Optimizing stocking strategy for bakeries

Willem de Muinck Keizer[1], Naqi Huang[1], Haojin Li[2], Michael Muskulus[3], Marco Saltini[2], Luke Visser[4]

Abstract

It is a challenge for bakeries to stock in a way that maximizes sales while minimizing leftover stock. The basic model often used is known as the newsvendor model. This model does not consider interaction between sales, such as substitution. We propose several ways to address these interactions. First, we propose an augmented newsvendor model which takes substitution into account. Second, we model a typical day of sales with a large-scale stochastic simulation. Finally, we develop a probabilistic evolution method to calculate expected profit numerically. These methods may be used to test stocking strategies, and we provide additional methods for optimization. With this, we aim to provide bakeries with ways to improve their stocking strategy.

Keywords: stocking strategies, multiproduct newsvendor, stochastic simulations, probability density evolution, optimization.

1 Introduction

We investigated the problem of finding an optimal stocking strategy in the case of a bakery. In particular, we looked at the effect of substitution sales: a customer turns up to purchase a product but finds it being sold out and either leaves or purchases a different product. Furthermore, we also investigated the influence of multi-product sales and set up a simulation model that can take stock-dependent preferences into account. In Table 1, we summarize our notational conventions.

For a simple model with no substitutions, the optimal stocking level q_i for product i is the solution to the newsvendor problem

$$q_i = F_i^{-1}((p_i - c_i)/p_i), \tag{1}$$

where F_i is the cumulative distribution function of demand, p_i the retail price of the product, and c_i the production cost (Arrow et al., 1951; Qin et al., 2011). This solution depends on the distribution of the demand for each product and on the profit margins. If we do allow

[1]Delft Institute of Applied Mathematics, Delft University of Technology, Delft, The Netherlands
[2]Mathematical & Statistical Methods, Plant Science Group, Wageningen University, The Netherlands
[3]Department of Civil and Environmental Engineering, NTNU, Trondheim, Norway
[4]Department of Mathematics and Computer Science, Eindhoven University of Technology, The Netherlands

for substitutions, the bakery will not only sell to the direct demand but also to people who switched because their initial choice was out of stock. As a result, the total demand will increase by adding substitutions.

Table 1: Notations, based on the conventions used in Smith and Agrawal (2000).

Indices	
n	Types of cakes ($i \in \{1, \ldots, n\}$)
m	Type of potential purchase (which cake i, or which multiset b_i of cakes in case of multi-item purchases)
Parameters	
$p = (p_1, \ldots, p_n)$	Prices of each cake i written as a vector
$c = (c_1, \ldots, c_n)$	Costs of each cake i written as a vector
$q = (q_1, \ldots, q_n)$	Stock levels of each cake i written as a vector
q_0	Initial stock levels at the beginning of a period
q_*	Optimal stock levels at the beginning of a period
$Q = (Q)_q$	Probability of having certain stock levels, indexed by q
ΔQ	Change in stock probabilities due to customer actions
$r(q)$	Revenue obtained for given initial stock levels q
D	Total demand
D_i	Demand for the i-th cake
K	Total number of customers arriving per cycle
$\psi(k)$	$P\{K = k\}$
$p_e(k)$	Probability that a k-th customer exists on a given day
μ	Customer arrival rate / Mean number of customers
f_i	$P\{$costumer initially prefers the i-th cake/multi-item purchase$\}$
$K_i(f_i)$	Total number of costumers initially preferring the i-th cake
$\psi_i(k_i \mid f_i)$	$P\{K_i(f_i) = k_i\}$
α_{ij}	$P\{$customer switches to cake $j \mid$ cake i was not available$\}$
a_{ij}	Same as α_{ij}, but generalized to multi-item purchases
$A = (a_{ij})_{i,j}$	Customer switching probabilities written as a m-by-m matrix
α	Symmetric switching probability for first two customer preferences, $\alpha = a_{12} = a_{21}$
b_i	Multiset of items for the i-th purchase/customer preference
$B = (b_{ij})_{i,j}$	Number of desired items (index j) for each purchase (index i), written as a m-by-n matrix
β	Desired service level for first stock
β_*	Actual service levels for each stock

Throughout this model, we make some assumptions about general customer behavior and determine some parameters that can be used to tailor the model to individual bakeries.

One of the more important sets of parameters is f_i, which is the probability that a person initially wishes to purchase product i. These parameters may be inferred by considering market shares or by overstocking and considering sales data. If the products are independent,

then f_i may be calculated from historical sales data directly by dividing sales of product i by the total number of customers per day. One may argue that the f_i can be calculated from past sales data in this way too. Though the f_i thus calculated are already influenced by stocking policy and substitutions, most bakeries will naturally stock in such a way that all their products go out of stock near the end of the day. This results in a natural balancing that ensures that the estimated preferences lie reasonably close to the real preference f_i.

Given the probabilities of initial preference, we now take substitution into account in our stocking policy. In Smith and Agrawal (2000), it is shown that the optimal stocking strategy with substitution can be found by first finding the net demand for a product as the sum of the direct demand and the demand due to substitutions. The optimal stocking strategy is then the solution of the newsvendor problem for independent products 1, but now with F_i the cumulative distribution of the net demand. To calculate this new net demand distribution, we need to make two assumptions on the initial distributions of the number of customers.

We first assume that the total number of customers K follows a negative binomial distribution

$$\psi(k) = \binom{N+k-1}{N-1} p^N (1-p)^{k_i}, \tag{2}$$

for some values of p and N inferred from data. For the demand of the individual products, this results in a negative binomial distribution

$$\psi_i(k_i|f_i) = \binom{N+k_i-1}{N-1} y_i^N (1-y_i)^{k_i}, \tag{3}$$

where $y_i = p/[p - f_i(1-p)]$ Smith and Agrawal (2000). The negative binomial distribution has two parameters N and p compared to the one parameter λ of the Poisson distribution. In the limit of $p \to 1$, $N \to \infty$ such that the expected value $N(1-p)/p$ is constant, the negative binomial distribution converges to a Poisson distribution.

For our simulations, we only have information on the expected sales value. Since the negative binomial is then over-parameterized, we choose the parameters such that we approximate a Poisson distribution. Thus, we set $p = 0.99$, and we let N be such that the mean of the distribution is equal to the mean of the sales data.

The second assumption is that we assume that the customers that switch from a product $j \neq i$ to product i do not change the number of customers that switch away from product i if i itself is out of stock. This strictly underestimates the expected number of customers that switch away from product i. The assumption gives a good approximation if the customers switch only once if their initial preference is not there and if either the number of people that switch is small compared to the total initial demand or if all products go out of stock at roughly the same time during the day.

