School of Business, Economics and Information Systems

Chair of Management Science/Operations and Supply Chain Management

---

Seminar in Management Science

Partial information

How to process and benefit from partial information in online resource allocation and other practical online applications

Supervisors:

Prof. Dr. Alena Otto, Catherine Lorenz

Edited by: Maya Marten, Aleksandra Petrenko

Matriculation number: 89416, 108448

Course of studies: M.Sc. Business Informatics, M.Sc. Business Administration

Semester: 3, 4

E-Mail: marten18@ads.uni-passau.de, petren01@ads.uni-passau.de

Passau, submission date: 19.07.2023

Contents

List of abbreviations, conventions and symbols

" $\triangleq$ " used for definitions
$(x)^+ \triangleq max\{x, 0\}$
Uppercase letters (e.g. $\overrightarrow{V}$): random variables
Lowercase letters (e.g. $\overrightarrow{v}$): realizations of random variables
deter.: abbreviation for deterministic

Figure 1: Abbreviations, conventions and symbols

1 Introduction

1.1 Motivation

Online algorithms are not only an important topic in management sciences, but also in other disciplines such as computer science or operations research. With the rise of electronic markets, there are more and more applications where online decision-making plays a role. Typical use cases are for example online routing, online revenue management or in the online resource allocation problem. In this paper, a particular online resource allocation problem is analyzed in detail.

1.2 Conceptual basics

In an online decision problem, individual model inputs are received sequentially. The moment the input is received, an irrevocable decision has to be made on each successive input (Ball and Queyranne (2009)). Typical objectives are to maximize revenue or to minimize costs.

There are two main performance metrics of online algorithms. The *regret* is the difference between the cost of the algorithm and the cost of the offline optimal static solution. Another metric is the *competitive ratio*, which is the maximum ratio between the cost of the algorithm and the cost of the offline optimal (dynamic) solution (Example: Andrew et al. (2013)). The metrics will be discussed further.

A distinction can be made between *adversarial* and *stochastic* models.
In an adversarial setting, the input sequence is controlled by an imaginary adversary and therefore cannot be predicted in any way. Many models designed for this case, assume that the adversary tries to generate the worst-possible algorithmic performance. This leads to conservative bounds and therefore to conservative online policies (Ball and Queyranne (2009)). In contrast, a stochastic setting assumes that the input follows a

certain probability distribution. The distribution can for example be derived based on historical data. This kind of models are known as random permutation models in the literature (Agrawal et al. (2014)). These models can perform well, despite the online setting. But demand can not always be perfectly predicted.
So, real world applications of online algorithms focusing only on adversarial input is rather pessimistic whereas purely stochastic models do not manage to fully capture uncertainty.
To a certain extent, there is often some information which allows us to predict future input and we can take an advantage of this data to improve the algorithm and our decisions. However, we should also account for unpredictability of some events. Therefore, recent research proposes mixed models that combine adversarial and stochastic approaches for online problems. This approach was also taken in the paper of Hwang et al. (2021), which is discussed in more detail in this paper. In Chapter 2 we extend the literature review by Hwang et al. (2021) and focus on mixed-models. Further, in Chapter 3 we revise the definition of the partially predictable demand arrival model, proposed by Hwang et al. (2021). Later on in Chapter 4, we examine the algorithms proposed there. We compare the two algorithms using a computational analysis.

2 Literature review

In this section, we extend the literature review by Hwang et al. (2021) focusing on various models combining both stochastic and adversarial components. We provide examples for different classical online problems, that is how we structure our research.

A stochastic shortest path problem is a problem where the goal is to find a shortest route - to reach the destination with a minimal total cost. In each of k episodes, the learner begins at a fixed state, sequentially selects an available action, exposes himself to a cost and randomly transits to the next state according to some probability distribution (Chen et al., 2021). A recent research of Chen et al. (2021) studies the stochastic short path problem with adversarial costs and known transition. The authors develop algorithms for full-information and bandit feedback setting (where at the end of each episode the learner receives only the cost of the visited state-action pairs) and derive minimax regrets for both settings, improving the existing suboptimal regrets. Neu et al. (2012) were pioneers in algorithms controlling stochastic and adversarial components at the same time. They consider learning in adversarial stochastic shortest path problems where the transition probabilities are unknown, the reward function is, however, fully observed after each finished episode for all state-action pairs. The expected cumulative regret the algorithm is of order $L|\mathcal{X}||\mathcal{A}|\sqrt{T}$ up to logarithmic factors, where L is the length of the longest path in the graph, $\mathcal{X}$ is the state space, $\mathcal{A}$ is the action space and T is the number of episodes.

