Chapter 6

Simulation Environment

In order to prepare the numerical studies of Chapter 7, we start by introducing the developed simulation environment. We divide this chapter into two parts. First, we discuss the technical settings and issues regarding the implementation, and second, we introduce the parameter set required to run simulations.

6.1 Technical Settings and Implementation Issues

6.1.1 Test Environment

All models and algorithms are part of a single simulation environment implemented in C++ and compiled under Microsoft Visual Studio 2005. The LP models (AP and GOP) are solved using the open source GNU Linear Programming Kit (GLPK) version 4.28. The GLPK library is accessed over a C++ interface in order to run a large number of simulations consecutively. All simulations are executed on a standard PC with a 2.4GHz Intel Core 2 CPU and 512MB memory.

Uniform random numbers are generated using the “Mersenne Twister” algorithm provided by the open source GNU Scientific Library (GSL) version 1.9. In extensive simulations, it was realized that the standard C++ random number generator does not yield satisfying results, so we turned to the Mersenne Twister algorithm described by Matsumoto and Nishimura (1998). To generate random numbers, we use the inverse transformation method: when $X$ is a random variable with a cumulative distribution function $F$, and $U$ is a Uniform distributed random number between 0 and 1, then $X$ can be computed with $X = F^{-1}(U)$.

6.1.2 Implementation Issues

As the implementation usually bears some complications, we illustrate the RM approach of Section 5.1 in pseudo code. In particular, we show how to compute the value function of Equation 5.5 which is required to compute the critical levels $L_t(c, i, \bar{y}_t)$. The value function $V_t(\bar{x})$ has to be computed for all realizations of $\bar{x}$ in all periods $t$. As $V_t(\bar{x})$ can only be computed when $V_{t+1}(\bar{x})$ is known, the algorithm iterates from the last period $T$ to the first period.
This step is illustrated in Algorithm 1. The algorithm recursion as shown in Line 2 contains the iterations through $\bar{x}$ and is explained below.

**Algorithm 1:** Stochastic Dynamic Program

- **input**: The $ATP(t)$ quantities
- **input**: The number of periods $T$
- **input**: A zero-initialized capacity vector $\bar{x}$
- **input**: A zero-initialized vector $v_t(\bar{x})$ for the value function
- **output**: The vector $v_t(\bar{x})$ filled with the expected profit-to-go for each realization of $\bar{x}$

```plaintext
1 for $t \leftarrow T$ to 1 do
2    recursion(0, $t$, $\bar{x}$, $v_t(\bar{x})$);
3 end
```

Algorithm 2 shows a recursive iteration through all realizations of $\bar{x}$. A recursive algorithm is necessary because the number of periods $T$—and accordingly, possible supply arrivals—might change from one run of the algorithm to another one, and for each period a loop through the supply quantities arriving in this and all other periods is required.

For each specific realization of $\bar{x}$, the expected value $E_{d,c}$ has to be computed. This is done by iterating through all possible demand realizations $d$ and $c$. As the theoretical limit of $d$ goes to infinity, we have to find an upper limit for $d$, denoted by $D$. If the limit $D$ is chosen too high, the computation time is above practical limits. If the limit is chosen too low, the results are not satisfying. In all simulations in this work we chose a limit $D$ such that 99.9% of all possible values $d$ are in the range between 0 and $D$. $D$ can be computed easily with the inverse transformation method: $D = F^{-1}(0.999)$.

In the following, we go through the steps of the algorithm and explain the data variables. In Line 1, the algorithm iterates through the supply quantities arriving in the period $atp\_arrival\_time$. In Line 2, the current supply quantity of ATP arriving in $atp\_arrival\_time$ is stored in the vector $\bar{x}$. In the next step, the stopping criteria is checked: when the last period is reached, there are no further supplies to be considered. Prior to the last period, a new recursion is started (Line 4) with the next arriving supply. At the time the stopping criteria is met and the "else" sector is reached (Line 5), the vector $\bar{x}$ contains the supply information of all arriving supplies. In the next steps, the expected value is calculated by iterating through all possible demand quantities (Line 6) and customer classes (Line 7).

