1. We can clearly see that

| $T$ | $\frac{1}{2}T(T-1)$ | Breakpoints found | Computation time in seconds |
|---|---|---|---|
| 4 | 6 | 6 | 0.0219 |
| 5 | 10 | 10 | 0.0205 |
| 6 | 15 | 15 | 0.0276 |
| 7 | 21 | 21 | 0.0436 |
| 8 | 28 | 28 | 0.0703 |
| 9 | 36 | 36 | 0.1073 |
| 10 | 45 | 45 | 0.1562 |
| 11 | 55 | 53 | 0.2260 |
| 12 | 66 | 65 | 0.3036 |
| 13 | 78 | 78 | 0.4018 |
| 14 | 91 | 91 | 0.5313 |
| 15 | 105 | 104 | 0.6855 |
| 16 | 120 | 102 | 0.7607 |
| 17 | 136 | 103 | 0.8348 |
| 18 | 153 | 101 | 0.9009 |
| 19 | 171 | 96 | 0.9548 |
| 20 | 190 | 95 | 1.0144 |

Table 2.2: Performance of the Optim function for various T

the computation time does not increase at the same rate anymore from $T = 16$ onwards.

### 2.3.2 Generating test cases

To be able to have a widely diverse collection of test cases, we will vary many parameters in the model. Since production plans are dependent on costs, it is obvious to let the setup costs $K_t$, unit production costs $c_t$ and holding costs $h_t$ vary. However, the holding costs $h_t$ can be incorporated in the unit production costs $c_t$ by using

$$c_t^* = c_t + \sum_{i=t}^{T} h_i$$

which gives the advantage of having one parameter less. One has to take into account that this will inflate the objective function, but will not affect the optimal production plan. The excess on costs can also be easily subtracted at the end if one wishes to have the proper costs. Next to the cost parameters, we also want to

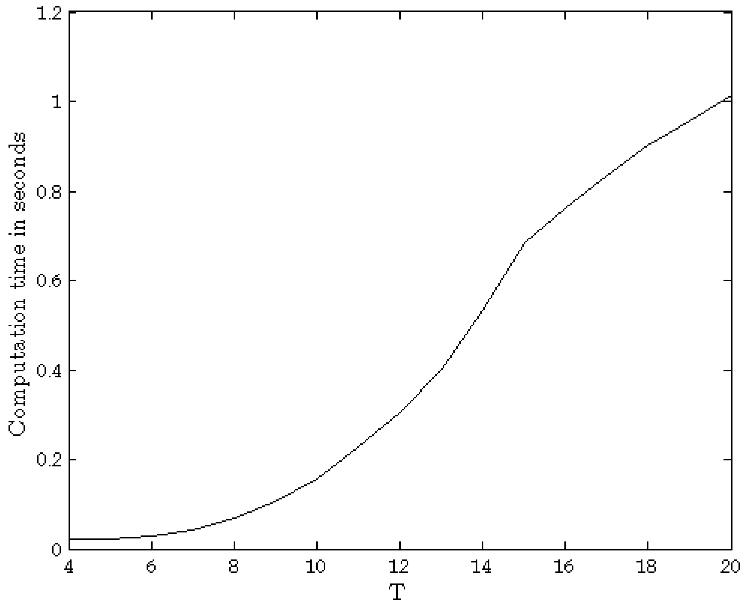

Figure 2.3.1: Graph of the computation time plotted against T

vary the demand function. Therefore we will also vary $\alpha_t$ and $\beta_t$. The term $d\,(p)$ in the demand function will be set to $p$, rather than $-p$, which is mentioned in the example used by Van den Heuvel and Wagelmans [2006].

The four parameters, $K_t$, $c_t$, $\alpha_t$ and $\beta_t$ will be randomly generated, with a predetermined range in which they have to fall. The ranges vary from $[0, 10^0]$ up to $[0, 10^4]$, increasing the upper bound by a factor 10 each time. We will also look at certain sorting orders in the parameters, like increasing or decreasing over time. This simulates cases where, for example, the demand decreases over time due to deterioration. One can think of computers for example. These orders might have a large influence on the amount of breakpoints.

### 2.3.3 Simulating test cases

As we have seen in 2.3.1, computation time increases as $T$ gets larger. Therefore we will focus on a relatively small $T$ to find us a subset of test cases which are prone to having a large number of breakpoints. We choose $T = 5$ as this is the first $T$ where the proof of the algorithm of Van den Heuvel and Wagelmans [2006] shows a flaw and it should have relatively short computation times. We will draw three random number sequences in each range for each parameter and combine them with each other to construct test cases on which we can use the exact algorithm. Since we have five ranges, from $[0, 10^0]$ to $[0, 10^4]$, from which we draw three numbers each for each of the four parameters, we end up with $(5 \cdot 3)^4 = 50,625$ test cases. We can leave the randomly drawn number sequences for each parameter unsorted or we can sort them either ascending or descending. This will be done to detect if any particular behaviour of a certain parameter has a substantial influence on the amount of breakpoints we can find. Since we have four parameters which we can sort, we end up with $2^4 = 16$ different ways of sorting the parameters. For each different way of sorting new parameters will be generated. We will also look at what happens when we leave the parameters unsorted. The results can be found in Table 2.3. For the parameters, an $u$ means the parameter is unsorted, $d$ stands for descending, whereas $a$ stands for ascending.

From the results we can see that leaving the parameters unsorted results in only two cases where the amount of breakpoints is six. This result is rather mediocre, but it is what one would expect, as the parameters are completely random. What is interesting is that the way the unit production costs $c$ are constructed have a large impact on the amount of breakpoints we can find. If we sort them descending, we find a substantial amount of cases with a rather high amount of breakpoints, whereas if we sort them ascending, we never find a breakpoint in any case whatsoever. This can be explained by the fact that if the unit production costs increase during time, one would simply produce everything in the first period, where it is the cheapest to produce items, since there is no capacity on the amount which can be produced. This way the total demand in all periods will be satisfied and there will be only one setup cost, as there is no need to pay any other setup cost other than in the first period. Even if the setup costs of later periods are cheaper than that of the first period, one has to produce something in the first period, as demand is nonzero in every period and one would incur the setup cost in the first period no matter what.

Unfortunately we can not conclude anything on the other parameters as their performances are rather close and none of them stand out. Looking at the computation times, we can only conclude that the cases where $c$ was sorted ascending

| $\alpha$ | $\beta$ | $K$ | $c$ | Max breakpoints | Number of cases | Computation time (sec) |
|---|---|---|---|---|---|---|
| u | u | u | u | 6 | 2 | 221 |
| d | d | d | d | 8 | 1 | 255 |
| a | d | d | d | 8 | 3 | 248 |
| d | a | d | d | 7 | 130 | 266 |
| a | a | d | d | 7 | 50 | 256 |
| d | d | a | d | 6 | 45 | 264 |
| a | d | a | d | 7 | 1 | 255 |
| d | a | a | d | 7 | 33 | 265 |
| a | a | a | d | 9 | 3 | 262 |
| d | d | d | a | 0 | 50625 | 214 |
| a | d | d | a | 0 | 50625 | 214 |
| d | a | d | a | 0 | 50625 | 215 |
| a | a | d | a | 0 | 50625 | 214 |
| d | d | a | a | 0 | 50625 | 213 |
| a | d | a | a | 0 | 50625 | 214 |
| d | a | a | a | 0 | 50625 | 216 |
| a | a | a | a | 0 | 50625 | 219 |

Table 2.3: Results for test cases with several ways of sorting the parameters

took less time compared to where $c$ was sorted descending and even when all the parameters were unsorted. This can be explained by the fact that the algorithm does not encounter any breakpoints, resulting in an early termination of the function.

These results were obtained from a relatively small sample with only $50,625$ cases per scenario. Many more cases need to be generated to be able to find a case which possibly has more than the expected 10 breakpoints. From the previous results we can say for certain that we do not need to take a look anymore at the scenario where the unit production costs are sorted ascending. So now we are only concerned about the scenarios with the unit production costs sorted descending. We will hold on to the five ranges we had before, from $[0, 10^0]$ to $[0, 10^4]$, but we will enlarge the amount of numbers drawn in each range from three to five for each of the four parameters. This change translates into an increase from $50,625$ cases to $(5 \cdot 5)^4 = 390,625$ test cases per scenario. We will be using the same eight scenarios where the remaining parameters, which are $\alpha$, $\beta$ and $K$, can be sorted

| $\alpha$ | $\beta$ | $K$ | Max breakpoints | Number of cases | Computation time (sec) |
|---|---|---|---|---|---|
| d | d | d | 8 | 11 | 12025 |
| a | d | d | 9 | 4 | 13294 |
| d | a | d | 8 | 2 | 12818 |
| a | a | d | 7 | 484 | 12982 |
| d | d | a | 7 | 2 | 13134 |
| a | d | a | 7 | 4 | 13866 |
| d | a | a | 9 | 6 | 14052 |
| a | a | a | 8 | 14 | 11693 |

Table 2.4: Results for test cases with unit production costs sorted descending

either ascending or descending. The results can be found in Table 2.4.

We can see that there are two scenarios where there are cases with nine breakpoints, being a/d/d and d/a/a respectively. However, these scenarios did not yield cases with more than eight breakpoints in our previous results, where they merely yielded cases with eight and seven breakpoints respectively. Furthermore, the scenario a/a/a which yielded a maximum of nine breakpoints in certain cases in our previous results, merely yields a maximum of eight breakpoints in this extended test. Looking at these and the other scenarios, we can unfortunately not discover a pattern regarding the sorting of the parameters $\alpha$, $\beta$ and $K$ and the amount of breakpoints found in the test cases.

Since it seems that sorting the three parameters $\alpha$, $\beta$ and $K$ does not affect the maximum amount of breakpoints found in the generated cases, we will now let these three parameters be completely random and unsorted. We will keep the five ranges the same and we will again draw five random numbers for each of the parameters. The unit production costs will also be sorted descending again. However, instead of conducting this test only once, we will perform this test ten times, generating a total of $3,906,250$ test cases. The test results are displayed in Table 2.5.