Another problem widely discussed in papers with mixed models is an online convex optimization (OCO). In brief, an OCO algorithm, using the notation of Cutkosky and Boahen (2017), plays T rounds of a game where on each round the algorithm outputs a vector w_t in some convex space W and then receives a loss function $l_t : W \to \mathbb{R}$ that is convex. The objective of the problem is to minimize regret, which is the total loss of all rounds relative to w^*, the minimizer of $\Sigma_{t=1}^{T} l_t$ in W:

$$R_T(w^*) = \sum_{t=1}^{T} l_t(w_t) - l_t(w^*) \tag{2.1}$$

Cutkosky and Boahen (2017) derive a universal OCO algorithm performing well (with an $O(log^4(T))$ regret) in many stochastic settings and achieving an optimal $O(\sqrt{T})$ regret in adversarial settings (up to logarithmic factors). Shamir and Szlak (2017) offer another approach to mix adversarial and stochastic settings for an OCO with delayed feedback. They propose Online Learning with Local Permutations, where the learner can randomly permute the order of the loss functions, which were created by an adversary. This learning models natural situations where the exact order of the learner's responses does not matter, which allow to get better learning and regret performance by avoiding highly adversarial loss sequences.

A further online problem, a multi-armed bandit problem, can be described as follows: There is casino with K slot machines (one-armed bandits), in which each machine has a different and unexpected payoff. The objective is to sequentially choose the sequence of slot machines to play to maximize the expected total reward Chakrabarti et al. (2008). Lykouris et al. (2018) and Gupta et al. (2019) study stochastic bandits with adversarial corruptions, where the overall behavior is essentially stochastic but a small fraction of the rewards can be changed by an adversary. Lykouris et al. (2018) propose an algorithm robust to corruptions, which achieves the $\widetilde{O}(KC\sum_{i\neq i^*} 1/\Delta_i)$ regret, where K is the number of arms, C is total corruption, the gaps $\Delta_i = \sqrt{K/T}$, T is the number of rounds and $\widetilde{O}$-notation suppresses all dependence on logarithmic terms. However, Gupta et al. (2019) develops a better algorithm with the regret $\widetilde{O}(KC) + \widetilde{O}(\sum_{i\neq i^*} 1/\Delta_i)$. Lykouris et al. (2020) propose and study an adversarial scaling model. In this model, the mean reward $\mu_t(a)$ is calculated as $\mu_t(a) = q_t \cdot \theta(a)$, where $\theta(a)$ is an intrinsic quality of the arm (e. g. quality of the ad) and q_t is a complex function or pattern, which can be assumed to be chosen by an adversary (for example, seasonality or clickiness of users in display ads). The idea is to exploit the structure of $\mu_t(a)$ without estimating the q_t model directly.

The AdWords problem introduced by Mehta et al. (2005) has also been studied in the past years with the revolution of internet search engine companies. The problem investigates how a search engine company decides what ads to display with each query so as to maximize its revenue. The problem can be stated as follows: There are N

bidders, each with a specified daily budget b_i. Q is a set of query words. Each bidder i specifies a bid c_{iq} for query word $q \in Q$. A sequence $q_1, q_2 \dots q_M$ of query words $q_j \in Q$ arrive online during the day, and each query q_j must be assigned to some bidder i (for a revenue of c_{iq_j}). The objective is to maximize the total revenue at the end of the day while respecting the daily budgets of the bidders. Devanur et al. (2019) introduce an adversarial stochastic input model, in which the distributions are allowed to change over time. The adversary decides how to pick the distributions, and is even allowed to pick them adaptively. As an example of application, the distribution of requests for advertising shows different trends that change over time. Esfandiari et al. (2015) propose a robust online stochastic model that captures the nature of traffic spikes in online advertising. In addition to the stochastic input which can be well forecast, an unknown number of impressions that are adversarially chosen arrive.

To summarize, there are different approaches in the literature how the adversarial assumption can be weakened and how stochastic models can be modified to real-life examples. In further sections we will describe and analyze another mixed model proposed by Hwang et al. (2021) in detail.

3 Partially predictable demand arrival model

3.1 Preliminaries

In the following, the partially predictable demand arrival model is presented. The problem formulation is based on the model presented in Hwang et al. (2021). The model differs from previous demand arrival models in containing both an adversarial component and a stochastic component. Because the adversarial component represents the unpredictable demand and the stochastic component represents the predictable demand, the model is called partially predictable.

In order to specify the level of predictability of demand in our model, we introduce a parameter $\boldsymbol{p}$, with $0 < p < 1$. In the extreme case of $p = 0$, the model reduces to the pure adversarial model, whereas in the extreme case of $p = 1$ the model reduces to the pure stochastic model. Both have been studied in the literature before.

In the following model, we assume that the firm has a single resource - more precisely, $\boldsymbol{b}$ identical units of a product to sell. We furthermore specify a parameter $\boldsymbol{n}$, which specifies the number of periods the product is sold, with $n \geq b$. The parameter $\boldsymbol{\lambda}$ represents time steps of a time horizon normalized to 1: $\lambda = 1/n, 2/n, \ldots, 1$. A maximum of one customer demands one unit of the product at each period. The demand realizes sequentially.