Under this assumption and with a given stocking policy q_i, the probability that a customer initially preferring product i arrives and finds product i out of stock is, on average, over the number of cycles given by

$$A_i = \frac{1}{E(K_i(f_i))} \sum_{q_i=k_i}^{\infty} (k_i - q_i)\psi_i(k_i|f_i),$$

where $(k_i - q_i)\psi_i(k_i|f_i)$ is the number of customers that find a product out of stock multiplied by the probability of that amount of customers arriving on a given day. Multiplying this with the switching probability and summing over all possible substitutions gives us the probability of a person wishing to purchase product i as

$$R_i = f_i + \sum_{i \neq j} f_j \alpha_{ij} A_j,$$

where the term $f_j \alpha_{ij} A_j$ captures the probability that a person initially prefers item j, finds it out of stock, and switches to item i. Let us mention that the paper Smith and Agrawal (2000) uses the upper bound

$$R_i = f_i + \sum_{i \neq j} f_j \alpha_{ij} (1 - r_i),$$

where r_i is the service rate. The service rate r_i is the probability of running out of stock on a given day.

With either of these new probabilities, the new distribution of the demand for product i is given by the negative binomial distribution of equation 3 with

$$y_i = p/[p - R_i(1 - p)].$$

Knowing the new net distribution of the demand, we can find the optimal stocking strategy from the solution of the newsvendor problem.

In Figure 1, one can see that with a relatively high substitution rate of 50% between only two products, the change in the demand is small. For the given stocking rates, the inclusion of substitution increases the optimal stocking strategy with only one product: from 18 to 19 for product 1 and from 7 to 8 for product 2.

In short, we still use newsvendor, but perturb the distributions to take substitution into account. We suggest that BuyFresh considers the above alternative stocking policies and tests these with either the probabilistic simulation or the computational approach worked out in the following sections.

2 Large-scale simulations: a day in the bakery

In this section, we take a practical approach to address this problem. Considering the complexity of customer behavior, our primary focus is on examining the scenario when customers substitute one product for another. We use the data from the bakery shop, which is presented in Table 2. The data consists of 10 distinct cakes categorized into 5 types, each of which comes in 2 varying sizes. Our objective is to find the optimal stocking strategy for each cake by simulating customer behaviors. More specifically, we aim to find the initial stocks of cakes that maximize the profit, i.e

$$\max_{q_1, q_2 \ldots q_{10}} \quad \{p | p = \sum_{i=1}^{10} (p_i s_i - c_i q_i)\}, \tag{4}$$

where s_i is the number of cake i sold. See description of other notions in Table 1.

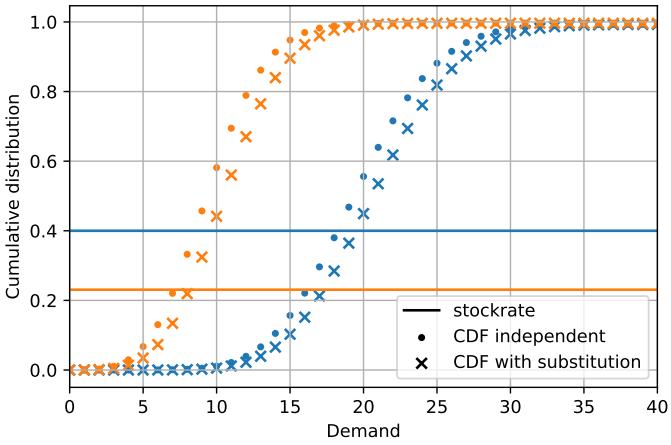

Figure 1: Comparison between demand without substitution and with substitution, with a given price-dependent optimal stocking rate. The expected total demand without substitution is $\mathbb{E}[D] = 30$, the initial preferences are $f_1 = 2/3$, $f_2 = 1/3$, the substitution rates are $\alpha_{12} = \alpha_{21} = 0.5$, and stocking rates are given by $r_1 = (p_1 - c_1)/p_1 = 0.4$ and $r_2 = (p_2 - c_2)/p_2 = 3/13$.

2.1 Model assumptions and parameters setup

We set up our assumptions and the other parameters needed in our simulation as follows:

1. The number of customers K is fixed at 200 per day.

2. The number of cakes each customer buys is drawn from the probability mass function shown in Table 3. Note that more realistic probabilities can be obtained from sale data.

3. Choice of the initial cake(s): we chose the initial cake by drawing randomly from a distribution taken from the frequency data of cake sales. In our experiment, the probability that each customer buys cake i is the ratio of the demand for cake i and the total demand for all cakes (the data is shown in Table 2). For customers buying two or three cakes, it can happen that their initial choice is to buy two or three cakes of the same type.

4. Substitution rule: if the preferred cake is out of stock, a customer buys the larger (or smaller) size cake of the same type with probability α, or a different cake of the same size as the initially selected one with probability $(1 - \alpha))/5$ each. If the replacement is also not there, the customer leaves without buying anything. Note that this approach includes a probability of $(1 - \alpha)/5$ to leave the shop immediately (i.e., the replacement is exactly the original cake). In our model, when customers want to buy more than one cake, they leave the shop if they cannot find one replacement for each of the two or three. Generalization to other types of behavior is possible by making minor changes to

Table 2: The price, cost, and demand of all available types of vlaai.

Product ID	Product	Price	Cost	Demand
1	Cherry Vlaai L	15,25	8,97	25
2	Apricot Vlaai L	16,25	9,64	10
3	Fruit Vlaai L	27,5	15,5	6
4	Butter Crumb Vlaai L	15,25	9,35	18
5	Apple Crumble Vlaai L	14,29	8,76	14
6	Cherry Vlaai S	9,95	6,4	30
7	Apricot Vlaai S	10,5	5,1	32
8	Fruit Vlaai S	14,99	7,5	16
9	Butter Crumb Vlaai S	10,99	6,87	31
10	Apple Crumble Vlaai S	10,5	6,5	21

Table 3: Probabilistic mass for number of cakes bought by one customer.

Number of cakes	1	2	3
Probability	0.7	0.25	0.05

the code. Note that α is a free, adaptable parameter, and it is set to 0.5 in our experiment run. Ideally, we can calculate the stocking policy as a function of that free parameter. The process is illustrated in Figure 2.