From the results we can see that only three cases out of the $3,906,250$ cases have nine breakpoints. Inspecting these three cases, there was no pattern to be found in the parameters. The three cases had nothing in common, aside from the unit production costs being sorted descending. It does seem we had more success with the amount of breakpoints in the cases where we sorted the parameters.

| Run | Max breakpoints | Number of cases | Computation time (sec) |
|---|---|---|---|
| 1 | 9 | 1 | 12179 |
| 2 | 8 | 20 | 11987 |
| 3 | 8 | 28 | 13757 |
| 4 | 8 | 20 | 14092 |
| 5 | 8 | 6 | 13644 |
| 6 | 9 | 1 | 12091 |
| 7 | 9 | 1 | 13299 |
| 8 | 8 | 10 | 13291 |
| 9 | 8 | 9 | 16749 |
| 10 | 7 | 169 | 14885 |

Table 2.5: Results for test cases with unsorted $\alpha$, $\beta$ and $K$ and descending $c$

However, there is unfortunately no sorting pattern discovered which leads to consistent results regarding the amount of breakpoints, aside from sorting the unit production costs descending.

### 2.3.4 Simulating test cases with $\alpha = 0$

If we look at the parameters we used in the example case for the computation time of the exact algorithm for various $T$, where we found the same number of breakpoints as expected by the theorem of Van den Heuvel and Wagelmans [2006] up to $T = 10$, one can see we used $D = \alpha_t = 0$, while the other parameters are nonzero. If we set $\alpha_t = 0$, it means demand is only dependent of $\beta_t$ and $p$. There will be no scalar increase in demand, which means sudden jumps in demand over time will occur less often. Demand will increase or decrease gradually due to mere $\beta_t$ as $p$ progresses, which might make for subtle changes in production plans. This might result in more breakpoints in the objective function.

We will generate test cases as before, with five ranges from $[0, 10^0]$ to $[0, 10^4]$. However, since we will only have three parameters now instead of four, being $\beta$, $K$ and $c$, we can increase the number of random numbers drawn in the specific ranges. We will draw ten random numbers from each range for each of the three parameters, resulting in $(5 \cdot 10)^3 = 125,000$ test cases. We will sort the parameters as well to see which scenario will generate cases with the most breakpoints. The parameters $\beta$ and $K$ will be sorted both ascending and descending, while we keep

| $\beta$ | $K$ | Max breakpoints | Number of cases | Computation time (sec) |
|---|---|---|---|---|
| d | d | 10 | 4 | 16030 |
| a | d | 9 | 13 | 18014 |
| d | a | 8 | 7 | 16451 |
| a | a | 9 | 21 | 15709 |

Table 2.6: Results for test cases with no $\alpha$ and unit production costs sorted descending

| $\alpha$ | $\beta$ | $K$ | $c$ | Max breakpoints | Number of cases | Computation time (sec) |
|---|---|---|---|---|---|---|
| 0 | d | d | d | 10 | 57 | 126728 |

Table 2.7: Results for ten million test cases with no $\alpha$ and $\beta$, $K$ and $c$ sorted descending

the unit production costs $c$ sorted descending, as we have found that sorting $c$ ascending will result in no breakpoints in the objective function at all. This makes for four scenarios and we will run each scenario ten times, such that we have a total of $1,250,000$ test cases for each scenario. The results can be found in Table 2.6.

From the results we can see that we have found cases with ten breakpoints, which is the maximum number of breakpoints proposed by Van den Heuvel and Wagelmans [2006] for $T = 5$. These cases were found in the scenario where both $\beta$ and $K$ are sorted descending. This result actually coincides with the example of the worst case scenario used by Van den Heuvel and Wagelmans [2006]. We can observe that these cases are quite rare as we have found only four cases out of a total of $1,250,000$ test cases.

Now that we have observed cases which contain the proposed maximum number of breakpoints for $T = 5$, we can search for a possible case where there might be even more breakpoints. Therefore we will generate a total of ten million test cases where all three parameters $\beta$, $K$ and $c$, again drawn from $[0, 10^0]$ to $[0, 10^4]$, are sorted descending. The results are presented in Table 2.7. We can observe that unfortunately we could not find a case with more than ten breakpoints among the ten million test cases.

## 2.4 Conclusion

Unfortunately we could not find any case in our simulations which contained more than ten breakpoints, the number that Van den Heuvel and Wagelmans [2006] suggested to be the maximum amount of breakpoints for $T = 5$, which would disprove their theorem. It seems that only the way the unit production costs are sorted have a consistent influence on the amount of breakpoints found. This is because if unit production costs increase during time, everything can be produced in the first period. The way the parameters $\alpha$, $\beta$ and $K$ are sorted seems to have no consistent effect on the amount of breakpoints found. However, it does seem that leaving them unsorted makes it harder to find cases with a high amount of breakpoints. If we set $\alpha_t = 0$, it seems that sorting the other parameters, $\beta$ and $K$, does have effect. We have only found cases which contained the maximum of ten breakpoints in the scenario where $\alpha = 0$ and all the other parameters are sorted descending. This could have been expected if we take a look at the example of the worst case scenario presented by Van den Heuvel and Wagelmans [2006], which supports these properties, except for $\beta$ which is constant.

In the end, we were unable to find a case which has more than ten breakpoints, therefore we could not disprove the theorem of Van den Heuvel and Wagelmans [2006]. We have seen that there are cases which would produce ten breakpoints, but these are worst case scenarios and we have seen that they do not appear that often in simulations or in practice. To disprove the theorem of Van den Heuvel and Wagelmans [2006], one has to find a case which yields more than ten breakpoints and thus far this has yet to happen. Until then, we are only sure that there is a flaw in the proof of Van den Heuvel and Wagelmans [2006], but we are unable to say anything about the credibility of the theorem itself.

# Chapter 3

# Capacitated lot-sizing

The uncapacitated ELSP is a fairly simple model and rather easy to understand for the general public. However, in practice it is rarely the case that there is no restriction on the amount one can produce or order. One can think of limited storage room in the case of production or a limited amount of trucks and the capacity of said trucks in the case of ordering. Capacities are a common hindrance when it comes to logistics and it makes a problem much harder compared to its uncapacitated counterpart. Due to the complexity capacities add to a problem, many resort to heuristics and algorithms rather than trying to obtain an optimal solution. More often than not it is computationally too demanding to obtain an optimal solution and if it is possible, solving the problem often still takes quite some time.

In this section we will take a closer look at a paper by Geunes et al. [2009], which applies the algorithm from Van den Heuvel and Wagelmans [2006] to the capacitated case with slight modifications. Instead of using the Wagner-Whitin algorithm to solve cases, we have to take a look at a method introduced by Florian and Klein [1971] which gives a heuristic solution to the capacitated ELSP. Furthermore, we will revisit the Kunreuther-Schrage method and the Eisner-Severance procedure as they need to be altered slightly to be applicable to the capacitated problem. The characteristics of a solution of a capacitated ELSP is not like that of its uncapacitated counterpart where one would obtain a concave piecewise linear function of the price against the costs. If one would plot the price against the costs in the capacitated case, one would actually obtain a piecewise function consisting of concave piecewise linear functions as in the uncapacitated case. We will take a closer look at this phenomenon and explain why this occurs. We will also generate test cases for the capacitated ELSP and the results will be compared to what one would expect the results to be according to the article of Geunes et al. [2009].

## 3.1 Problem description update

Capacities introduce a new dimension to the problem as one has to take into account that one may not produce the amount one wants as it might exceed the capacity. Therefore one has to plan ahead of time and produce the amount that would not be able to be produced because of capacity at an earlier point in time. This of course generates holding costs from the moment it is produced until the moment it gets sold, which would often be the point in time where there will be more demand than there is capacity to produce it all in one go. Because holding costs accumulate over time, one also does not want to produce the goods too early from the time they would be demanded. Producing goods earlier and having a stock of them, such that demand can be met even when the demand exceeds the production capacity seems like a safe option, but this can eventually become very pricy. So next to the trade-off of production costs and holding costs of the original ELSP, capacities introduce another trade-off of the reliability of meeting demand and holding costs.

One could think of capacities in many ways, such as production capacity, where there are limited resources whether it be raw materials or labour, storage capacity, where a warehouse has a certain size, or transport capacity, where ships and trucks have limited space. Capacities can be either fixed or variable over time. One can think of a warehouse in which a factory has to store its produced goods as a fixed capacity, as its size would not change frequently over time, merely every now and then when the company decides to expand its warehouse. As for variable capacity, one could think of storage space rented from a third party company, which has to provide storage space to other companies as well, so that the amount of space for every company will vary every period their contract is renewed.

In this paper, we will concentrate on the case of fixed capacities, thus the capacity remains the same over time. Therefore we will introduce a new parameter to the problem which is independent of time:

| | |
|---|---|
| $C$: | capacity of the storage (or transportation device) which cannot be exceeded by the production quantity. |

We also need to add a restriction to the original problem which incorporates this capacity, such that the production quantity does not exceed the capacity. The problem can now be formulated as

$$\begin{aligned} C(D) = \min \quad & \textstyle\sum_{t=1}^{T} \left(K_t \delta\left(x_t\right) + c_t x_t + I_t h_t\right) & \\ \text{s.t.} \quad & I_t = I_{t-1} - D_t + x_t & t = 1, \ldots, T, \\ & 0 \leq x_t \leq C & t = 1, \ldots, T, \\ & x_t, I_t \geq 0, & t = 1, \ldots, T, \\ & I_0 = 0. & \end{aligned}$$

## 3.2 Methods

Since the exact algorithm from Van den Heuvel and Wagelmans [2006] can not be applied immediately to the capacitated lot-sizing problem, it needs to be adjusted accordingly. First we will take a look at an algorithm developed by Florian and Klein [1971], which can solve equal-capacity lot-sizing problems in $O(T^4)$ time. Then we will take a look at the cost function as it differs from the cost function we know from the uncapacitated case. Since the structure of the cost function is different from the uncapacitated case, we need to adjust the exact algorithm such that it takes this into consideration. Furthermore, because the Florian-Klein algorithm is used rather than the Wagner-Whitin algorithm, the Eisner-Severance method can be simplified. These changes to the exact algorithm can be found at the end of this section.