We also assume that we can face exactly two types of customers: type-1 customers and type-2 customers. The type of customer is not obvious to the firm until the customer arrives. At this moment the firm must decide whether to accept or reject the customer. If the company accepts the customer, the one unit of the product is allocated and assigned to the customer. The customers differ only in how much revenue they generate (also see Table 3.1). By accepting a type-1 customer, the firm earns the most, namely \$1. By accepting a type-2 customer, the firm earns \$$a$, with $0 < a < 1$, and therefore $a < 1$. By not allocating a unit of the product to any customer, the firm earns nothing. The number of type-1 customers and type-2 customers in the sequence is represented by $\boldsymbol{n_1}$ and $\boldsymbol{n_2}$, respectively.

	Revenue
Type-1 customer	1
Type-2 customer	a, with $0 < a < 1$
No customer	0

Table 3.1: Revenue obtained from different customer types

The sequence of arrival $\overrightarrow{\boldsymbol{v}} = (v_1, v_2, \ldots, v_n)$, with $v_i \in \{0, a, 1\}$ is denoted as follows: The customers are represented by the revenue they generate if accepted. So a type-1 customer will be represented with 1 and a type-2 customer will be represented with a. The situation that no customer arrives at period i will be represented with 0.

The company's goal is to maximize its revenue.

Optimal offline solution

The optimal revenue for the offline solution $OPT(\overrightarrow{v})$ is the following: Accept all type-1 customers. If there demand fewer type-1 customers products than the firm has units to sell, the remaining units should be allocated to as many type-2 customers as possible. I.e. if $n_1 < b$, $min\{n_2, b - n_1\}$ type-2 customers should be accepted.

$$\boldsymbol{OPT(\overrightarrow{v})} = min\{n_1, b\} + a * min\{n_2, (b - n_1)^+\} \tag{3.1}$$

Example a):
Let's consider the following example, with $n = 10, b = 10, n_1 = 4, n_2 = 3, a = 0.6$ and $\overrightarrow{v} = (0, 1, 1, a, 0, a, 1, 0, a, 1)$.

The sequence of customers is represented as follows:

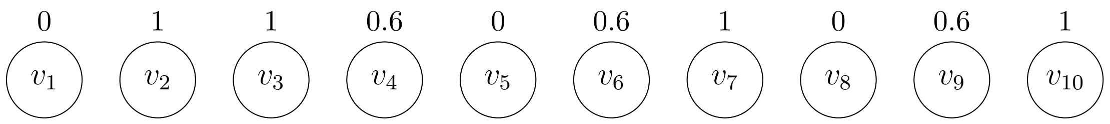

Figure 3.1: Example a): Offline Solution $OPT(\overrightarrow{v})$

The optimal offline solution for this problem instance is:

$$\begin{aligned} OPT(\overrightarrow{v}) &= min\{4,10\} + 0.6 * min\{3,(10-4)^+\} \\ &= min\{4,10\} + 0.6 * min\{3, max\{10-4,0\}\} \\ &= min\{4,10\} + 0.6 * min\{3,6\}\} \\ &= 4 + 0.6 * 3 \\ &= 5.8 \end{aligned}$$

The maximum revenue, the firm can earn in this example is \$5.8.

Reasonable online solution

Similar to the optimal offline solution, an upcoming type-1 customer should always be accepted. In case of an arriving type-2 customer the firm has to face the following trade-off: Because there is only a limited number of products b to sell, accepting too many type-2 customer could end up in rejecting potential future type-1 customers. But accepting too few type-2 customer, however, could end up in leaving unsold products. The online algorithm should try to balance this trade-off. The solution of the online algorithm is represented by $ALG(\overrightarrow{v})$.

3.1.1 Model

An imaginary adversary first sets an initial sequence $\boldsymbol{\overrightarrow{v_I}} = (v_{I,1}, v_{I,2}, \ldots, v_{I,n})$, with $v_{I,j} \in \{0, a, 1\}$, for $1 \leq j \leq n$.
A subset of customers $\boldsymbol{S}$ represents the *stochastic group* and won't follow the order determined by the adversary. The subset of customers $\boldsymbol{A}$ represents the *adversarial group* - all customers which don't belong to the stochastic group S. The customers in A follow the initial adversarial sequence $\overrightarrow{v_I}$.

The customers in S join the subset independently and with the same probability p. They are permuted uniformly at random among themselves: $\sigma_s : S \to S$. This permutation determines the order of arrivals among the stochastic group. The customers in S follow this random order.
It is important to mention that the initial adversarial sequence is fixed without knowing which customers will later on belong to the subset S.

Example b):
In the following, we extend the example to include the stochastic customer group. Let's consider $p = 0.5, S = \{3, 4, 6, 7, 8\}, \sigma(3) = 7, \sigma(4) = 3, \sigma(6) = 6, \sigma(8) = 4$ and $\sigma(7) = 8$.