5. We repeat this simulation for 100 days and optimize the total profit based on the average.

2.2 Simulation method and result

The simulation is done on a DELL laptop with Inter Core 7-10610U CPU@1.80GHz and 16GB memory using Python. First, we examine how the total profit changes with different stocking strategies by varying the ratio of initial stocks and the demands of the cakes, see in Figure 3.

It can be seen in Figure 3 that based on the demand data, shop owners can gain more profit to maintain a slightly higher stock level than the demand. Beyond a certain point, increasing the stock further leads to a decrease in the total profit due to the increased number of unsold products. It is worth noting that the relationship between total profit and the demand shown in Figure 3 suggests an approximate quadratic relationship between them, worth digging into further.

Next, we aim to optimize the total profit over all possible stocking options. After small exploratory runs, we estimate that the initial stocks of each cake individually should lie within the interval $[0, 60]$. These bounds can always be easily adjusted and refined if more data and restrictions are taken into account. Given that the objective function we are optimizing

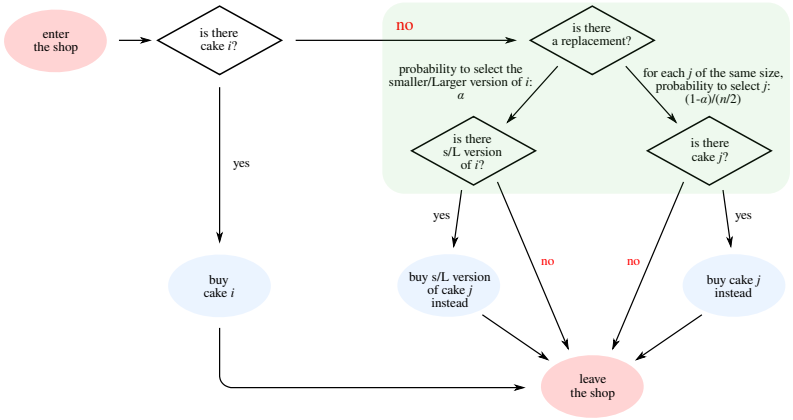

Figure 2: Flow chart of stochastic simulations.

cannot be expressed analytically and includes randomness, a heuristic algorithm to solve this optimization problem is more suitable for this scenario. In our experiments, we chose Genetic Algorithm as our optimization method (Mitchell, 1966). The number of generations is set as 100, the mutation rate is 0.1, and the crossover rate is 0.8. The maximal profit achieved is 1222 euros under the stocking strategy shown in Table 4.

In Figure 4, we compare the demand, initial stock, and leftover for each cake, and we can see that interestingly, in the best stocking strategy achieved (listed in Table 4), the demand for a certain product can outnumber the initial stock. This is due to the introduction of substitution rules among products. Also, note that this latter result is influenced by our choice to consider optimization of stocking policies based on the customer behavior and sales averaged over 100 days. A more comprehensive and future study should aim to incorporate higher moments of profit distribution into the optimization process rather than focusing only on the average quantities.

Table 4: Optimal Stock Levels of Cakes

Cake Id	1	2	3	4	5	6	7	8	9	10
Optimal stock	29	15	9	29	15	41	35	22	22	29

3 Probabilistic simulation

A third approach that was considered is the probability density evolution method. This is a general method that can be used to obtain accurate answers for probabilistic problems involving some kind of dynamics. It is similar to Monte Carlo simulation in that the state of a system and its evolution over time is calculated. However, instead of simulating one or more realizations, moving a single state through time, in the probability density evolution

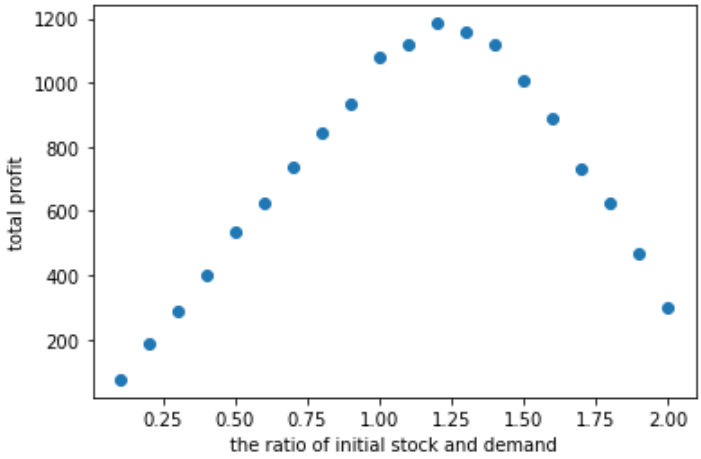

Figure 3: Total profit as a function of the initial stock. Values on the horizontal axis are the multiplicative factors to the initial stock based on the demands listed in Table 2, and approximated to the nearest integer.

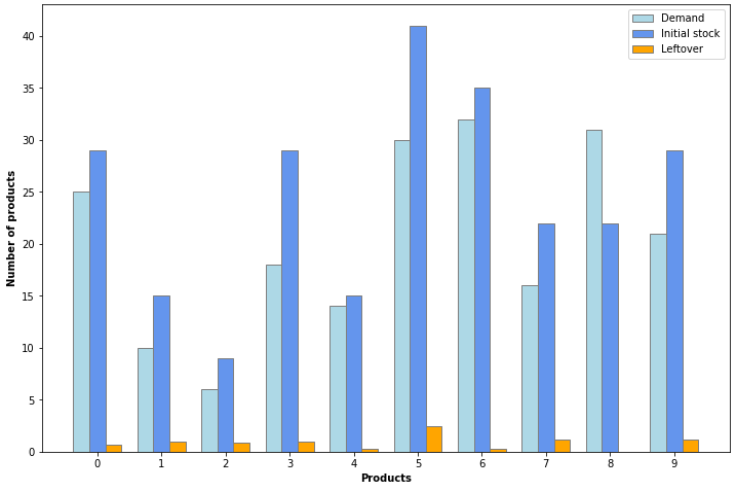

Figure 4: Comparison of demand, stock, and leftover under optimal profit averaged over 100 days of sales.

approach, we keep track of all possibilities and instead evolve the probability distribution of finding the system in different states. In other words, this is a type of probabilistic simulation in which all possible outcomes are considered and kept track of.

The probability density evolution method is usually formulated in a continuous setting (Li, 2016) and can seem somewhat intimidating, but it becomes rather straightforward in a discrete setting, Indeed, if we consider the state of the bakery to be fully described by the number of items $q = (q_1, \ldots, q_n) \in \mathbb{N}_0^n$ currently in stock, the probability density evolution approach fits the problem quite naturally. Since the stock levels $q_i \geq 0$ are discrete, we can expect the probability density evolution approach to generate exact results.