### 3.2.1 Florian-Klein method

Florian and Klein [1971] developed a dynamic programming shortest route algorithm for problems with a fixed capacity in each period. They consider both the nonbacklog as the backlog cases, where backlogging means that a customer will get its order or a part of it at a later time than when they asked for it. One can think of the case where someone wants to buy a product, but the store does not have it in stock anymore and it has to be reserved or ordered, for the customer to pick it up at a later time. We will only look at the case where there is no backlogging, as we want customers to receive their orders on time.

The optimal production plan in the capacitated case consists of independent subplans in which the inventory level is nonzero in every period, except the last period, where it is equal to zero. This is similar to the Wagner-Whitin algorithm in the uncapacitated case where a new production can only be started when the inventory is empty and therefore the starting inventory has to be empty ($I_0 = 0$). Furthermore, within each subplan the production level will be either zero or at capacity, except for at most one period, in which it will be less than capacity. The intuition behind this is to utilize production periods at its fullest and having at most one period to produce the remainder, if any.

As in Florian and Klein [1971], we will call a period $t$ a regeneration point if $I_t = 0$, and we will define the production sequence $S_{uv}$ as a subset of a feasible production plan $X$ that includes the components of $X$ for all periods between the two consecutive regeneration points $u$ and $v$:

$$S_{uv} = \{x_i, i = u+1, \dots, v | I_u = 0 = I_v; I_i > 0 \text{ for } u < i < v\}$$

where $0 \leq u < v \leq n$. Since $I_0 = I_n = 0$, at least one production sequence $S_{0n}$ exists. Since the production level can be zero, at less than capacity or at capacity, we have three values for each production level: $0, \epsilon, C$. Because of the fact that there is only at most one period in which the production level will be at less than capacity ($\epsilon$) in an optimal production sequence, there will be $k = \lfloor \frac{\sum_{i=u+1}^{v} D_i}{C} \rfloor$ periods in which the production level will be equal to $C$. The cumulative amount $X_j = \sum_{i=u+1}^{j} x_i, j = u+1, \dots, v$ produced over periods $u+1$ to $j$ can only be of values $\{0, \epsilon, C, C+\epsilon, \dots, kC, kC+\epsilon\}$.

With this, one can construct an acyclic network with vertices corresponding to the possible values of $X_j$ for each $j = u+1, \dots, v$ and with directed edges $(X_j, X_{j+1})$ defined as follows:

– If $X_j = mC$, where $m = 0, 1, \dots, k$, then the terminal vertices of the edges have values $X_{j+1} = X_j + e$ with $e = \{0, \epsilon, C\}$;

- If $X_j = mC + \epsilon$, where $m = 0, 1, \ldots, k$, then the terminal vertices of the edges have values $X_{j+1} = X_j + g$ with $g = \{0, C\}$.

This means that each vertex value contains information about whether the quantity $\epsilon$ has been produced yet or not. If the second case of vertices is once reached, it will remain in this class, as there is at most one period in which the production will be at less than capacity ($\epsilon$) and it can never return to the first case. The costs for each vertex can be calculated by determining the cheapest option out of the various possibilities a certain vertex has to be formed. For example, a vertex $X_3 = C + \epsilon$, can either be formed by producing $C$ while $X_2 = \epsilon$, or by producing $\epsilon$ while $X_2 = C$, or even by producing 0 while $X_2 = C + \epsilon$. The option with the least costs will be determined and its cost will be assigned to the vertex $X_3$. Vertices with values associated with nonfeasible inventory levels, i.e.

$$X_j : X_j - \sum_{i=u+1}^{j} r_i \leq 0, \; j = u+1, \ldots, v-1$$

will be assigned an arbitrarily large cost, such that these vertices will not be chosen in the optimal production sequence. The optimal production sequence and its cost can be determined by using dynamic programming recursion using the periods $0, \ldots, n$ as states:

$$f_0 = 0$$
$$f_v = \min_{u=v-1,\ldots,0} \{d_{uv} + f_u\}, \quad v = n, \ldots, 1$$

where $f_v$ is the cost associated with an optimal production plan over periods $1, \ldots, v$ and $d_{uv}$ is the cost associated with following an optimal plan over periods $u+1, \ldots, v$, where $I_u = I_v = 0$ and $I_t > 0$ for all $t = u+1, \ldots, v-1$. This solution approach takes order $O(T^4)$ time.

**Example** Consider the same four-period problem instance used in the example in section 2.2.1 with $K = (10, 7, 4, 1)$, $c = (1.2, 0.6, 0.6, 0.4)$, $h_t = h = 0$ and $D = (4, 6, 9, 2)$. In addition we will have a constant capacity of $C = 7$ in all

| *u*\|*v* | *1* | *2* | *3* | *4* |
|---|---|---|---|---|
| *0* | 14.8 | 27.2 | 35.4 | 37.8 |
| *1* | | 10.6 | $+\infty$ | $+\infty$ |
| *2* | | | $+\infty$ | $+\infty$ |
| *3* | | | | 1.8 |

| *u*\|*v* | *1* | *2* | *3* | *4* |
|---|---|---|---|---|
| *0* | $\epsilon$ | $C\epsilon$ | $\epsilon CC$ | $\epsilon cc0$ |
| *1* | | $\epsilon$ | $-$ | $-$ |
| *2* | | | $-$ | $-$ |
| *3* | | | | $\epsilon$ |

Table 3.1: Florian-Klein algorithm example costs and production schemes

| *u*\|*v* | *1* | *2* | *3* | *4* |
|---|---|---|---|---|
| *0* | 14.8 | 27.2 | 35.4 | 37.8 |
| *1* | | 25.4 | $+\infty$ | $+\infty$ |
| *2* | | | $+\infty$ | $+\infty$ |
| *3* | | | | 37.2 |

Table 3.2: Florian-Klein algorithm example cumulative costs

periods. The cost of the production sequence $S_{01}$ can be calculated the same way as before, where $\epsilon = 4$ is produced in period $t = 1$ resulting in $10 + 1.2 \cdot 4 = 14.8$. Now for $S_{02}$ we need to produce a total of 10. Since the capacity $C = 7$, it is not possible to produce it all in $t = 1$ as we did before in the uncapacitated case. Now there are two possibilities, either produce $C = 7$ in $t = 1$ and $\epsilon = 3$ in $t = 2$, or produce $\epsilon = 3$ in $t = 1$ and $C = 7$ in $t = 2$. However, since the demand in $t = 1$ is equal to $D_1 = 4$, it is not sufficient to produce $\epsilon = 3$. Therefore this solution is infeasible and will be assigned a cost of $+\infty$, such that it will be never selected as the best solution. The first possibility will incur a total cost of $10 + 1.2 \cdot 7 + 7 + 0.6 \cdot 3 = 27.2$, which is automatically the optimal cost for $S_{02}$ with the corresponding production scheme of $(x_1, x_2) = (C, \epsilon)$. We can calculate this for every production sequence and the corresponding costs and production schemes can be found in Table 3.1. The cumulative costs can be found in Table 3.2, which can be used to determine the optimal production sequence and its cost by using dynamic programming recursion. One can see that $f_4 = d_{34} + f_3 = 37.2$ and $f_3 = d_{03} + f_0 = 35.4$. This results in the optimal production plan consisting of two production sequences, namely $S_{03}$ and $S_{34}$, with corresponding production schemes of $(x_1, x_2, x_3) = (\epsilon, C, C)$ and $x_4 = \epsilon$. This means that the production in $t = 1$ will be equal to $\epsilon = (4 + 6 + 9) \bmod 7 = 5$, such that $x_t = [5, 7, 7, 2]$ with corresponding costs of 37.2.

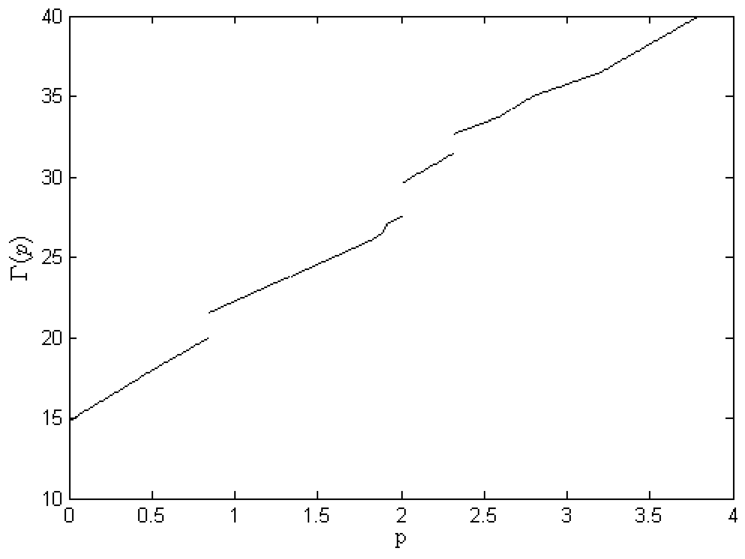

Figure 3.2.1: Cost function $\Gamma(p)$ plotted against $p$

### 3.2.2 Cost function

The optimal solution of a capacitated ELSP is not as simple as that of the uncapacitated ELSP, where its graph is a piecewise linear concave function. Instead, the graphical representation of the optimal solution of a capacitated ELSP consists of multiple piecewise linear functions. This is due to the fact that if demand is increased, the production capacity of a certain production period will be reached and a new production period has to be started. This will cause the costs to rise significantly again, due to newly induced setup and production costs. It may even occur that a subplan needs to be altered in order to satisfy the new demand, as the total demand will not be able to be met anymore by the available production in the periods of the subplan. This will cause a jump in the graph, where a new function starts above the point where the previous function ended, leaving a vertical gap. This phenomenon can be seen in Figure 3.2.1. At certain prices $p$ where new intervals begin, for example at $p = 0.84$ and $p = 2$, one can observe pretty significant jumps in the costs. This happens when a different subplan becomes better than what was used before in the optimal production plan. This problem did not occur in the uncapacitated case as production would just increase as demand rose.