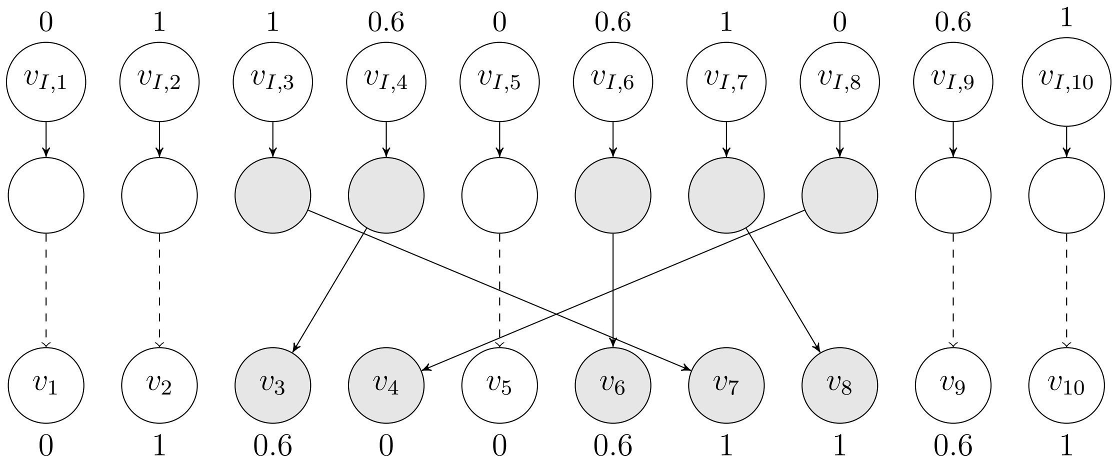

Figure 3.2: Example b): Stochastic Group

Notational Conventions

Given the initial customer sequence v_i, we define the *random* customer arrival sequence by $\overrightarrow{\boldsymbol{V}} = (V_1, V_2, \ldots, V_n)$. The corresponding *realization* of the customer arrival sequence by $\overrightarrow{\boldsymbol{v}} = (v_1, v_2, \ldots, v_n)$.
First, we present the notations regarding the random customer arrival sequence $\overrightarrow{V}$: The random number of type-j customers ($j = 1, 2$) arriving up to time λ is defined by $\boldsymbol{O_j(\lambda)}$. Furthermore, the number of type-j customers ($j = 1, 2$) in the stochastic group S arriving up to time λ is defined by $\boldsymbol{O_j^S(\lambda)}$. We define the realizations of $O_j(\lambda)$ and $O_j^S(\lambda)$ as $o_j(\lambda)$ and $o_j^S(\lambda)$, respectively. Because the algorithm is not able to distinguish between customers in the adversarial group A or the stochastic group S, the realizations of $O_j^S(\lambda)$ (the $o_j^S(\lambda)$) cannot be observed.

Second, we present further notations regarding the initial adversarial sequence v_I. The variable $\boldsymbol{\eta_j(\lambda)}$ ($j = 1, 2$) specifies the number of type-j customers among the first λn customers in $\overrightarrow{v_I}$. Both $\eta_j(\lambda)$ and n_j are deterministic. Furthermore, the variable $\tilde{o_j}(\lambda)$ is calculated by the formula $(1-p)\eta_j(\lambda) + p\lambda n_j$. Also the variable $\tilde{o_j}^S(\lambda)$ is calculated by $p\lambda n_j$. Both $\tilde{o_j}(\lambda)$ and $\tilde{o_j}^S(\lambda)$ are used as deterministic approximations for the parameters $O_j(\lambda)$ and $O_j^S(\lambda)$, respectively.

Example c) We calculated the above expressions for the parameters $\lambda = 7/10$ and $p = 0.5$. The results can be found in Table 3.3.

$o_1(7/10)$	2	two type-1 customers arrive up to time 7/10
$o_1^S(7/10)$	1	one type-1 customer in the stochastic group arrives up to time 7/10
$o_2(7/10)$	2	two type-2 customers arrive up to time 7/10
$o_2^S(7/10)$	2	two type-2 customers in the stochastic group arrive up to time 7/10
n_1	5	a total of five type-1 customers are in the initial customer sequence $\overrightarrow{v_I}$
$\eta_1(7/10)$	3	three type-1 customers are among the first 7 customers in $\overrightarrow{v_I}$
n_2	3	a total of three type-2 customers are in the initial customer sequence $\overrightarrow{v_I}$
$\eta_2(7/10)$	2	two type-2 customers are among the first 7 customers in $\overrightarrow{v_I}$
$\tilde{o_1}(7/10)$	3.25	deter. approximation of $O_1(7/10)$
$\tilde{o_1}^S(7/10)$	1.75	deter. approximation of $O_1^S(7/10)$
$\tilde{o_2}(7/10)$	2.05	deter. approximation of $O_2(7/10)$
$\tilde{o_2}^S(7/10)$	1.05	deter. approximation of $O_2^S(7/10)$

Figure 3.3: Example c)

4 Algorithms

In the following chapter we present two algorithms for the partially predictable demand arrival model. These algorithms will be compared through a detailed computational example in Section 4.3.