The approach used here is summarized in Algorithm 1. Starting from given initial stock levels $q_0 = (q_{0,1}, \ldots, q_{0,n}) \in \mathbb{N}_0^n$, the actions of a sequence $k = 1, 2, \ldots$ of customers are evaluated, leading to changes of the stock. The number of customers K is described by a probability distribution. For example, we can use the negative binomial distribution $f_K(k) = \Psi(k)$ from above and its distribution function $F_K(k) = \text{Prob}(K \leq k)$. We iterate over the customers one by one. Since this is a probabilistic simulation, we need to consider the probability

$$p_e(k) = \text{Prob}(K \geq k) = 1 - \text{Prob}(K \leq k-1) = 1 - F_K(k-1), \tag{5}$$

that there exists a k-th customer and its inverse $1 - p_e(k)$. The number of customers is inherently stochastic, so we iterate over a potentially infinite number of customers. However, the probability $p_e(k)$ falls off quickly for large values of k, and in practice, we stop the simulation once $p_e(k) < \epsilon$ for some small constant ϵ. For example, a typical value would be $\epsilon = 10^{-8}$, the square root of machine precision (Press, 2007).

If the evolution of stock levels takes place independently of each other, we could use vectors for each stock to keep track of the probability mass distribution of the level of each item. However, since we want to consider questions of dependence between items, we need to consider the joint distribution of the stock levels of all (correlated) items. We therefore represent their joint probability mass distribution by $Q = Q_{i_1,i_2,\ldots,i_n}$, where each index i_k runs from 0 (no stock left) to the initial stock level $q_{0,k}$. Computationally, Q is represented as a n-dimensional array of size $q_{0,1} \times \cdots \times q_{0,n}$. Obviously, the size of Q increases exponentially in the number of dependent items we are tracking, and this curse of dimensionality is a limitation of the method. In the following, we are mostly concerned with the case of two distinct items, so for reasons of clarity, let us assume this from now on and write $Q = Q_{i,j}$.

The initial stock level is represented by

$$Q_0 = \begin{cases} 1.0, & \text{for } (i,j) = (q_{0,1}, q_{0,2}), \\ 0.0, & \text{else.} \end{cases} \tag{6}$$

For each customer, the matrix $Q = Q_{ij}$ is then updated to take into account the actions of the customer. For example, for the first customer ($k = 1$), we calculate:

$$Q_1 = p_e(1)\Delta Q_0 + (1 - p_e(1))Q_0. \tag{7}$$

Here $\Delta Q_0 = \Delta(Q_0)$ describes the result of the actions of the customer, described below, whereas the second term represents the case that no customer is coming to the shop during this period (with probability $1 - p_e(1)$).

Algorithm 1: Probabilistic simulation of stock levels

Input: Initial stock levels $q_0 \in \mathbb{N}_0^n$ for n different stocks, a set of customers with preferences f_i, switching matrices a_{ij}, and purchase matrices b_{ij}
Output: Stock level probability distribution $Q = Q_{i_1,\ldots,i_n}$ at end of period

$Q \leftarrow 0^{q_0}$; // Empty array, one dimension for each item
$Q[q_0] \leftarrow 1.0$; // Initial stock has probability 1
$k \leftarrow 0$; // Number of customers
while *True* **do**
  $k \leftarrow k + 1$; // Consider another customer
  for *all different customer types* **do**
    $p \leftarrow p_e(k)$; // Probability of k-th customer of this type
    // Initialize temporary array Q_1
    $Q_1 \leftarrow (1 - p)Q$; // No customer, no change in stock
    // Consider all currently possible stock levels
    for *all q such that $Q[q] > 0$* **do**
      // Consider all customer preferences
      for *all $i \in \{1, \ldots, m\}$* **do**
        $p_1 \leftarrow p * f_i * Q[q]$; // Probability of this
        $d \leftarrow b_i$; // Customer demands these items
        if $d \leq q$ **then** // Purchase possible?
          $Q_1[q - d] \leftarrow Q_1[q - d] + p_1$; // Lower stock
        else
          // Consider all alternatives (incl. not switching)
          for *all $j \in \{1, \ldots, m\}$* **do**
            $p_a \leftarrow p_1 * a_{ij}$; // Probability of this
            $d \leftarrow b_j$; // Customer demands these items
            if $d \leq q$ **then** // Alternative purchase possible?
              $Q_1[q - d] \leftarrow Q_1[q - d] + p_a$; // Lower stock
            else
              // $i = j$ always goes here
              $Q_1[q] \leftarrow Q_1[q] + p_a$; // No change
            end
          end
        end
      end
    end
    $Q \leftarrow Q_1$; // Update Q
  end
  if $p_e(k) < 10^{-8}$ *for all customer types* **then** // Terminate?
    return Q;
  end
end

The customer behavior is represented by the matrix ΔQ_0, which results from operating on each (non-zero) entry of Q_0.

3.1 Customer modeling

In the linear framework of the probability density evolution method, we can add the effects of different customers, weighting them by the probability of each customer type, for example. The approach chosen here is to model different streams of customers, each with their own distribution F_{K_i}. This allows us to model different populations of customers that arrive independently of each other at the shop, with potentially different numbers for each type.

Modeling customer behavior is the key to being able to obtain useful results and see the dependence on customer demand in effect. We would like a model that is straightforward and relatively simple to implement yet sufficiently flexible to allow for understanding behaviors that cannot be described by the simple independence assumption. Here we propose to describe customer behavior in the following way:

1. Each customer has a certain number of potential *purchases* that she wants to do on any given day. The preference for each of these purchases is modeled by a discrete function f with values $f_i, i = 1, 2, \ldots, m$, where m is the number of different possible purchases. The preferences are assumed to sum up to 1, $\sum_{i=1}^{m} f_i = 1$. In other words, f is a probability mass function of the customer preferences over the set of potential purchases. Computationally, we represent this by a vector (array).

2. Each purchase is not limited to a single item but can be a multiset of items (where items can also be demanded multiple times). We model this by discrete functions $b_i : \{1, \ldots, n\} \mapsto \mathbb{N}_0$, one for each potential purchase of the customer. The totality of these functions can be represented by a matrix $B = b_{ij} \in \mathbb{N}_0^{m \times n}$, where the index i runs over all potential purchases, and the index j runs over all items $j = 1, \ldots, n$. Importantly, we assume that a purchase is only possible if all the demanded items (from the multiset b_i) are in stock. If one or more items are not in stock in sufficient quantity, the purchase will fail. This is the main mechanism that allows us to model (strict) dependence between items.