Let us call the cost function $\Gamma$. The value of this function at a certain point $p$ is equal to the minimum of the costs of all procurement plans that are valid at point $p$. In the uncapacitated case the function $\Gamma$ was piecewise linear and everywhere concave. In the capacitated case however, $\Gamma$ is still piecewise linear, but will not necessarily be concave everywhere. Instead, it is piecewise linear and concave on a certain interval of the prices, in which we only change the structure of the optimal production plan to obtain the least costs. At the endpoint of such an interval the production plan needs to be changed, as the current production plan will become infeasible if $p$ is increased beyond that point. This is caused by a fractional production period reaching capacity $C$, such that production has to start in a different period which requires a change in the structure of the production plan. Since the production plan will not be valid anymore outside this interval, the cost of this production plan will be set to infinite. This makes the function $\Gamma$ the lower envelope of a set of functions that is linear in $p$ on some interval and infinite elsewhere.

We can calculate said endpoints by the following formula presented by Geunes et al. [2009]:

$$p_{t\tau}^{m} = \frac{mC - \sum_{j=t}^{\tau-1} \alpha_j}{\sum_{j=t}^{\tau-1} \beta_j} \quad \text{for } m = 1, \ldots \tau - t + 1; \tau = t+1, \ldots, T; t = 1, \ldots, T$$

This means that there are $O(T^3)$ endpoints that we need to take into consideration. On any interval created by taking two consecutive values of $p_{t\tau}^{m}$ the optimal cost function $\Gamma$ is piecewise linear and concave. The endpoints can thus be sorted in increasing order to create a contiguous sequence of $O(T^3)$ intervals for all $p \geq 0$ on which $\Gamma$ is piecewise linear and concave. As for the last endpoint, we also want an interval in which this is the lower bound. Therefore we need an upper bound on $p$ for this interval. For this purpose, we will calculate the $p$ for which demand can still be satisfied in every period given the capacity. Let us call this $p_{UB}$, then because we have $D_t(p) = \alpha_t + \beta_t d(p)$ with $d(p) = 1$, we get

$$p_{UB} = \min_t \left( \frac{C - \alpha_t}{\beta_t} \right).$$

We choose the minimum of these prices, as higher prices would let the demand

| $T$ | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|
| $n(T)$ | 2 | 7 | 16 | 30 | 50 | 77 | 112 | 156 | 210 | 275 |
| $T$ | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | |
| $n(T)$ | 352 | 442 | 546 | 665 | 800 | 952 | 1122 | 1311 | 1520 | |

Table 3.3: Number of endpoints as $T$ increases

exceed production capacity in the period where the lowest price was found, as this was the price with which demand would be equal to $C$.

As the amount of endpoints is dependent on $T$, we can calculate the amount of endpoints to see how it develops as $T$ increases. The results can be found in Table 3.3. We can see that the amount of endpoints increases rapidly as $T$ becomes larger. The amount is still reasonable at $T = 5$, but the rapidly increasing amount afterwards will put a large strain on computation time. Upon closer inspection, it seems the relation between the amount of endpoints $n(T)$ and $T$ can be expressed in the following cubic function:

$$n(T) = \frac{1}{6}T^3 + \frac{1}{2}T^2 - \frac{2}{3}T,$$

which shows that the amount of endpoints is of the order $O(T^3)$.

### 3.2.3 Modified exact algorithm

Now that we have defined the intervals in which $\Gamma$ is piecewise linear and concave, we can apply the same exact algorithm from Van den Heuvel and Wagelmans [2006] as in the uncapacitated ELSP on each interval, but with minor modifications. First of all, it should be clear that instead of the Wagner-Whitin algorithm we will be using the Florian-Klein method for the capacitated case. The Kunreuther-Schrage algorithm and Eisner-Severance method remain the same, however, the way the Eisner-Severance method is executed will be different. In the uncapacitated case there was a separate function which took two production plans $S_L$ and $S_U$ as inputs, calculated the intercepts and slopes of the corresponding lines and produced a suitable $\bar{p}$. However, this approach is not needed in the capacitated case, since the intercept and slope of a line can be calculated immediately during the Florian-Klein method.

First, we will have production sequence $S_{uv}$ which is a production plan for the periods $u+1,\ldots,v$. Let us define $F$ to be the set of periods in which production is equal to the capacity $C$ and let period $e$ be the period in which the production is at less than capacity. We can calculate the intercept of the line corresponding to the production plan as follows:

$$r = \sum_{t \in F} \left(K_t + \left(c_t^* - c_e^*\right) C\right) + K_e + c_e^* \sum_{t=u}^{v} \alpha_t$$

and the corresponding slope can be calculated as

$$s = c_e^* \sum_{t=u}^{v} \beta_t.$$

The respective intercepts and slopes of the different production sequences which make up the best production plan for a given $p$ can be simply added up to calculate the intercept and slope for the production plan, which can then in turn be used in the exact algorithm to calculate the intersection point $\bar{p}$ and to see whether the production plans are equivalent.

As mentioned before, we can apply the exact algorithm from Van den Heuvel and Wagelmans [2006] on each interval. However, the exact algorithm can only be applied on a closed interval. In our case, we have endpoints which define the consecutive intervals. These intervals however are left-open, which means it has no minimum as the lower bound is not included in the interval. This is due to the fact that an endpoint is a transition from one optimal production plan to another. This endpoint is included in an interval as the upper bound, but is not included as the lower bound. If it were, then it would result in the same optimal production plan as in the interval before, where it was used as an upper bound, which is not desired. Geunes et al. [2009] seem to have overlooked this issue and hence do not explain how to tackle this problem.

We will use an unorthodox way to still be able to obtain a production plan on the lower bounds of intervals by increasing the price $p$ with a small $\varepsilon = 2^{-24} = 5.96 \cdot 10^{-8}$. This artificial increase in the demand will bring the function past the endpoint, into the new interval. However, if this artificial increase proves to be too much of an increase, such that it surpasses the upper bound of the same interval, the algorithm will not be run on that interval. We choose this option instead of even further lowering the $\varepsilon$, since the machine epsilon of MATLAB is equal to $\varepsilon_m = 2 \cdot 2^{-53} = 2.22 \cdot 10^{-16}$. If we would lower our $\varepsilon$ even further, the algorithm will be more prone to numerical imprecisions as it would use numbers with a difference close to $\varepsilon_m$.

According to Geunes et al. [2009] the overall solution approach for the capacitated problem will run in $O(T^9)$ time. This is because we now have the function $\Gamma$ which contains $O(T^3)$ intervals and within each interval there are $O(T^3)$ breakpoints. This means we need to solve a total of $O(T^6)$ equal-capacity lot-sizing problems with the Florian-Klein method, each of which can be solved in $O(T^3)$ time, which results in a total time of $O(T^9)$. However, because of the flaw in the proof of the exact algorithm of Van den Heuvel and Wagelmans [2006], we can not be sure whether there are actually $O(T^3)$ breakpoints in each interval. Therefore we also can not be sure whether the solution approach of Geunes et al. [2009] has a running time of $O(T^9)$. The only thing we can say for certain at this point is that the running time will be at least $O(T^6)$ due to the number of intervals and the time to solve an equal-capacity lot-sizing problem with the Florian-Klein method.

## 3.3 Results

We are curious about how the running time of this algorithm will be in practice compared to the proposed $O(T^9)$ from Geunes et al. [2009]. Therefore we need to simulate test cases again as we did for the uncapacitated part. Before we do that, we are curious about how the algorithm performs on known test cases. Therefore we will be using various cases depicted by Deng and Yano [2006]. After that, we will take a look at how the computation time develops as we let $T$ vary. Finally, we will discuss how the test cases are chosen and how the algorithm fares on these test cases regarding computation time. As with the results obtained in the uncapacitated part, all of the results are obtained by using the 64-bit version of MATLAB on a desktop computer with a quad-core processor at 3.2 GHz and 4GB RAM.

### 3.3.1 Known test cases

To be sure our algorithm performs well and will be giving optimal solutions, we want to test our algorithm on certain known cases. The paper of Deng and Yano [2006] offers some test cases we can use for this purpose. The model horizon for these test cases is $T = 6$. We will be using the three demand scenarios specified in the paper, namely homogeneous demand ($a_t = 10$, for all $t$), a scenario with increasing demand curves ($a = (7.5, 8.5, 9.5, 10.5, 11.5, 12.5)$) and a scenario with seasonal demand ($a = (10, 14, 6, 10, 14, 6)$). The other parameters are as follows: $b_t = 1$, $K_t = 10$, $c_t = 1$ and $h_t = 0.1$ for all $t$. As always we can incorporate the unit holding costs in the unit production costs as specified in section 2.3.2. Do note that Deng and Yano [2006] use a different demand function than the one specified in Kunreuther and Schrage [1973], which is used throughout this paper. They use an inverse demand function of the form $p_t = a_t - b_t D_t$, which can be rewritten as $D_t = \frac{a_t - p_t}{b_t}$. Since $b_t = 1$ for all $t$ and we will be using constant prices, this simplifies to $D_t = a_t - p$. Our demand function is of the form $D_t(p) = \alpha_t + \beta_t d(p)$, which means that we obtain $\alpha_t = a_t$, $\beta_t = -1$ and $d(p) = p$. Since Deng and Yano [2006] are interested in optimal prices rather than optimal production plans, we need to calculate the optimal prices in our algorithm as well. Since we have the linear cost function readily available for each production plan within an interval, we can determine the profit as follows:

$$\begin{aligned}
\Pi(D(p)) &= \textstyle\sum_t (D_t p) - \Gamma(p) \\
&= \textstyle\sum_t ((\alpha_t + \beta_t p) p) - (r + sp) \\
&= p \textstyle\sum_t \alpha_t + p^2 \sum_t \beta_t - r - sp \\
&= p^2 \textstyle\sum_t \beta_t + p(-s + \sum_t \alpha_t) - r
\end{aligned}$$

Then we can find the optimal price at which profit is maximized for each optimal production plan on $\Gamma$ as:

$$\begin{aligned}
\Pi'(D(p)) &= 0 \\
2p^* \textstyle\sum_t \beta_t + (-s + \sum_t \alpha_t) &= 0 \\
2p^* \textstyle\sum_t \beta_t &= s - \textstyle\sum_t \alpha_t \\
p^* &= \tfrac{s - \sum_t \alpha_t}{2 \sum_t \beta_t}
\end{aligned}$$

However, this resulting $p^*$ needs to lie within the interval in which the production plan is defined. Otherwise the optimal price for the production plan is one of the endpoints of the interval, more specifically, the endpoint with the higher profit. This can be deduced intuitively, since $\Pi(D(p))$ is a quadratic function with $p^*$ corresponding to its peak, if $p^*$ does not lie within $[p_L, p_U]$, then $\Pi(D(p))$ is either strictly increasing or decreasing on $[p_L, p_U]$, which makes the endpoint with the higher profit the optimal $p$ for that interval. Finally, we can determine the optimal price for the test case by looking for the $p^*$ which corresponds to the highest profit. This will be done for various capacities.