4.1 Non-adaptive Algorithm

4.1.1 General

The first presented algorithm $\boldsymbol{ALG_1}$ is non-adaptive because it does not adapt to the observed data. It uses precalculated dynamic thresholds: an **evolving threshold** $\boldsymbol{\epsilon}$ and a **fixed threshold** $\boldsymbol{\theta}$.

$$\boldsymbol{\epsilon(\lambda)} = \lfloor \lambda p b \rfloor \tag{4.1}$$

As explained in section 3.1.1, the customers in S are distributed uniformly over the time horizon. Since we expect a fraction p of customers to join the stochastic group, we can assume that a fraction p of the inventory can be assigned at an approximately constant rate. Therefore we calculate $\epsilon(\lambda)$ as stated above in equation 4.1, which evolves as λ increases over time. It is calculated by rounding off the product of *lambda*, the fraction of customers in S and the number of units to sell b.

$$\boldsymbol{\theta} \triangleq \frac{1-p}{2-a} \tag{4.2}$$

We have no information about the distribution of the arrivals of the other fraction $(1-p)$ of customers in A, but we want to avoid rejecting too many Type-2 customers right at the beginning. Because of that, another fixed threshold is defined in equation 4.2. A more detailed analysis of the formula is provided in the Appendix.

4.1.2 Pseudo code

The pseudo code for ALG_1 can be found below. For better programmability, the algorithm was slightly changed and an additional variable $rem.inv$ was introduced. It is used to track the products that can still be allocated to customers, i.e. have not yet been assigned to customers. We also make use of the variables $q_1, q_{2,e}, q_{2,f}$, which are used as counters for the Type 1 customers already accepted, the Type 2 customers accepted by the evolving threshold, and the Type 2 customers accepted by the fixed threshold, respectively.
In a first step, we initialize the variables $q_1, q_{2,e}, q_{2,f}$ with 0 and the variable $rem.inv$ with b. Furthermore, θ is calculated. No customer was accepted yet, therefore all product units b are still available. From time $\lambda = 1/n, 2/n, \cdots$ to 1, the respective customer $i = \lambda \cdot n$ is accepted, if there is still remaining inventory ($rem.inv > 0$) and one of the following three conditions is fulfilled:

- **Condition 1** (lines 5-7)
  The customer generates a revenue of 1 (i.e. the customer is of Type-1). If that is the case, the Type-1 customer gets accepted and therefore, q_1 is increased by 1.

- **Condition 2** (lines 8-10)
  The customer generates a revenue of a (i.e. the customer is of Type-2). Also, the number of already accepted Type-1 customers (q_1) and accepted Type-2 customers by the evolving threshold rule ($q_{2,e}$) does not exceed $\lfloor \lambda pb \rfloor$. If that is the case, the Type-2 customer gets accepted and therefore, $q_{2,e}$ is increased by 1.

- **Condition 3** (lines 11-13)
  The customer generates a revenue of a (i.e. the customer is of Type-2). Also, the number of already accepted Type-2 customers by the fixed threshold rule ($q_{2,f}$) does not exceed $\lfloor \theta b \rfloor$. If that is the case, the Type-2 customer gets accepted and therefore, $q_{2,e}$ is increased by 1.

If condition 2 and condition 3 are fulfilled simultaneously, the evolving threshold rule is prioritized.

4.1.3 Competitive ratio

ALG_1 achieves a competitive ratio of $p + \frac{1-p}{2-a} + O(\frac{1}{a(1-p)p}\sqrt{\frac{logn}{b}})$.

Algorithm 1 Non-adaptive Algorithm ALG_1

```
1: Initialize:
       $q_1 \leftarrow 0$
       $q_{2,e} \leftarrow 0$
       $q_{2,f} \leftarrow 0$
       $rem.inv \leftarrow b$
       $\theta \triangleq \frac{1-p}{2-a}$
2: for $\lambda = \frac{1}{n}$ to 1 do
3:     $i = \lambda \cdot n$
4:     if $rem.inv > 0$ then
5:         if $v_i = 1$ then
6:             $q_1 \leftarrow q_1 + 1$
7:             $rem.inv \leftarrow rem.inv - 1$
8:         else if $v_i = a$ and $q_1 + q_{2,e} < \lfloor \lambda pb \rfloor$ then
9:             $q_{2,e} \leftarrow q_{2,e} + 1$
10:            $rem.inv \leftarrow rem.inv - 1$
11:        else if $v_i = a$ and $q_{2,f} < \lfloor \theta b \rfloor$ then
12:            $q_{2,f} \leftarrow q_{2,f} + 1$
13:            $rem.inv \leftarrow rem.inv - 1$
14:        end if
15:    end if
16: end for
```

4.2 Adaptive Algorithm

4.2.1 General

The second presented algorithm $ALG_{2,c}$ is, in contrast to ALG_1, an adaptive algorithm. The thresholds in this algorithm are based on the observed data. The input parameter $c, c \in [0, 1]$ represents the competitive ratio. that the algorithm achieves. The competitive ratio of the algorithm is examined further in the Subsection about the competitive ratio.