In Line 8, the policy of Theorem 5.1 is evaluated in the Function acceptance Rule. An order is accepted stepwise as long as the profit is larger than the opportunity costs. Note that in this step, any other policy can also be used.
6.2 SIMULATION ISSUES

Algorithm 2: recursion(atp_arrival_time, period, \( \bar{x}, v_t(\bar{x}) \))

**input**: The `atp_arrival_time` of the current considered ATP quantity
**input**: The current considered period \( t \)
**input**: The vector of the current considered capacities \( \bar{x} \)
**input**: A zero-initialized vector \( v_t(\bar{x}) \) for the value function
**output**: The vector \( v_t(\bar{x}) \) filled with the expected profit-to-go for each realization of \( \bar{x} \) in the current period \( t \)

1 for \( cap \leftarrow 0 \) to \( ATP_{atp\_arrival\_time} \) do
2 \( x_{atp\_arrival\_time} = cap; \)
3 if \( atp\_arrival\_time < T \) then
4 recursion(\( atp\_arrival\_time + 1, t, \bar{x}, v_t(\bar{x}) \));
5 else
6 for \( d \leftarrow 0 \) to \( D \) do
7 for \( c \leftarrow 1 \) to \( C \) do
8 profit \( \leftarrow \) acceptanceRule(\( \bar{x}, d, c \));
9 \( \text{expprofit} \leftarrow \text{expprofit} + \text{Prob}(d, c) \times \text{profit}; \)
10 end
11 end
12 \( v_t(\bar{x}) \leftarrow \text{expprofit}; \)
13 end
14 end

e.g. FCFS or the SOPA approach. Therefore, we do not describe this step in more detail. The function `acceptanceRule` returns the profit associated with the fulfillment decisions of the current order represented by the demand quantity \( d \) and class \( c \). For each demand realization (possible values of \( d \) and \( c \)), the probability \( \text{Prob}(d, c) \) is multiplied to the profit resulting from the acceptance decision (Line 9). After the iterations through all demand realizations, the profit-to-go for the current considered realization of vector \( \bar{x} \) is stored in \( v_t(\bar{x}) \) (Line 12).

6.2 Simulation Issues

6.2.1 Data Generation

Kimms and Müller-Bungart (2007) present a review on papers dealing with demand data generation with a focus on different assumptions on demand. The authors state that "... assuming demand data that follows a non-homogeneous
Poisson process is more or less standard nowadays”. However, the Poisson assumption requires that demand and variance are equal, which might not be true in many applications. For instance, Lawless (1987) states that count data often display extra variation beyond the scope of Poisson distributions. In such cases, the negative Binomial distribution (NB) exhibits certain advantages, as NBs have a long tradition in the marketing and operations literature. It is often mentioned that NBs fit best to observed customer demand (e.g. Ehrenberg, 1959, Agrawal and Smith, 1998).

We distinguish two settings of demand data streams. First, we present a general setting representing demand fulfillment decisions of one year executed on a rolling horizon. Second, we consider a setting which is consistent with the assumptions of the RM approach of Section 5.1. For instance, in the RM setting, only one order is allowed to arrive per period. Table 6.1 displays the options considered in the two settings and shows the used symbols and underlying assumptions.

<table>
<thead>
<tr>
<th>Description</th>
<th>Distribution</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Demand data stream</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Orders per period</td>
<td>Poisson($\mu$)</td>
<td>$n_t$</td>
</tr>
<tr>
<td>Order quantity</td>
<td>Deterministic</td>
<td>$d_i$</td>
</tr>
<tr>
<td>Revenues per order</td>
<td>$U^c(a,b)$</td>
<td></td>
</tr>
<tr>
<td>Requested due date</td>
<td>Deterministic</td>
<td>$r_i$</td>
</tr>
<tr>
<td>No-arrival probability</td>
<td>Deterministic</td>
<td>$p_o$</td>
</tr>
<tr>
<td><strong>Supply data stream</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inter-arrival time</td>
<td>Deterministic</td>
<td>$\phi$</td>
</tr>
<tr>
<td>Supply quantity</td>
<td>Deterministic</td>
<td>$s_t$</td>
</tr>
</tbody>
</table>

In the general setting, the number of orders $n_t$ arriving in period $t$ is modeled as a Poisson process. The demand quantity per order $d_i$ is set to a fixed value. The revenues $r_i$ of order $i$ are uniformly distributed in the range $a$ to $b$. As we focus on make-to-stock environments, we assume in the simulation runs that an immediate order confirmation is required (i.e., $\gamma_i = 0$).

Supply data is generated in both settings without any stochastic influences. This is due to our assumption that on the short-term, supply can be considered as given. The supply quantities $s$ arrive in specific intervals, defined by the inter-arrival time $\phi$. We model the supply arrival process $s_t$ with $s_t = s$ if $t \mod \phi = 0$ and $s_t = 0$ otherwise.