3. In addition, we assume a set of switching functions $a_i, i = 1, \ldots, m$ that for each potential purchase describe the possibility of the customer switching to a different purchase, in case the initially desired purchase is not possible. The totality of these functions can be represented by a matrix $A = a_{ij} \in \mathbb{R}^{m \times m}$. The a_{ij}-th entry is the probability that the customer tries to purchase the j-th multiset of items if the i-th multiset is not available. The diagonal a_{ii} is the probability that the customer does not want to switch and that the sale will fail if the i-th purchase is not available (because one or more of the desired items are not in stock), and thus each row $\sum_{j=1}^{m} a_{ij} = 1$ sums to one. We only allow for a single switch here, although, in principle, multiple switches could be analyzed.

This approach, in principle, allows for a model of arbitrarily complex customer preferences, although it might become somewhat unwieldy if customers have a lot of indifferences

between items while at the same time requiring a specific number of items in a single purchase. For a small number of different items as here, however, it should be able to cover most situations of interest.

Example 1: A simple customer. Let us call a customer *simple*, if she only performs single item purchases. These purchases are independent of each other, and this customer is represented by her preferences $f_i, i = 1, \ldots, n$, where we use $m = n$ since there are n different items and thus $m = n$ different single item purchases. The switching distribution is trivial, $A = E_n$, where E_n is the unit $n \times n$-matrix, as is the purchase matrix $B = E_n$.

Example 2: A flexible customer. Let us call a customer *flexible*, if she only performs single item purchases, all independent of each other, but potentially switches (once) to a different item if an initially desired item is not available. This customer is still represented by a preference vector $f_i, i = 1, \ldots, n$ and a trivial purchase matrix $B = E_n$, but the switching matrix $A = (a_{ij})$ is now non-trivial.

Example 3: A correlated customer. Let us call a customer *correlated*, if she desires a single multi-item purchase, where all items must be in stock or the whole sale fails. This customer only has a single desired purchase, so $m = 1$ and thus $f_1 = 1$. The switching matrix is trivial, $A = a_{11} = 1$, and the desired items are given by a single multiset $b_1 = b_{1j}$, where $j = 1, \ldots, n$ and $b_{1j} \in \mathbb{N}_0$ is the number of cakes of j-th type that the customer needs.

3.2 Demand calculation

The probability density evolution approach allows us to calculate the demand distribution exactly. The customer behavior is computationally an operator $\Delta : \mathbb{R}^{q_0} \mapsto \mathbb{R}^{q_0}$ that moves the probability Q of stock levels around. To obtain the demand distribution, we assume sufficient stock levels so that we do not ever run out of stock and all desired purchases are possible. In practice, we realize this by using a sufficiently large initial stock q_0. If the demand calculation detects that stock levels are not sufficient, we let the simulation fail and retry with increased initial stocks. A simple modification/simplification of Algorithm 1 can be used for this.

Example 4: Demand distribution of simple customer. Let us consider a simple customer choosing between two items with preference vector $f = (2/3, 1/3)$. Let us use a negative binomial distribution. Assume a customer rate $\mu = 20$, this leads to $N = \mu \frac{b}{1-b}$, so for $b = 0.99$ (see above) we use $N = 1980$. Simulation shows that an initial stock $q_0 = (28, 22)$ is sufficient to satisfy all sales. Figure 5 shows the initial stock distribution and its final form after a full day of customers. After $k = 50$ iterations, the distribution has converged (to the used numerical accuracy). In other words, we do not need to consider more than 50 customers in this case.

3.3 Revenue calculation and optimization

Calculating the revenue distribution and its expectation from the final stock level distribution is straightforward. If the initial stock level was $q_0 \in \mathbb{N}^n$ and the final stock level is q, this

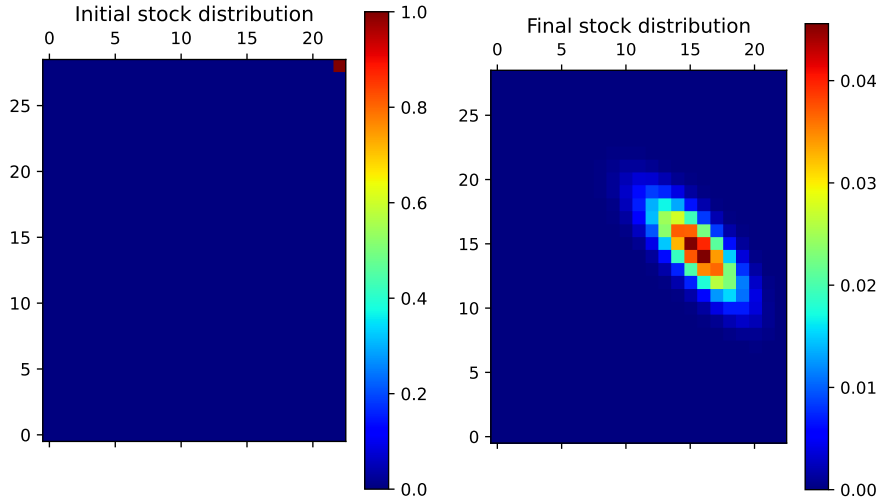

Figure 5: Stock distribution during probability density evolution method. Left: Initial stock distribution for an initial stock $q_0 = (28, 22)$. Right: Final stock distribution after $k = 50$ potential customer actions have been simulated. See text for details.

means that $q_0 - q \in \mathbb{N}_0^n$ items of each type have been sold. The overall revenue is then $r(q) = (q_0 - q) \cdot p - q_0 \cdot c = q_0 \cdot (p - c) - q \cdot p$, where the dot signifies the scalar product with the prices and costs vectors p and c.

The expected revenue is thus the expectation of $r(q)$ over all possible final stock levels, weighted by their probability of occurrence Q_q. Moreover, since we have access to the full distribution Q of all possible stock levels at the end of the period, we can easily calculate the variance and other indicators of interest. With some slight modifications to the method (not shown here in detail), we can similarly evolve and keep track of the probability distribution of the number of satisfied customers or the number of lost sales for each item during a day.