The algorithm uses upper bounds on the total number of Type-1 and Type-2 customers to decide whether an arriving Type-2 customer should be accepted or rejected. The upper bound for the number of Type-1 customers in the sequence (n_1) is denoted as $\boldsymbol{u_1(\lambda)}$. It is calculated in the following way:

$$u_1(\lambda) \triangleq \begin{cases} b & \text{if } \lambda < \delta \\ \min\{\frac{o_1(\lambda)}{\lambda p}, \frac{o_1(\lambda)+(1-\lambda)(1-p)n}{1-p+\lambda p}\} & \text{otherwise} \end{cases} \tag{4.3}$$

Note, that the equation uses a variable $\boldsymbol{\delta} \triangleq \frac{(1-c)b}{(1-a)n}$, that is used to elaborate if enough data is observed. In case of an adaptive algorithm, that adjusts its thresholds dynamically based on the observed data, it is important that enough data is collected to adjust the thresholds properly. In the particular case of the partially predictable demand arrival model, it is important that the algorithm has enough time to adapt effectively to the data. Therefore, in order to achieve a certain performance with the Algorithm, it is crucial, that the inventory b is large enough. With increasing b, the algorithm can observe and adjust better to the data.

Similar to u_1, the upper bound for the number of Type-1 and Type-2 customers in the sequence $(n_1 + n_2)$ is denoted as $\boldsymbol{u_{1,2}(\lambda)}$ and is calculated in the following way:

$$u_{1,2}(\lambda) \triangleq \begin{cases} b & \text{if } \lambda < \delta \\ \min\{\frac{o_1(\lambda)+o_2(\lambda)}{\lambda p}, \frac{o_1(\lambda)+o_2(\lambda)+(1-\lambda)(1-p)n}{1-p+\lambda p}\} & \text{otherwise} \end{cases} \tag{4.4}$$

The adaptive algorithm does not distinct between a fixed and an evolving threshold. Instead, only one threshold is calculated, which can be found in Equation 4.5.
For notational convenience a new variable $\boldsymbol{\Phi} \triangleq \frac{1-c}{1-a}$ is introduced.

$$\lfloor \Phi b + c(b - u_1(\lambda))^+ \rfloor \tag{4.5}$$

The threshold is used to decide whether to accept or reject a Type-2 customer. A more detailed analysis of the threshold and the variables Φ and δ is provided in the Appendix 6.2.

4.2.2 Pseudo code

The pseudo code for $ALG_{2,c}$ can be found below. In a first step, similar to ALG_1, q_1 and q_2 are defined as counters for the number of type-1 customers and type-2 customers, respectively and initialized with 0. Further, variable Φ is initialized with $\frac{1-c}{1-a}$, δ with $\frac{\Phi b}{n}$ and again, like in ALG_1, $rem.inv$ is initialized with b.
From time $\lambda = 1/n, 2/n, \cdots$ to 1, the functions u_1 and $u_{1,2}$ are calculated by using the equations 4.3 and 4.4 (lines 3-8).

The respective customer $i = \lambda \cdot n$ is then accepted, if there is still remaining inventory $(rem.inv > 0)$ and one of the following three conditions is fulfilled:

- **Condition 1** (lines 12-14)
  The customer generates a revenue of 1 (i.e. the customer is of Type-1). If that is the case, the Type-1 customer gets accepted and therefore, q_1 is increased by 1.

- **Condition 2** (lines 15-17)
  The customer generates a revenue of a (i.e. the customer is of Type-2). Further, the upper bound $u_{1,2}(\lambda)$ calculated for the current λ on $n_1 + n_2$ is smaller than b. In this case, we can accept the Type-2 customer, because with the information we currently have, we expect that we will not have so much demand as product units we have available.
  If the customer is accepted, the counter q_2 has to be increased by one.

- **Condition 3** (lines 18-20)
  The customer generates a revenue of a (i.e. the customer is of Type-2). Further, if the counter for Type-2 customers q_2 is less than the above specified threshold, the customer can be accepted. Then, the counter q_2 has to be increased by one.

4.2.3 Competitive ratio

$ALG_{2,c}$ attains a competitive ratio of $c - O(\frac{1}{(1-c)^2 ap^{\frac{3}{2}}}\sqrt{\frac{n^2 \log n}{b^3}})$ for a certain range of $c, c \in [0,1]$ and $c \leq c^*$. For too large c values, i.e. $c \to 1$, the algorithm can not guarantee the competitive ratio.