First we will look at the results for the case with homogenous demand. We will let capacity $C$ vary from 1 to 20, taking only the integer values. The optimal prices found by Deng and Yano [2006] for both the constant price and the average unit price can be found in Figure 3.3.1, while our results can be found in the left graph of Figure 3.3.2. We can immediately see something peculiar. We would expect our results to coincide with the results of Deng and Yano [2006] in the case of constant prices. However, we can see that they roughly coincide with the results of average unit prices instead.

To doublecheck that our prices are correct, we will calculate the optimal prices through enumeration, where we gradually increase the price with 0.01 in the interval $[5, 9.5]$. The results are displayed in Figure 3.3.2 on the right. We can see that the prices obtained in both ways are nearly identical, except for $C = 4$ and $C = 11$. These differences can be accounted to the imprecision of the iterative method, as we used steps of 0.01 for the price. This minor difference can influence the choice of the best price in the case of $C = 11$ or even the choice of subplans for the optimal production plan in the case of $C = 4$. Overall, our two results obtained through different means are similar.

We can see that there are also some disparities between our results and that of Deng and Yano [2006]. For example, one can see that for $C = 4$ we get $p^* = 6.8$, while Deng and Yano [2006] seem to get $p^* = 6$, and for $C = 10$ we get $p^* = 6.67$ and they get $p^* = 5.52$. However, Deng and Yano [2006] make no notion on the exact capacity values they use and the markers on the graphs can be misleading. On closer inspection, it turns out that $p^* = 6.1$ is obtained at $C = 3.9$ and we get $p^* = 5.52$ at $C = 9.74$, after which it immediately rises to $p^* = 6.75$ at $C = 9.75$.

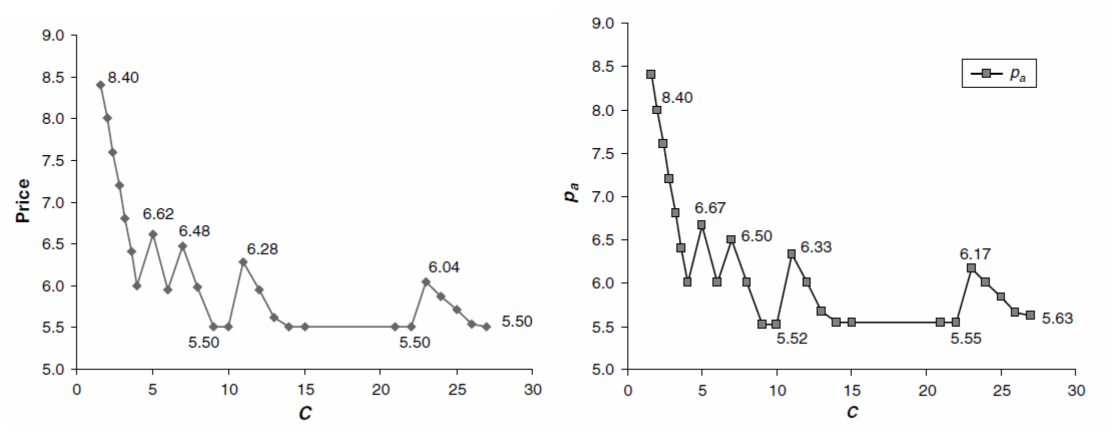

Figure 3.3.1: Optimal Constant (left) and Average Unit Prices (right) as Functions of Capacity: Homogenous Demands (Figures taken from Deng and Yano [2006], p.749 and 751)

We can say that in general our results correspond to the results of Deng and Yano [2006] for the homogenous demand case, albeit the case of average unit price and not constant price.

If we look at the case with increasing demand, displayed in Figure 3.3.3, we can see that the optimal constant and average unit prices obtained by Deng and Yano [2006] lie fairly close to each other. The case with constant prices however, obtains lower optimal prices than when average unit prices are used for every $C$. Even though it is quite hard to see to which capacities and prices the markers on the graphs belong, we can see that our results coincide more with the results from Deng and Yano [2006] with constant prices, which is expected. This can be seen between $C = 9$ and $C = 10$, where our optimal price slightly increases. The same happens for Deng and Yano [2006] in the case of constant prices, whereas it decreases when we look at the average unit price. Furthermore, we obtain a price of around 5.55 for $C = [15, \ldots, 20]$, which is similar to the case of constant prices. One might notice that our graph does not match that of Deng and Yano [2006] completely. This is due to different points being plotted, as we can see that Deng and Yano [2006] do not have strict guidelines on the capacities they plot. This can be seen clearly around $C = 5$ where there is a cluster of markers, whereas we plot the prices for integer capacities. If we take capacities with steps of 0.1 and plot the optimal prices, the resulting graph fluctuates even more as little changes

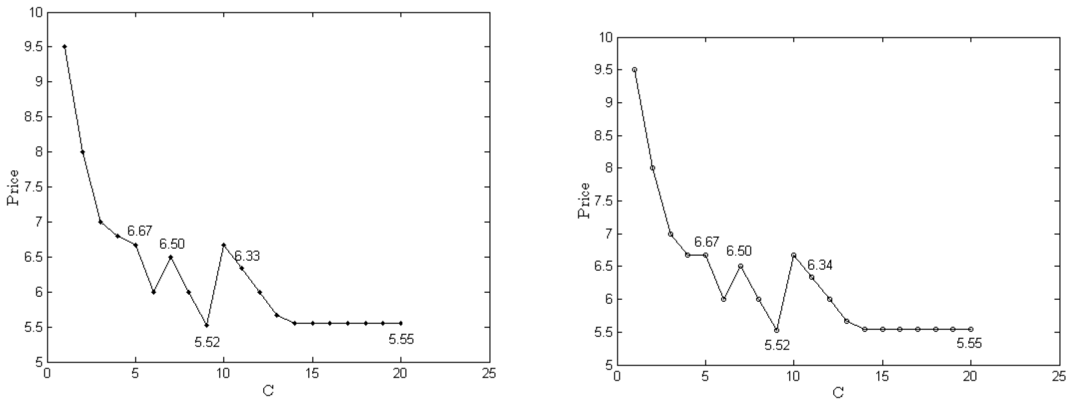

Figure 3.3.2: Optimal Prices as Functions of Capacity: Homogenous Demands; Left: Obtained through the algorithm; Right: Obtained through enumeration

in capacity change the optimal production plans and therefore the optimal price, especially at lower $C$. This graph differs even more from the graph of Deng and Yano [2006].

Finally, we take a look at the case of seasonal demand. The results can be found in Figure 3.3.4. Again our results are similar to that of Deng and Yano [2006] in the case of constant prices. We can see that Deng and Yano [2006] decided not to plot points for the case of constant prices for $C \leq 6$. Unfortunately, they do not mention the reason behind this decision. In our results we can see that the optimal price is rather high for $C \leq 6$, after which it drops below 6.00 and stays there as $C$ increases. In $C = [15, 20]$ we can see minor fluctuations in the price, with the most notable one being the price getting back to 6.00 at $C = 17$. Unfortunately, we can not see this in the results of Deng and Yano [2006], as it seems they obtain the same optimal price at these values of $C$.

All in all we can say that our results roughly coincide with that of Deng and Yano [2006]. It is hard to compare the results as Deng and Yano [2006] do not report the exact capacity values plotted in the graph, nor give numerical values for the prices for the cases with increasing and seasonal demands. Peculiar is that our results for the case with homogenous demand coincide with their results on average price, whereas for the other demand cases our results coincide with their results on constant prices. One would think that our results would be similar to

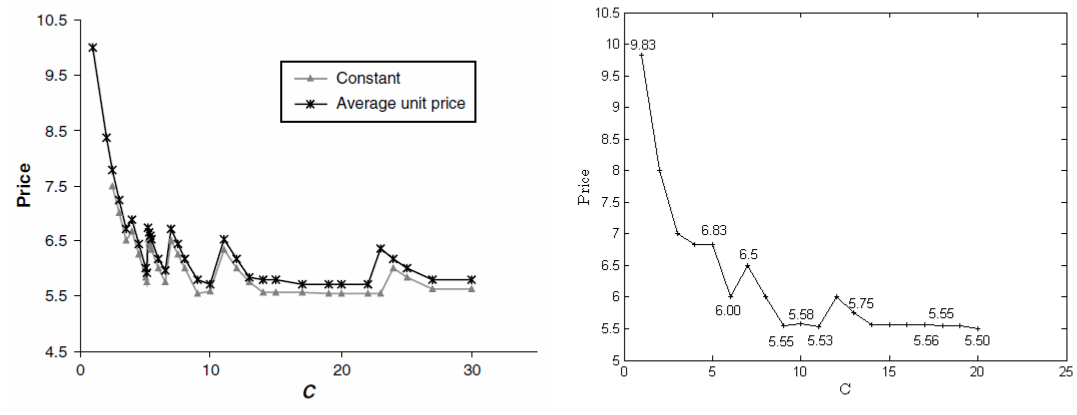

Figure 3.3.3: Left: Optimal Constant and Average Unit Prices as Functions of Capacity: Increasing Demands (Figure taken from Deng and Yano [2006], p. 751); Right: Optimal Constant Price for Increasing Demands