The RM setting is different to the general setting in order to account for the different demand assumptions. First of all, the number of orders $n_t$ is
6.2 SIMULATION ISSUES

deterministic and set to 1 (cf. Assumption 5.1). However, in order to be able to capture the effects of no-arrival probabilities, we introduce \( p_0 \) denoting the probability that no customer order arrives in a period. Then, the arrival process can be modeled as such that one customer class arrives per period with an arrival probability of \( \left( \frac{1-p_0}{\# \text{classes}} \right) \) for each class (all classes have the same arrival probability). As the customer demand is expected to be discrete and positive, we chose an \( \text{NB} \) to model the demand quantity. Additionally, it allows us to analyze the effects of large variations in demand. The demand quantity is modeled to be strictly positive with \( d_i \sim 1 + \text{NB}(\mu - 1, \sigma^2) \). Note that in contrast to the general setting, the demand quantity is always positive (i.e. \( > 0 \)).

In contrast to the general setting, the revenues of all orders within the same class are equal. The assignment of orders to classes is stochastic and uniformly distributed. Due to Assumption 5.3, orders require an immediate fulfillment, but are willing to accept later deliveries with a price discount. Therefore, \( \gamma_i = 0 \) holds in all simulations.

To study the impact of supply shortage on the performance, we define shortage as a ratio of the total supply throughout the simulation horizon and the total expected demand, more formally stated as

\[
s_{R} = 1 - \frac{\sum_{t=1}^{T} S_t}{(1 - p_0) \times E[n_t] \times E[d_t] \times T}.
\]

6.2.2 Simulation Options

The simulation runs of Chapter 7 can be distinguished according to the data stream options of the previous section and simulation options as shown in Table 6.2. We describe the simulation options in the following.

<table>
<thead>
<tr>
<th>Description</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of customer classes</td>
<td>( K )</td>
</tr>
<tr>
<td>Simulation horizon</td>
<td>( T )</td>
</tr>
<tr>
<td>Planning window</td>
<td>( W )</td>
</tr>
<tr>
<td>Replanning frequency</td>
<td>( F )</td>
</tr>
<tr>
<td>Backlogging costs</td>
<td>( b )</td>
</tr>
<tr>
<td>Holding costs</td>
<td>( h )</td>
</tr>
<tr>
<td>Forecast error</td>
<td>( e )</td>
</tr>
</tbody>
</table>

The number of customer classes \( K \) is considered as a given input and can be changed from one simulation run to the next. By this, we can assess the trade-
off between finer customer differentiation and increasing forecast accuracy. In this regard, the assignment of customers to classes is done based on a simple procedure which results in a well-balanced amount of customers in each class. Refer to the work of Meyr (2008) for a more detailed analysis of different clustering methods. The procedure works as follows: first, determine the average number of customers per class by dividing the number of customers by the number of classes. Second, sort all customers according to their profits. Then, assign the most valuable customer to the first class. If the first class contains the average number of customers, move to the next class. Repeat this process until all customers are assigned.

The next three options are used to define a rolling horizon. The simulation horizon \( T \) covers all periods of a simulation run, e.g. one year. The planning window \( W \) covers the periods in which the short-term demand management decisions are simulated. In practical settings, the planning window covers a few weeks to month. The replanning frequency \( F \) determines the amount of time that lies between two consecutive planning windows. Usually, the replanning frequency is shorter than the planning window in order to have overlapping time periods. Figure 6.1 illustrates the three options.

![Simulation Horizon, Window and Frequency](image)

Figure 6.1: Simulation Horizon, Window and Frequency

Backlogging and holding costs \((b\) and \(h\), respectively\) are not dependent on a specific order and, therefore, are part of the simulation options. We did not implement class- or order-dependent backlogging and holding costs because neither the RM approach (cf. Sect. 5.1) nor the network version of GOP (cf. Sect. 4.3.1) support it.

The last option determines how demand forecasts are generated during a simulation run. We distinguish between forecasts based on the mean demand, and forecasts that are generated according to a predefined forecast error \( e \). In the first case, we assume that the mean demand is known (e.g. by exponential smoothing) and is used as a forecast for the demand in SOPA and the RM approach. In case of RM, we additionally assume that the demand variance is...
known as well. If the mean demand is chosen for generating demand forecasts, we denote this by $M$. The resulting forecast errors in this case are hence equal to the standard deviation of the demand stream. For example, if the demand is Poisson distributed with $\lambda = 10$, then the standard deviation is 3.33 which is also the mean deviation from the forecasts.