In order to find the optimum stock levels, we first establish a *theoretical baseline* by assuming independence of the stocks. If we add the rates of the various customer types modeled, multiplied by their preferences f_i for each item, we obtain an effective overall demand intensity for each stock. Using the inverse of the negative binomial distribution function, we can calculate these theoretical stock levels. Of course, these will usually not be optimal in case customers are modeled with dependence or switching.

It is currently unclear how to optimize the initial stock level efficiently, but we have implemented an exhaustive search that starts at a given stock level $q_0 \in \mathbb{N}_0^n$ and then successively tries to increase the level of the i-th stock for all $i = 1, 2, \ldots, n$. If the revenue increases while doing so, the new stock level becomes another target for optimization, and the procedure is repeated. In fact, a queue is used to keep track of potential candidates for further exploration, and the algorithm terminates when the queue has become empty. The only assumption that

is made is that there is a clear path from the initial level q_0 to the optimal level q_*, i.e., that there is a sequence of intermediary stock levels $q_0 < q_1 < \cdots < q_k < q_{k+1} = q_*$ where each $q_i, i = 0, 1, \ldots, k$ differs from q_{i+1} by an increase of exactly one stock by one unit, while the revenue also increases, $r(q_i) < r(q_{i+1})$. The default is to use $q_0 = 0 \in \mathbb{N}_0^n$ as the starting point for the search, but a larger q_0 obviously results in a somewhat faster search and might thus be desired in some cases.

While the stated assumption seems a natural assumption, leading to a straightforward algorithm, one complication is multi-item purchases. In order to accommodate these, the algorithm queues additional candidates for further exploration that are further away from the currently explored q by more than one increase in stock levels. The number of increases considered is called the *depth* of the search and a parameter of the algorithm. In this case, we modify the above assumption such that each q_i differs from q_{i+1} not by a single unit but by an increase of stocks up to the maximum number of items that can be acquired in a multi-purchase purchase by any one customer. Under this assumption, it is clear that the optimal solution will be found if the depth is equal to (or larger) than this number. The algorithm is shown as Algorithm 2.

3.4 Examples

In this section, a number of scenarios are explored. The code for running these examples can be found online[1].

Example 5: A simple customer. Let us return to the above simple customer with preferences $f = (2/3, 1/3)$ for two items. The independent optimum stock levels are $q_0 = (12, 5)$ with a revenue of $r(q_0) = 55.64$. The optimization finds an optimum at $q_* = (13, 5)$, with a revenue of $r(q_*) = 56.04$, see Figure 6. These two solutions should be identical, which they are not exactly — but this is probably just a numerical issue. When trying the same with a larger rate, e.g. $\mu = 30$, we obtain the same optimum level $q_0 = q_* = (19, 8)$, with expected revenue $r(q_*) = 88.92$.

Example 6: A flexible customer. Let us assume a flexible customer with switching matrix a_{ij}. If we assume strict switching $a_{12} = a_{21} = 1$, the customer will always try to buy the other item if his first choice is not available. It is expected that the optimum stock levels then depend on the profit margins, which are here $p - c = (4, 3)$, so we expect the optimum to mainly feature the first stock. Indeed, optimization leads to $q_* = (20, 0)$ with a revenue of $r(q_*) = 73.86$. The independent solution still retains $q_0 = (12, 5)$ with the revenue of $r(q_0) = 62.85$, so is clearly inferior (as expected). Note that the revenues are larger than in the case of an independent customer (Example 5).

If we now consider partial switching, say $a_{12} = a_{21} = 0.5$, the optimum stock level obtained is $q_* = (14, 5)$, with a revenue $r(q_*) = 63.33$, which is a lot closer to the independent solution. These customers cannot be so easily convinced to substitute the more profitable product for the other one if that is out of stock, so this is expected, although it is somewhat surprising to see these numbers so close to the independent solution.

[1] Python code available from: https://github.com/muskulus/swi2024

Algorithm 2: Optimization of stock levels

Input: Initial stock levels $q_0 \in \mathbb{N}_0^n$ for n different stocks; a set of customers with preferences f_i, switching matrices a_{ij}, and purchase matrices b_{ij}; item prices p_i and costs c_i; the search depth d; the maximum stock level considered $q_{\max}$

Output: Stock level $q_* \in \mathbb{N}^n$ with largest revenue, $\forall q \leq q_{\max} : r(q) \leq r(q_*)$

Data: Queue C; // Storing candidates for further exploration

Data: $V \in \mathbb{R}^{q_{\max}}$; // Keeping track of all revenues already seen

Procedure `AddFrom`(q, *depth*) // Recursive subroutine

- $current \leftarrow V[q]$;
- **for** $i \in \{1, \ldots, n\}$ **do**
  - $q[i] \leftarrow q[i] + 1$; // Increase the i-th stock level by one
  - **if** $V[q] = -Inf$ **then** // Not explored these stock levels yet
    - $V[q] \leftarrow r(q)$; // Perform simulation and calculate revenue
    - **if** $V[q] > current$ **then** // Only if there is improvement
      - C.put(q); // q is candidate for further exploration
    - **end**
  - **end**
  - // Try additional stock increases
  - **if** $depth > 1$ **then** `AddFrom` (q, depth - 1);
  - $q[i] \leftarrow q[i] - 1$;
- **end**
- **return**

Algorithm `Search`(q_0, ...)

- $V \leftarrow -\text{Inf}$; // Revenue of all stock levels currently unknown
- best, $V[q_0] \leftarrow r(q_0)$; // Calculate revenue of starting point
- $q_* \leftarrow q_0$; // Starting point is current best solution
- `AddFrom` (q_0, depth); // Add initial candidates from q_0 to C
- **while** C *is not empty* **do**
  - $q \leftarrow$ C.pop(); // Remove next candidate from queue
  - // All candidates in queue have been evaluated, $V[q] > -\text{Inf}$
  - **if** $V[q] > best$ **then**
    - best $\leftarrow V[q]$;
    - $q_* \leftarrow q$;
  - **end**
  - `AddFrom` (q, depth); // Add candidates for exploration to C
- **end**
- **return** q_*