Algorithm 2 Adaptive Algorithm $ALG_{2,c}$

```
1: Initialize:
       $q_1 \leftarrow 0$
       $q_2 \leftarrow 0$
       $\Phi \triangleq \frac{1-c}{1-a}$
       $\delta \triangleq \frac{\Phi b}{n}$
       $rem.inv = b$
2: for $\lambda = \frac{1}{n}$ to 1 do
3:     if $\lambda < \delta$ then
4:         $u_1(\lambda) \triangleq b$
5:         $u_{1,2}(\lambda) \triangleq b$
6:     else if $\lambda \geq \delta$ then
7:         $u_1(\lambda) \triangleq \min\{\frac{o_1(\lambda)}{\lambda p}, \frac{o_1(\lambda)+(1-\lambda)(1-p)n}{1-p+\lambda p}\}$
8:         $u_{1,2}(\lambda) \triangleq \min\{\frac{o_1(\lambda)+o_2(\lambda)}{\lambda p}, \frac{o_1(\lambda)+o_2(\lambda)+(1-\lambda)(1-p)n}{1-p+\lambda p}\}$
9:     end if
10:    $i = \lambda \cdot n$
11:    if $rem.inv > 0$ then
12:        if $v_i = 1$ then
13:            $q_1 \leftarrow q_1 + 1$
14:            $rem.inv \leftarrow rem.inv - 1$
15:        else if $v_i = a$ and $u_{1,2}(\lambda) < b$ then
16:            $q_2 \leftarrow q_2 + 1$
17:            $rem.inv \leftarrow rem.inv - 1$
18:        else if $v_i = a$ and $q_2 \leq \lfloor \Phi b + c(b - u_1(\lambda))^+ \rfloor$ then
19:            $q_2 \leftarrow q_2 + 1$
20:            $rem.inv \leftarrow rem.inv - 1$
21:        end if
22:    end if
23: end for
```

For convenience of reference, in Table 4.1 we summarize the defined notations.

b	(identical) units of the product to sell
n	Number of periods the product is sold ($n \geq b$)
p	fraction of randomly chosen customers which do not follow the prescribed adversarial order
λ	time steps of a time horizon normalized to 1; $\lambda = 1/n, 2/n, \ldots, 1$
$\overrightarrow{v_I}$	Initial customer sequence, determined by the adversary; $\overrightarrow{v_I} = (v_{I,1}, v_{I,2}, \ldots, v_{I,n})$ with $v_{I,j} \in \{0, a, 1\}$, for $1 \leq j \leq n$
$\overrightarrow{V}$	Random customer arrival sequence; $\overrightarrow{V} = (V_1, V_2, \ldots, V_n)$
$\overrightarrow{v}$	Realization of $\overrightarrow{V}$; $\overrightarrow{v} = (v_1, v_2, \ldots, v_n)$
n_j	Number of type-j customers in the entire sequence $\overrightarrow{v_I}, j = 1, 2$ (deter.)
S	subset of customers in the stochastic group
A	subset of customers in the adversarial group
$O_j(\lambda)$	Random number of type-j customers arriving up to time λ
$o_j(\lambda)$	Realization of $O_j(\lambda)$
$O_j^S(\lambda)$	Random number of type-j customers in S arriving up to time λ
$o_j^S(\lambda)$	Realization of $O_j^S(\lambda)$ (not observable by the algorithm)
$\eta_j(\lambda)$	Number of type-j customers among the first λn customers in $\overrightarrow{v_I}$ (deter.)
$\tilde{o}_j(\lambda)$	$(1-p)\eta_j(\lambda) + p\lambda n_j$ (deter. approximation of $O_j(\lambda)$)
$\tilde{o}_j{}^S(\lambda)$	$p\lambda n_j$ (deter. approximation of $O_j^S(\lambda)$)
θ	fixed threshold, used in ALG_1
$\epsilon(\lambda)$	evolving threshold, used in ALG_1
q_1	counter for the number of accepted Type-1 customers
$q_{2,e}$	counter for the number of accepted Type-2 customers by the evolving threshold ϵ
$q_{2,f}$	counter for the number of accepted Type-2 customers by the fixed threshold θ
$rem.inv$	counter for the remaining product units that can be allocated

Table 4.1: Notations

4.3 Comparison

In this section, we will look at the performance of the non-adaptive and adaptive algorithms and compare them. The analysis was conducted using a programming language Python and is presented by a markdown in Google Colab.

4.3.1 Methodology

Firstly, we introduce the function imitating the demand arrival model of Hwang et al. (2021) with arguments n, p and a. It generates the customer sequence of size n from the set of rewards $\{0, a, 1\}$. As soon as randomizing algorithms in Python generate numbers uniformly, the imitation of the adversarial sequences is restricted. One should also pay attention to the fact that every time one runs the code, a new sequence is created. After that, we implement two functions of the non-adaptive algorithm. The first one, `NonAdaptivePrint`, includes decision explanations at each step as well as the number of product units left and returns the reward. The second one, `NonAdaptive`, is without explanations and returns the name of the algorithm, the number of accepted type-1 and type-2 customers, the reward, number of product units left, computational time and number of periods n. The same two functions, `AdaptivePrint` and `Adaptive`, are introduced for the adaptive algorithm. The examples and outputs of these functions are presented in Appendix 6.3.

4.3.2 Airplane example

We run the algorithms for our example with selling airplane tickets. Let us imagine that there are $b = 300$ seats in the airplane and we have $n = 650$ periods for selling the tickets. Business class customers bring the reward 1 and economy-class customers pay $a = 0.5$. $p = 60\%$ of the demand is spread uniformly and can be predicted. The competitive ratio parameter for the adaptive algorithm is $c = 0.5$. The results are presented in the table:

As we see, the adaptive algorithm outperforms the non-adaptive one in rewards, because it accepts more type-2 customers and leaves no tickets unsold. However, it is almost three times slower than the adaptive algorithm.