In the second case, we generate forecasts according to the predefined error and the true realization of the demand, denoted by $d^*_{kt}$. Let $d^\text{max}_{kt}$ denote the demand forecast for class $k$ in period $t$ and $\epsilon_{kt}$ the forecast error, then the forecast error is distributed as

$$\epsilon_{kt} \sim \text{Norm}(0, \frac{\sigma_e}{\sqrt{K}}),$$

and the demand forecast can be calculated with $d^\text{max}_{kt} = \max\{0, d^*_{kt} + \epsilon_{kt}\}$. Note that negative demand is changed to 0. $\sigma_e$ is chosen as a percentage of the mean demand during the planning window. Let $m$ denote the mean demand in the planning window over all classes, then $\sigma_e$ can be calculated with $\sigma_e = \epsilon \times m$.

6.2 shows that the variability of the forecast error within a specific class decreases with an increasing number of classes. However, the variability of the forecast error over all classes $\sum_{k=1}^{K} \epsilon_{kt}$ increases with an increasing number of classes. This behavior resembles common forecasting methods used in practical settings.

For illustration of this rather counterintuitive behavior, consider a mean demand of 1,000 units per period with a standard deviation of 500. If the demand is equally segmented into 1,000 individual classes, a mean of one unit per period per class can be expected. According to Formula 6.2, the standard deviation of the demand per class changes to $\frac{500}{\sqrt{1000}} = 15.81$. This small example shows that the standard deviation of the demand of one class has to be smaller than the standard deviation of the total demand.

### 6.2.3 Output and Key Performance Indicators

Since the discussed approaches of Chapter 5 focus on profit maximization, we choose the expected profit as the key performance indicator (KPI) in order to compare the different approaches. Expected profits of the RM approach described in Section 5.1 can be directly calculated by solving the value function 5.5. An implementation of the value function is discussed in Algorithm 2. If the assumptions of the RM approach hold, expected profits of SOPA and FCFS can be calculated by means of the value function, as well, just replacing the
RM acceptance rule (Line 8 in Algorithm 2) with the SOPA or FCFS rules. Equation 6.3 shows the value function adapted to FCFS or SOPA.

\[
V_t(\bar{x}) = E_{d,c} \left[ \sum_{0 \leq u_t \leq x_t, \sum_{i=1}^{T} u_i \leq d} \left( u_t P_t(i, c) - h x_t \delta_{it} \right) + V_{t+1}(\bar{x} - \bar{u}) \right] \tag{6.3}
\]

However, in the simulation runs of Chapter 7 we do not calculate expected profits by means of the value function, but rather run a “reasonable” large number of simulations with different random variates (drawn from the discussed distributions). Subsequently, we calculate the average profit over all simulation runs as an approximation of the real expected profit. The reasons for this are: (1) in some of the scenarios, the demand assumptions of the RM approach do not hold. (2) The GOP solution cannot be calculated by means of the value function, because the Bellman principle of optimality does not hold in this case. The principle states that the optimal decision \(\bar{u}\) in period \(t\) only depends on the current state \(\bar{x}\) and the expected profits in period \(t + 1\). However, in case of GOP, an optimal decision in period \(t\) depends on the actual demand realizations of all periods. (3) In large scenarios, computing the value function is practically not possible due to the curse of dimensionality. In these cases, the approximate expected profit is sufficient to identify trends in the numerical results.

A problem remains how to choose a reasonable number of simulation runs, denoted by \(\Omega\) in the following. If \(\Omega\) is chosen too large, the simulation time increases too much. If chosen too low, the approximation of the expected profits is insufficient and the results might be due to random influences. In scenarios with a large variance in the demand, \(\Omega\) must be chosen high in order to prevent random outcomes. In the simulation runs of Chapter 7, we have manually chosen \(\Omega\) in the way that a larger \(\Omega\) would not contribute very much to the trends seen in the different figures.

In order to capture how much of the performance can be attributed to the different approaches directly and not to differences in the simulation settings, we predominantly show profit deviations to GOP in the numerical results, instead of showing absolute profits. These profit deviations, or relative profits respectively, are calculated according to the formula \(\sum_{i=1}^{\Omega} (\text{GOP}_i^* - S_i^*) / \text{GOP}_i^*\), when \(S_i^*\) represents the optimal profit of a certain approach \(S\) in simulation run \(i\) and \(\text{GOP}_i^*\) the optimal profit of GOP in simulation run \(i\).

We complement the analysis of expected profits with an analysis of service rates. We distinguish between four different service rates: (1) order quantities fulfilled \textit{in time}, order quantities fulfilled \textit{too early}, order quantities fulfilled
too late (backlogging), and rejected order quantities (lost sales). We choose quantity-oriented service measures (Tempelmeier, 2006) adapted to our setting with simultaneous lost sales and backlogging. Furthermore, since order quantities can be split, we did not focus on orders as a whole but calculated the service measures according to single units. For instance, the amount of backlogging covers all units (not orders) with a delivery date in the planning window that are backlogged.