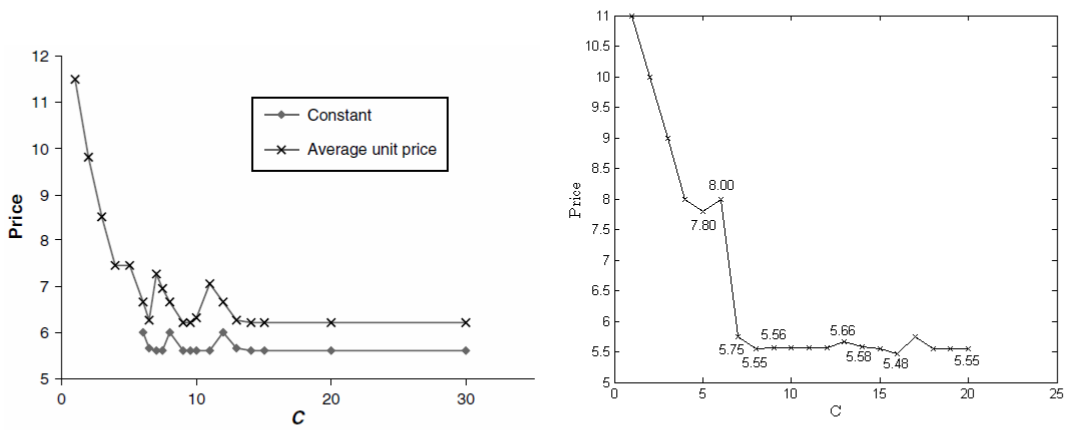

Figure 3.3.4: Left: Optimal Constant and Average Unit Prices as Functions of Capacity: Seasonal Demands (Figure taken from Deng and Yano [2006], p. 751); Right: Optimal Constant Price for Seasonal Demands

theirs for constant prices, as we use time-indifferent prices in our model. However, it is hard to believe that it is coincidence that our results with constant prices are the same as their results with average prices. It might be possible that the figures in Deng and Yano [2006] have been switched. Although this would also be peculiar as one can see that for the cases with increasing and seasonal demand the constant prices are always lower than the average prices. If the figures for the case of homogenous demand would be switched, it would mean that the optimal prices would be higher when constant prices are used than when average prices are used.

### 3.3.2 Computational results for a single case

As with the uncapacitated case, we are interested in the development of the computation time as we let $T$ vary. Therefore we will look at a specific case which we will run for various $T$. We will also look at the amount of endpoints and the amount of breakpoints in each interval individually, instead of the total amount. We can compare our results with the results from Geunes et al. [2009].

We will use the same parameters again as in section 2.3.1, which are as follows:

$$\begin{aligned}
&D_t(p) = D + p && \text{for } t = 1, \dots, T,\\
&K_t = (T - t + 1)K && \text{for } t = 1, \dots, T,\\
&c_t = T^{T-t+1} && \text{for } t = 1, \dots, T,
\end{aligned}$$

with $D = \alpha_t = 0$, $\beta_t = 1$ and $K = 100$. In addition to these parameters, we set the capacity at $C = 2$, 5 and 10. We run the modified exact algorithm over all intervals using these parameters and $T = 4, \dots, 20$. The results are shown in Table 3.4.

We can see that the computation time steeply rises as $T$ gets larger. Whereas the computation time in the uncapacitated case was merely a second for $T = 20$, it is well over two minutes in the capacitated case. This can be accounted to the number of intervals, which is equal to the number of endpoints $n(T)$, as the algorithm needs to run for each interval consisting of two consecutive endpoints $n(T)$ and where the last endpoint gets paired up with an upper bound on $p$. Notice that the amount of endpoints found is smaller than what we would have expected. This is caused by our choice of parameters, which generates many similar endpoints, such that the amount of unique endpoints is smaller. We cannot detect a drop in the computation time anymore as $T$ gets larger as we did before in the uncapacitated case. This can be accounted to the low amount of total breakpoints ($bp$) found, which means that in the capacitated case the problem gets more intensive as $T$ increases and there needs to be gradually more calculations. It does not suffer from a drop in the total amount of breakpoints, which would cause less calculations and therefore lower computation times, as what was the case in the uncapacitated problem. As for the amount of breakpoints in each interval, there was merely one case where there were two breakpoints in one interval, namely at $T = 4$ with $C = 5$. For all the other cases there were either none or one breakpoint in every interval for each $C$. The larger computation time in the case of $T = 4$ for $C = 5$ compared to the time for $C = 2$ is due to this occurrence of two breakpoints in one interval for $C = 5$. This is because the algorithm needed to solve an additional equal-capacity lot-sizing problem with the Florian-Klein method, which takes more time.

If we look at the computation times across the different capacities, we can not find a trend unfortunately. However, we do see that the computation time is dependent on the amount of breakpoints found, as one would expect. As the amount of breakpoints found is equal for different $C$, the computation time only differs slightly. Whereas if the amount of breakpoints is larger, one can see a significant increase in computation time. One can also see that the amount of breakpoints is either equal or lower for larger $C$ at the same $T$. This is the case until $T = 15$. This is as expected since as $C$ gets larger, one can produce more in one production period, thus one would exceed the production capacity less often, resulting in less breakpoints. This in turn means that there were less calculations needed, thus resulting in lower computation times.

From $T = 16$ onwards this is not the case anymore, as the amount of breakpoints seems to get larger for higher $C$. This can be explained as follows. Due to the aforementioned formula used to calculate endpoints by Geunes et al. [2009], the endpoints will have small values for a low $C$. Therefore the range on which $p$ can operate is limited. As $C$ gets larger, $p$ can get larger as well and therefore demand will be able to rise. A large $C$ for small $T$ means that the problem is as good as uncapacitated. However, as $T$ becomes larger, there will be more total demand. At some point the total demand will surpass capacity $C$ again, forcing a new production period. This means that the problem will become more like a capacitated problem again, which will have more possible production plans and therefore generate breakpoints again. What happens with smaller $C$ at large $T$ is that its endpoints remain small, thus demand can not increase as much. Therefore the production plans are limited to low demands, which does not create much diversity and thus not many breakpoints. This is why we see higher amounts of breakpoints for $C = 5$ and $C = 10$ than for $C = 2$ for these larger $T$.

Figure 3.3.5 shows a plot of the computation times over $T = 4, \ldots, 20$ for $C = 2$, 5 and 10. We can clearly see that the computation time does not suffer for higher $T$ as it did in the uncapacitated case. We can deduce in which order of time the algorithm has run for this particular test case by looking at what happens when $T$ is doubled. This will be done only for $C = 2$ as we can see that the computation times lie close to each other and the graphs are similar. We can see that by doubling $T$, the computation times get multiplied by $(19, 23.8, 23.5, 30.2, 27.5, 31.3, 32.3)$ respectively. This seems to indicate that the algorithm has run in $O(T^5)$ time, since by doubling $T$, the computation time has been multiplied by about $2^5 = 32$.

We would actually have expected this algorithm to run in at least $O(T^7)$. This is due to the fact that we should have $O(T^3)$ intervals for which the Florian-Klein method has to be run at least once, which runs in $O(T^4)$ time as we have implemented it in the traditional way. However, if we take a look at our results, one can see that the number of endpoints found is greatly lower than what one would expect. If $T$ would be doubled, we can see that the amount of endpoints is about a factor four larger. This means that we have about $O(T^2)$ endpoints. This is again due to the way we have chosen our parameters, which causes many endpoints to be similar, resulting in a low amount of unique endpoints. For each endpoint the Florian-Klein method has to be run, which totals an order $O(T^6)$ time. Do note that these results come from fairly small $T$. We would see this effect better for larger $T$. Furthermore, these results have been obtained using a mere example, not a worst case scenario, as it was in the uncapacitated case. This explains why we are not even close to the proposed running time of $O(T^9)$.

| $T$ | $n(T)$ | Found endpoints | $C = 2$ | | $C = 5$ | | $C = 10$ | |
|---|---|---|---|---|---|---|---|---|
| | | | time (s) | *bp* | time (s) | *bp* | time (s) | *bp* |
| 4 | 16 | 7 | 0.08 | 2 | 0.10 | 2 | 0.07 | 1 |
| 5 | 30 | 10 | 0.18 | 3 | 0.16 | 1 | 0.15 | 0 |
| 6 | 50 | 15 | 0.45 | 3 | 0.41 | 1 | 0.39 | 0 |
| 7 | 77 | 18 | 0.75 | 2 | 0.73 | 1 | 0.73 | 1 |
| 8 | 112 | 25 | 1.52 | 1 | 1.52 | 1 | 1.53 | 1 |
| 9 | 156 | 30 | 2.54 | 1 | 2.56 | 1 | 2.55 | 1 |
| 10 | 210 | 37 | 4.28 | 1 | 4.23 | 0 | 4.24 | 0 |
| 11 | 275 | 42 | 6.37 | 1 | 6.37 | 1 | 6.48 | 0 |
| 12 | 352 | 53 | 10.59 | 1 | 10.63 | 1 | 10.85 | 0 |
| 13 | 442 | 58 | 14.53 | 1 | 14.57 | 1 | 14.43 | 0 |
| 14 | 546 | 71 | 22.64 | 1 | 22.63 | 1 | 22.50 | 0 |
| 15 | 665 | 78 | 32.14 | 1 | 31.82 | 1 | 30.35 | 0 |
| 16 | 800 | 87 | 41.74 | 0 | 42.61 | 2 | 42.74 | 1 |
| 17 | 952 | 96 | 56.24 | 1 | 57.01 | 2 | 56.93 | 2 |
| 18 | 1122 | 113 | 79.48 | 1 | 80.04 | 1 | 79.99 | 1 |
| 19 | 1311 | 120 | 98.73 | 0 | 99.06 | 0 | 98.94 | 0 |
| 20 | 1520 | 139 | 138.16 | 1 | 141.01 | 4 | 141.26 | 4 |

Table 3.4: Computation time and number of total breakpoints of the modified exact algorithm for various $T$ and $C = 2$, 5 and 10