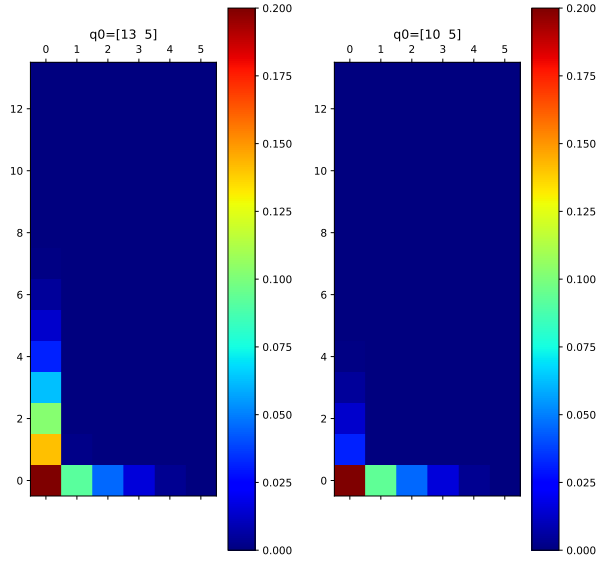

Figure 6: Final stock distribution during probability density evolution method with limited stock. The vertical axis corresponds to the levels of the first stock, and the horizontal axis corresponds to the levels of the second stock. Left: Optimal initial stock $q_0 = (13, 5)$, leading to an expected revenue of 56.0. Right: Smaller initial stock $q_0 = (10, 5)$, leading to an expected revenue of 50.9. The results are plotted on the same domain as the left plot for comparison purposes. One can see how there is less probability of one or more stock remaining for the first item, compared to the previous case, as expected.

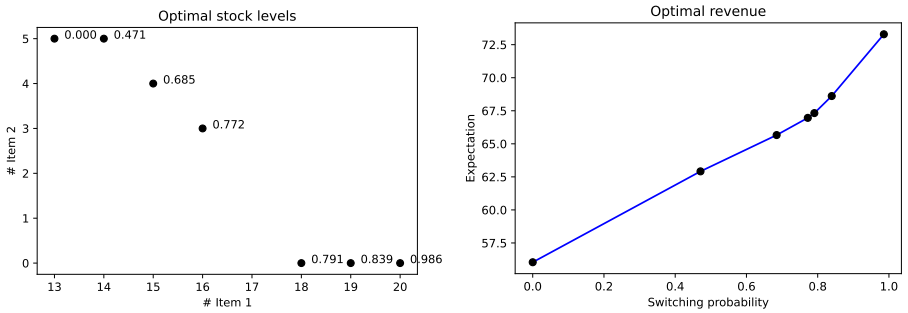

Figure 7: Results for varying the switching probability $a_{12} = a_{21}$. Left: Optimal stock levels for different switching probabilities. Larger switching probabilities correspond to points further on the right (with more of the first stock and less of the second, less profitable one). Right: Expected revenue for various switching probabilities α under optimal stocks. Notice the change in slope around $\alpha \approx 0.79$, which corresponds to the jump in the left plot (see text for details).

Indeed, plotting the optimal stock levels for different values of switching probability $\alpha = a_{12} = a_{21}$ in Figure 7 shows an interesting phenomenon: For small levels of α (left side of the left plot) the optimal stock levels are close to the one for $\alpha = 0$ from Example 5 ($q_* = (13, 5)$), whereas for large levels of α the optimum is obtained by stocking only the first, more profitable item ($q_* = (18, 0)$). Interestingly, the optimal stock level is quite sensitive to the switching probability and changes rapidly around a certain value of α. Here this value is approximately $\alpha \approx 0.79$, where the optimal stock level jumps from $q_1 = (16, 3)$ for $\alpha = 0.79$ to $q_2 = (18, 0)$ for $\alpha = 0.80$. Bisecting the value of α numerically leads to an estimate of $0.790630588 \leq \alpha^* \leq 0.790630589$ for the critical value where the jump occurs, so this is indeed a very sudden change. Moreover, it is unclear if the optimal stock level takes on other intermediate values between the two values q_1 and q_2 for values of α inside the bracketed interval.

Example 7: A correlated customer. Let us assume a customer that only buys the two stocks together - or nothing. We halve the rate of appearance so that, on average, the same number of items are demanded during a day as in the previous examples. The optimal stock level is then $q_* = (9, 9)$ with a revenue of 59.45.

Example 8: A general customer. Let us assume a customer that either buys the first item or buys the two stocks together, with equal probability but with no switching possible. If we adjust the rate of appearance by a factor of $2/3$, the optimal stock level is $q_* = (13, 6)$ with a revenue of $r = 51.57$. The optimal stock level is slightly larger than in the case of a simple customer (Example 5), but the revenue obtained is significantly less, even though, on average, both purchase the same items. In other words, it seems that not only should we stock slightly more items if some purchases are correlated, but also the revenue will be somewhat

Example	q_*	$r(q_*)$	β_*	Description
5	(12,5)	55.64	0.447, 0.422	Independent solution
	(13,5)	56.04	0.520, 0.422	Simple customer
6				Flexible customer (switching)
	(13,5)	56.04-62.92	0.520, 0.422	α = 0.000–0.470
	(14,5)	62.92-65.66	0.516, 0.395	α = 0.471–0.684
	(15,4)	65.66-66.96	0.508, 0.367	α = 0.685–0.772
	(16,3)	66.96-67.32	0.509, 0.356	α = 0.773–0.790
	(18,0)	67.32-68.61	0.472, 0.000	α = 0.791–0.839
	(19,0)	68.61-73.29	0.540, 0.000	α = 0.840–0.985
	(20,0)	73.29-73.86	0.540, 0.000	α = 0.986–1.000
7	(9,9)	59.45	0.358, 0.358	Correlated customer
8	(13,6)	51.57	0.647, 0.503	General customer
9	(11,8)	52.32	0.558, 0.568	Multiple customers
10				Simple customer (service level)
	(17,5)	42.56	0.916, 0.423	β = 0.90
	(18,5)	36.93	0.964, 0.423	β = 0.95
	(19,5)	31.05	0.996, 0.423	β = 0.99
	(21,5)	19.10	1.000, 0.423	β = 1.00

Table 5: Overview of results from considered examples. In all cases $p = (10, 13)$, $c = (6, 10)$, and the effective rate of customers is $\mu_{\text{eff}} = 20$. Simple customers have preferences $f = (2/3, 1/3)$ for purchasing one of the two items. Parameter α is the switching probability; β is the required service level for the first item, and β_* are the actual service levels obtained. For the flexible customer, the reported service levels are for the first value of α in each interval.

smaller than if all purchases are independent. This might be important when considering overall business profitability.

Example 9: Multiple customers. Let us study another case where we expand on the last two examples. Consider the same general and correlated customers and add them together so that we have these two different customer types alternatingly visiting the shop. We adjust their appearance rates, so that on average the same number of customers enter the shop, as before. The optimal stock level is now $q_* = (11, 8)$ with an average revenue of $r = 52.32$, so lies right between the two cases found in Example 7 and Example 8.