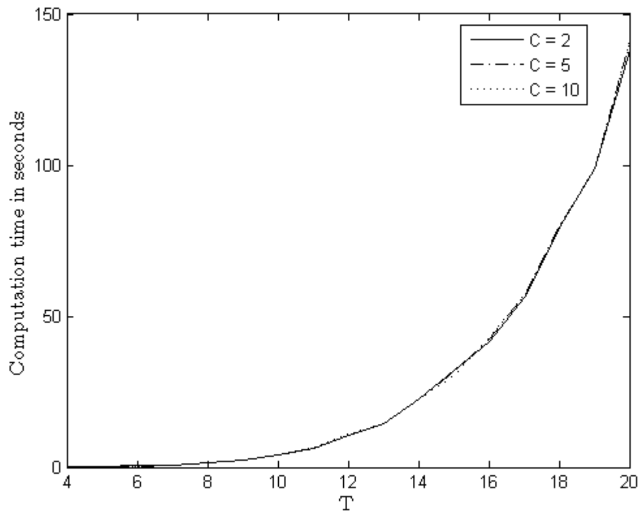

Figure 3.3.5: Computation time for $T = 4, \ldots, 20$ and $C = 2$, 5 and 10

### 3.3.3 Computational results for multiple cases

Since one case does not say much about the computation time of the algorithm in practice, we will generate a large amount of test cases and see how well the algorithm performs on them. Because the algorithm takes more time to solve the capacitated problems than the uncapacitated ones, we will have a smaller amount of test cases this time compared to section 2.3.1. We will let the parameters $\alpha$, $\beta$, $K$, $c$ be drawn from three ranges, namely from $[0, 10^0]$ to $[0, 10^2]$. From each range, we will draw one random number sequence for each parameter. All the number sequences for these parameters will be sorted descending, as we have seen in the uncapacitated case that this gave us the most amount of breakpoints. The random drawn parameters from each range will be combined with each other, such that we can see what kind of effect the scale of the parameters have on the computation time and amount of breakpoints. This means we will have $2^4 = 16$ test cases per range combination. Since we will have $3^4 = 81$ range combinations, this results in a total of $16 \cdot 81 = 1296$ test cases per $T$. Note that capacity $C$ is a single value instead of a time-variant sequence. Therefore we will generate $C$ individually for each case. Since we want all cases to be solvable, we will draw the capacity $C$ for each case as a random number from $\left[\max_t(\frac{\sum_{t=1}^{T} D_t}{t}), 2 \cdot \max_t(D_t)\right]$, where $D_t = \alpha_t + \beta_t$. We choose this minimum on $C$, since it needs to be able to satisfy the demand in all periods, otherwise we will have an unsolvable case. For the maximum, we want to allow a case to be treated as nearly uncapacitated as well. However we do not want too many uncapacitated cases, so twice the maximum demand should be a reasonable maximum for $C$.

First we will look at $T = 4$. In Table 3.5 we can see that there was one out of 1296 cases with a total of five breakpoints. Furthermore, the maximum amount of breakpoints within an interval was two, which occurred in twelve cases. We are now curious whether the various ranges of the parameters had any influence on the results. Therefore we will group up all the cases with the same parameter ranges, making 81 groups of 16 cases each. For each group we will determine the maximum amount of breakpoints, the amount of breakpoints within an interval and for both these statistics the number of cases that contained these numbers. Furthermore, we are interested in the minimum amount of breakpoints in a group, so that we can see whether there is a group which has breakpoints in all sixteen cases. These statistics can be found in Table 3.6, where we have chosen to display the groups for which there were 4 or 5 breakpoints. We can see that for all the cases with 4 or more breakpoints the setup cost $K$ was in the range of $[0, 10^2]$. Furthermore, the unit production cost is low when there were 4 breakpoints found, in the range of $[0, 10^0]$. However, in the case of 5 breakpoints the unit production cost was in the range of $[0, 10^1]$. This seems to be inconsistent. The same can be said about $\beta$. The only parameter which seems to have no consistent effect at all is $\alpha$, as it differs for each case.

To determine whether these parameters have a consistent effect on the amount of breakpoints, we will also take a look at $T = 5$ and $T = 6$. The results are shown for 6 breakpoints for $T = 5$ and for 8 to 11 breakpoints for $T = 6$ in Tables 3.7 and 3.8 respectively. We can see that for $T = 5$ the setup cost $K$ is still large. However, $\beta$ seems to be fairly random, which means that it has no consistent effect on the number of breakpoints. The unit production cost $c$ on the other hand seems to stay fairly low in most cases. If we take a look at $T = 6$, we can see that $K$ is not in the range of $[0, 10^2]$ anymore for two of the four cases with a large number of breakpoints. However, $c$ is still low in all these cases.

If we take a look at the minimum amount of breakpoints within a group, we can see that for some groups all the cases contained at least one or even two breakpoints. Unfortunately, there seems to be no combination of parameter ranges for which this occurs consistently. As for the maximum amount of breakpoints within an interval, there are no cases where this number is extraordinarily high. The breakpoints are well spread over the intervals, as one would expect from randomly generated data.

In the end, it seems that the setup cost $K$ and unit production cost $c$ have an effect on the amount of breakpoints, while $\alpha$ and $\beta$ do not. This could be expected as these costs shape the cost function $\Gamma$. If the contrast between $K$ and $c$ is big, as it is in these cases, where $K$ lies in the range $[0, 10^2]$ and $c$ in $[0, 1]$, the total cost will change dramatically if one would start a new production period. This in turn could cause a breakpoint in $\Gamma$.

Now that we have seen that a relatively large $K$ and small $c$ have a significant effect on the amount of breakpoints found, we can generate more test cases using

| $T$ | max $bp$ | #cases | max $bp$ interval | #cases | time (sec) |
|---|---|---|---|---|---|
| 4 | 5 | 1 | 2 | 12 | 174 |
| 5 | 6 | 7 | 3 | 1 | 570 |
| 6 | 11 | 1 | 3 | 1 | 1514 |
| 7 | 17 | 1 | 2 | 65 | 3541 |

Table 3.5: Results for test cases over $T$

| factor | | | | max $bp$ | #cases | min $bp$ | max $bp$ interval | #cases |
|---|---|---|---|---|---|---|---|---|
| $\alpha$ | $\beta$ | $K$ | $c$ | | | | | |
| 0 | 0 | 2 | 0 | 4 | 1 | 1 | 1 | 16 |
| 0 | 2 | 2 | 0 | 4 | 2 | 0 | 1 | 15 |
| 1 | 2 | 2 | 0 | 4 | 3 | 2 | 1 | 16 |
| 1 | 2 | 2 | 1 | 5 | 1 | 1 | 2 | 2 |
| 2 | 2 | 2 | 0 | 4 | 1 | 0 | 2 | 2 |

Table 3.6: Results for $T = 4$: Range combinations with 4 and 5 breakpoints

| factor | | | | max $bp$ | #cases | min $bp$ | max $bp$ interval | #cases |
|---|---|---|---|---|---|---|---|---|
| $\alpha$ | $\beta$ | $K$ | $c$ | | | | | |
| 0 | 0 | 2 | 2 | 6 | 1 | 1 | 2 | 2 |
| 0 | 1 | 2 | 1 | 6 | 1 | 0 | 1 | 15 |
| 1 | 1 | 2 | 1 | 6 | 2 | 2 | 2 | 2 |
| 2 | 0 | 2 | 0 | 6 | 1 | 1 | 2 | 3 |
| 2 | 1 | 2 | 0 | 6 | 1 | 2 | 1 | 16 |
| 2 | 2 | 2 | 0 | 6 | 1 | 2 | 2 | 1 |

Table 3.7: Results for $T = 5$: Range combinations with 6 breakpoints

| factor | | | | max $bp$ | #cases | min $bp$ | max $bp$ interval | #cases |
|---|---|---|---|---|---|---|---|---|
| $\alpha$ | $\beta$ | $K$ | $c$ | | | | | |
| 0 | 0 | 1 | 0 | 11 | 1 | 2 | 2 | 1 |
| 0 | 1 | 2 | 1 | 10 | 1 | 2 | 3 | 1 |
| 1 | 1 | 2 | 0 | 8 | 1 | 2 | 2 | 1 |
| 2 | 0 | 0 | 0 | 9 | 1 | 0 | 1 | 13 |

Table 3.8: Results for $T = 6$: Range combinations with 8 to 11 breakpoints

| $T$ | Found endpoints | max $bp$ | #cases | max $bp$ interval | #cases | time (s) | avg time (s) |
|---|---|---|---|---|---|---|---|
| 4 | 16 | 3 | 6 | 2 | 1 | 36.3 | 0.14 |
| 5 | 30 | 6 | 1 | 2 | 2 | 117.7 | 0.46 |
| 6 | 50 | 7 | 1 | 2 | 4 | 308.5 | 1.21 |
| 7 | 77 | 8 | 4 | 3 | 1 | 736.1 | 2.88 |
| 8 | 112 | 11 | 1 | 2 | 20 | 1528.8 | 5.97 |
| 9 | 156 | 10 | 2 | 2 | 16 | 2842.3 | 11.10 |
| 10 | 210 | 13 | 1 | 2 | 19 | 5368.7 | 20.97 |

Table 3.9: Results for fixed ranges for $T = 4, \ldots, 10$

these parameter characteristics. Therefore we will not vary the ranges of the parameters anymore, but only draw random number sequences from predetermined ranges. We will draw $K$ from the interval $[0, 10^2]$, $c$ from $[0, 1]$ and $\alpha$ and $\beta$ from $[0, 10^2]$. We choose the interval $[0, 10^2]$ for $\alpha$ and $\beta$ since we have seen for $T = 4$ and $T = 5$ that it produced cases which contained fairly large amounts of breakpoints. For each parameter four random number sequences will be drawn from their respective ranges and these sequences will be combined again to construct $4^4 = 256$ test cases for each $T$. We have chosen for four random number sequences, since 256 test cases seems like a fair amount to let our algorithm run on. We have seen that for 1296 test cases the running time at $T = 7$ was already nearing an hour. This would mean that for larger $T$ the computation time would be many hours. Therefore we have chosen a smaller amount of test cases, such that we will have reasonable computation times for larger $T$. Again, $C$ will be drawn randomly for each case in the aforementioned way. The results can be found in Table 3.9.

We can see that we unfortunately do not obtain as many breakpoints as we had hoped. The amount of breakpoints for $T = 5$ is similar to the maximum amount we found when we looked at the various range combinations. However, for the other $T$ the amount of breakpoints is rather low compared to our previous results. This was to be expected, as we could not find a range combination which consistently produced a vast amount of breakpoints. The only parameters we were sure of having an impact on the amount of breakpoints were $K$ and $c$. The choice of $\alpha$ and $\beta$ might seem to be unfortunate, but these parameters seemed to have little impact, which would probably mean that we would have obtained similar results if we had chosen different ranges for these parameters.

If we take a look at the computation time, we can see that it rises vastly as we increase $T$. Even though we have chosen a smaller set of test cases for each $T$, the computation time already comes close to an hour for $T = 9$. We can compare the average time of a run with what we obtained in 3.3.2. We can see that the running time of these cases with randomly generated parameters is larger than with the constructed case. To be exact, the running times are larger by factors $(1.4, 2.3, 2.5, 3.5, 3.6, 4.0, 4.5)$ compared to the constructed case with $C = 2$. This is due to the fact that $\alpha$ and $\beta$ are time-dependent now, which causes more endpoints. In fact, the amount of endpoints found coincides with the amount that we would expect from Geunes et al. [2009].

Since it seems that even 256 test cases for each $T$ might seem too much regarding the computation time, we will look at the practical running time for one test case for each $T$. We will take a look at $T = 11, \ldots, 20$, as we already have results for $T$ up to 10. The results can be found in Table 3.10. We can see that we continue to find the expected amount of endpoints. The amount of total breakpoints and breakpoints per interval found seems lackluster, but one should note that this is merely for one test case. We can see that the computation time goes from a mere 30 seconds at $T = 11$ to a hefty 1216 seconds at $T = 20$, which is about 20 minutes. If we calculate the factor the computation times have increased after doubling $T$, we obtain $(47.8, 51.3, 49.1, 50.8, 58.0)$ for going from $T = (6, 7, \ldots, 10)$ to $T = (12, 14, \ldots, 20)$. These factors are within $O(T^6)$, however, one should note the sudden increase in factor at $T = 20$. This might indicate that the computation time for even larger $T$ have an even larger factor.

## 3.4 Conclusion

We know that solving an equal-capacity lot-sizing problem using the method from Florian and Klein [1971] takes $O(T^4)$ time. This result can be improved to $O(T^3)$ as Van Hoesel and Wagelmans [1996] show. From our results we can see that for practical problems, where the parameters of the problem are not artificially constructed, the number of endpoints is of the order $O(T^3)$. This amount can go down as parameters coincide in such a way that endpoints overlap each other. However, this requires very specific values for the parameters and these are not

| $T$ | Found endpoints | $bp$ | max $bp$ interval | time (sec) |
|---|---|---|---|---|
| 11 | 275 | 3 | 1 | 32.70 |
| 12 | 352 | 4 | 1 | 57.80 |
| 13 | 442 | 3 | 1 | 86.15 |
| 14 | 546 | 8 | 1 | 147.71 |
| 15 | 665 | 9 | 1 | 207.90 |
| 16 | 800 | 5 | 1 | 293.38 |
| 17 | 952 | 5 | 1 | 436.63 |
| 18 | 1122 | 4 | 1 | 563.66 |
| 19 | 1311 | 18 | 1 | 881.14 |
| 20 | 1520 | 22 | 1 | 1216.27 |

Table 3.10: Results for one test case for $T = 11, \ldots, 20$

prominent in practice. Within an interval the number of breakpoints is equal to $O(T^3)$ according to Geunes et al. [2009], using a theorem by Van den Heuvel and Wagelmans [2006]. However, due to a flaw in the proof of this theorem, it is not sure whether this theorem holds. As we have seen in the previous chapter, we could not disprove this theorem unfortunately. Therefore we used simulations to determine the practical running time of the algorithm suggested by Geunes et al. [2009] to see whether their proposal still holds even with the flaw in the proof. However, we were unable to even come close to their result of $O(T^9)$, let alone find a higher order of time.

It proved to be difficult to find many breakpoints within an interval, as there is no predefined worst case scenario which holds for the capacitated case. Therefore our test cases did not come close to the proposed $O(T^3)$ amount of breakpoints within an interval. This resulted in lackluster computation times, where it seemed that only the amount of endpoints and the Florian-Klein method had impact on the computation time. We did try to find a certain direction towards a worst case scenario, in the form of ranges on the parameters. However, this proved to be not highly successful, as the amount of breakpoints within an interval remained low for all scenarios. We did see that a relatively large setup cost $K$ and relatively small unit production cost $c$ resulted in cases with a decent total amount of breakpoints. However, the amount of breakpoints found seemed to be inconsistent throughout $T$. This could also be ascribed to the fact that the number of test cases became smaller as $T$ became larger. We were unable to generate many test cases for large $T$ as this would consume a lot of time.

In the end, we were unable to generate test cases such that they would approach the computation time proposed by Geunes et al. [2009]. Therefore we could not compare our results with theirs and determine whether the flaw in the proof of the theorem by Van den Heuvel and Wagelmans [2006] has an effect on the algorithm of Geunes et al. [2009].

# Chapter 4

# Conclusion

The economic lot-sizing problem is a well-known problem within the field of logistics. However, it stays intriguing as we develop new methods and try to improve the already existing methods. To this extent, Van den Heuvel and Wagelmans [2006] proposed an exact algorithm to determine the optimal price and lot-sizing decisions. Geunes et al. [2009] used this algorithm in turn to propose an algorithm of their own, now for the capacitated case. Unfortunately, it turned out there was a flaw in the proof of Van den Heuvel and Wagelmans [2006], which made it uncertain whether their theorem still holds. This impacts the result of Geunes et al. [2009] as well. Therefore, we tried to disprove the theorem, since if this would be successful, we would have a clear answer.

Unfortunately, we were unable to do so. We were unable to find a counterexample to disprove the theorem. We were able to isolate a scenario which would contain many breakpoints. However, after simulating many test cases, we were unable to find a test case which contained more breakpoints than what was expected from the theorem.

The capacitated lot-sizing problem is of a different caliber than the uncapacitated problem. It is much harder to tackle and even though it is possible to solve it, it is often not desired as it takes quite some time. We have seen that the practical running time increases dramatically as the model horizon becomes larger. Due to this, we could not conduct our simulations in the same way as for the uncapacitated case. We started off with a large set of test cases, prone to find test cases with many breakpoints, as we did before. However, as computation time became larger, we had to decrease our amount of test cases. Therefore we focussed on a certain pattern for the parameters, for which we thought we would obtain cases with many breakpoints. After limiting our test cases, it still seemed to be not enough, after which we even reduced the number of test cases to one per model horizon $T$.

Whereas the choice of parameters to concentrate on was successful for the uncapacitated case, it proved to be not that worthwhile for the capacitated case. It is hard to define a worst case scenario for the capacitated lot-sizing problem, as one has to take into account many parameters and the structure of the problem and costs. We did see that the computation time increased immensely as $T$ became larger, up to the point where one test case took up to twenty minutes. However, we were unable to determine for a large number of $T$ what the order of the running time is. We were unable to show whether the flaw in the proof of the theorem by Van den Heuvel and Wagelmans [2006] has an effect on the results of the algorithm for the capacitated lot-sizing problem.

## Further research

As there are many questions left open after this research, there are many points for further research. First of all, we have stumbled upon a scenario which seemed that Geunes et al. [2009] have overlooked while constructing their algorithm. They did not take into account that the exact algorithm of Van den Heuvel and Wagelmans [2006] is only defined on a closed interval. The construction of their endpoints into intervals however suggest a left-open interval. We have remedied this by increasing the endpoint with a small $\varepsilon$ such that the lower bound of the interval could be used in the algorithm. However, this is just a quick solution and by all means not a proper way of doing it. One could search for a solution to do this in a proper way.

Furthermore, the scenarios that we used for the capacitated case seemed to have no consistent effect on the total amount of breakpoints found. Neither did it affect the amount of breakpoints within an interval, even though we expect $O(T^3)$ breakpoints within each interval. It would be interesting to look for a worst case scenario in the capacitated lot-sizing problem, where one would consistently get $O(T^3)$ breakpoints within an interval, such that the algorithm from Geunes et al. [2009] can be properly tested.

Finally, it is still not clear whether the theorem of Van den Heuvel and Wagelmans [2006] is correct, as there is no new proof available. It would be interesting if someone would find a counterexample to disprove the theorem, or maybe even come up with proof that the theorem still holds.

# Bibliography

A. Aggarwal and J.K. Park. Improved algorithms for economic lot-size problems. *Operations Research*, 14:549–571, 1993.

S. Deng and C.A. Yano. Joint production and pricing decisions with setup costs and capacity constraints. *Management Science*, 52:741–756, 2006.

M.J. Eisner and D.G. Severance. Mathematical techniques for efficient record segmentation in large shared databases. *Journal of the Association for Computing Machinery*, 23:619–635, 1976.

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.

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

J. Geunes, Y. Merzifonluoglu, and H.E. Romeijn. Capacitated procurement planning with price-sensitive demand and general concave revenue function. *European Journal of Operational Research*, 194:390–405, 2009.

H. Kunreuther and L. Schrage. Joint pricing and inventory decisions for constant priced items. *Management Science*, 19:732–738, 1973.

W. Van den Heuvel and A.P.M. Wagelmans. A polynomial time algorithm for a deterministic joint pricing and inventory model. *European Journal of Operational Research*, 170:463–480, 2006.

C.P.M. Van Hoesel and A.P.M. Wagelmans. An O($T^3$) algorithm for the economic lot-sizing problem with constant capacities. *Management Science*, 42:142–150, 1996.

A.P.M. Wagelmans, C.P.M. 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.

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