Example 10: Service level optimization. For this final case, we have considered optimizing the *service level*, i.e., the probability that a certain item does not run out of stock during a given day. This information is readily available[2] from the stochastic simulation, with just a

[2]There is some ambiguity in what we mean by service level when considering customer switching probabilities. Traditionally, we would define it as the probability of no lost sale for the item in question during a given period.

few simple modifications of the code (not shown). For the optimization, we use a simple penalty method. The service level constraint puts very strict requirements on the stock levels, though. For example, if we require the service level β of the first item to be $\beta = 0.90$, an initial stock of $q_0 = (17, 5)$ is needed, and the profit shrinks to just $r(q_0) = 42.56$. Even higher service levels lead to even stricter requirements, and the profits become even smaller, see Table 5. Therefore, in general, service levels are relatively small (around 50 percent) for optimal solutions when we do not impose such a requirement.

3.5 Discussion

We employed several approaches to explore optimal stocking strategies for cakes in small bakeries, aiming to maximize profits. Our approach involved conducting computationally efficient large-scale simulations to mimic sales on a typical day, considering variations in customer behavior. This framework enabled us initially to calculate the average profit derived from specific stocking strategies. Subsequently, we compared various strategies against each other to identify the one that optimizes profit while minimizing waste, employing random search techniques such as genetic algorithms (Mitchell, 1966).

Our findings suggest that, for straightforward scenarios where stocking levels are directly proportional to demand estimates from typical sales data, increasing inventory by approximately 25% is beneficial. However, surpassing this threshold leads to an increase in unsold cakes with associated production costs, which are not counterbalanced by additional sales. To develop more sophisticated and effective stocking strategies, it will be crucial to enhance our computational models to consider the variability in customer numbers, as well as the wide range of customer behaviors and preferences. Insights into these aspects, such as the distribution of customer purchases and combinations of cakes bought in a single transaction, can be obtained from sales data and payment receipts.

Moreover, our optimization algorithm can be further refined to include additional considerations. For example, if a bakery is known for a specific product like “Limburgse vlaai”, it may prioritize ensuring the supply of these items, accepting potential financial losses to maintain its reputation for having such a product readily available. Adding this feature to our computational models is straightforward and will be the topic of future investigations.

The probability density evolution method seems to be a useful approach to finding the demand distribution and the expected revenue for a given initial stock level exactly. That said, what can we learn from the example scenarios studied? Table 2 gives an overview of the results from the examples.

The main conclusion seems to be the following: If customers are amenable to switching between different items (with different profit margins), the expected revenue will be larger than if customers have strict purchase preferences. This is obvious since if customers are more flexible in their purchases, then if one item has run out of stock, there might still be a sale for a substitute item that would otherwise not go through. Interestingly, the optimal stock levels change quite abruptly — there is a certain level of switching probability after which it

However, it is somewhat unclear what this means when customers can switch to different items. Therefore we have implemented the service level here as the inverse of the probability that the item in question goes from a non-negative stock to zero stock (a so-called “stock-out”) during a given period.

makes sense for the shop to only stock the more profitable item(s). The main conclusion here is that it makes sense for shops to motivate customers to consider switching to (preferably higher profit) items if their original purchase is not available. Related to this, it could make sense for shops to motivate customers to buy items together, as this makes the demand more predictable and potentially somewhat easier to optimize. However, it has been seen that correlated demand can also reduce profits, depending on how profit margins and customer preferences are aligned.

Some limitations of the probability density evolution method should be mentioned: It is somewhat time-consuming, and the computational effort rises dramatically in the number of stocks that are considered together. That said, it is possible to find the optimal stock levels for a small number of items (say $n = 3$) in a few minutes on a standard computer. Some assumptions have been made to simplify the problem, though that should be mentioned. The most important is that different customer types are assumed to arrive at the shop in a certain order: The purchases of the first customer are simulated for each customer type one after the other, before considering the next customer for each customer type. This means that if the arrival rates (the expected number of customers per day) are quite different for different types of customers, the results of the probabilistic simulation will not be representative. A better strategy would be to consider all (or most of) the different orders with which customers might arrive at the shop. However, it is currently unclear how to do this in a way that is computationally not too demanding. This interesting issue is thus left for future work.

Going back to the implemented model, there are also some interesting mathematical questions remaining on how to optimize the stock levels efficiently; the current exhaustive search strategy, while being able to find the optimal stock levels also in complicated cases (with $depth > 1$), is clearly inefficient. It has also not been proven that the assumption behind the optimization strategy (that there is always a path of increasing stock levels to the optimum) is always valid.

Finally, a general takeaway is that customer modeling is the most important issue in optimizing stock levels. It seems, therefore, advisable to spend more effort on understanding actual customer behavior. For example, future work might want to study the interesting problem of how to infer different customer types and their arrival rates from sales data.

4 Conclusion

This article has discussed multiple approaches for stocking strategy and the testing of given stocking strategies. Firstly, based on probabilistic considerations, we suggest to stock according to newsvendor, but to augment the probability distribution to take into account substitution sales as in section 1. We suggest simulations as in section 2 to test large-scale stocking strategies. To gain insight into a given stocking strategy with a small number of items, we suggest using probability evolution as in section 3. For both large-scale simulation and probability evolution, we have also provided tools for optimizing stocking strategy. In those sections, we have also discussed in more detail the merits of these simulations and provided directions for further research.

References

Kenneth J. Arrow, Theodore Harris, and Jacob Marschak. Optimal Inventory Policy. *Econometrica*, 19:250–272, 1951. doi: 10.2307/1906813.

Jie Li. Probability density evolution method: Background, significance and recent developments. *Probabilistic Engineering Mechanics*, 44:111–117, 2016. doi: 10.1016/j.probengmech.2015.09.013.

M. Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, Cambridge, MA, 1966.

William H. Press, editor. *Numerical recipes: the art of scientific computing*. Cambridge University Press, New York, 3rd edition, 2007.

Yan Qin, Ruoxuan Wang, Asoo J. Vakharia, Yuwen Chen, and Michelle M.H. Seref. The newsvendor problem: Review and directions for future research. *European Journal of Operational Research*, 213:361–374, 2011. doi: 10.1016/j.ejor.2010.11.024.

Stephen A Smith and Narendra Agrawal. Management of multi-item retail inventory systems with demand substitution. *Operations Research*, 48:50–64, 2000.