Criteria	Non-adaptive	Adaptive
Customers type-1 accepted	200	199
Customers type-2 accepted	52	101
Reward	226.0	249.5
Units left	48	0
Computational time	0.00036	0.00105

Table 4.2: Airplane example: results

4.3.3 Simulation and performance comparison

To compare different parameters of the algorithms, we run a simulation. We begin with $n = 25$ and iteratively increase it by 25. The number of iterations is $k = 300$. To conduct a simulation, we randomly create a sequence of customers and run the algorithms for this sequence, after that we repeat it k times. Moreover, we introduce the parameter $d = b/n$. We run the simulation loop with different combinations of parameters a, d, p and c. Then the results are analyzed in terms of the reward, computational time, number of units left and numbers of accepted type-1 and type-2 customers.

As an example, the comparison results for the parameters $a = 0.5$, $d = 0.5$, $p = 0.5$ and $c = 0.5$ are presented in the graphs in Appendix 6.4. The comparison results for different high, low and medium parameters are presented in the table below. NA stands for the non-adaptive algorithm, A for the adaptive one, T1 and T2 stand for customers type-1 and type-2.

a	d	p	c	Larger reward	Less computational time	Units left	Customers T1 and T2
0.2	0.2	0.2	0.2	NA	NA	No units left	NA accepts more T1 and less T2
0.2	0.2	0.2	0.8	A	NA	No units left	A accepts more T1 and less T2
0.2	0.2	0.8	0.2	NA	NA	No units left	NA accepts more T1 and less T2
0.2	0.2	0.8	0.8	NA	NA	No units left	NA accepts more T1 and less T2
0.2	0.8	0.2	0.2	A	NA	Units left increase with n, NA leaves more units	T1 - equally, A accepts more T2
0,2	0,8	0,2	0,8	NA	NA	Units left increase with n, A leaves more units	T1 - equally, NA accepts more T2
0,2	0,8	0,8	0,2	A	NA	Units left increase with n, NA leaves more units	T1 - equally, A accepts more T2
0,2	0,8	0,8	0,8	A	NA	Units left increase with n, NA leaves more units	T1 - equally, A accepts more T2
0,8	0,2	0,2	0,2	NA	NA	No units left	NA accepts more T1 and less T2
0,8	0,2	0,2	0,8	Equally, NA a bit more with larger n	NA	No units left	NA accepts more T1 and less T2

Table 4.3: Performance comparison of both algorithms for different parameter combinations

a	d	p	c	Larger reward	Less computational time	Units left	Customers T1 and T2
0,8	0,2	0,8	0,2	NA	Equally, A a bit better	No units left	NA accepts more T1 and less T2
0,8	0,2	0,8	0,8	NA	NA	No units left	NA accepts more T1 and less T2
0,8	0,8	0,2	0,2	A	NA	Units left increase with n, NA leaves more units	T1 - equally, A accepts more T2
0,8	0,8	0,2	0,8	A	NA	Units left increase with n, NA leaves more units	T1 - equally, A accepts more T2
0,8	0,8	0,8	0,2	A	NA	Units left increase with n, NA leaves more units	T1 - equally, A accepts more T2
0,8	0,8	0,8	0,8	A	NA	Units left increase with n, NA leaves more units	T1 - equally, A accepts more T2
0,5	0,2	0,2	0,5	NA	NA	No units left	NA accepts more T1 and less T2
0,5	0,2	0,5	0,5	NA	NA	No units left	NA accepts more T1 and less T2
0,5	0,2	0,8	0,5	NA	NA	No units left	NA accepts more T1 and less T2
0,5	0,5	0,2	0,5	A	NA	Units left increase with n for NA, NA leaves more units	T1 - equally, NA a bit more and less T2

Table 4.4: Performance comparison of both algorithms for different parameter combinations

a	d	p	c	Larger reward	Less computational time	Units left	Customers T1 and T2
0,5	0,5	0,5	0,5	A	NA	Units left increase with n for NA, NA leaves more units	T1 - equally, NA a bit more and less T2
0,5	0,5	0,8	0,5	A	NA	Units left increase with n for NA, NA leaves more units	T1 - equally, NA a bit more and less T2
0,5	0,8	0,2	0,5	A	NA	Units left increase with n, NA leaves more units	T1 - equally, A accepts more T2
0,5	0,8	0,5	0,5	A	NA	Units left increase with n, NA leaves more units	T1 - equally, A accepts more T2
0,5	0,8	0,8	0,5	A	NA	Units left increase with n, NA leaves more units	T1 - equally, A accepts more T2

Table 4.5: Performance comparison of both algorithms for different parameter combinations

The comparison results lead to following observations:

- With larger d, the number of units left increases with n. The non-adaptive algorithm leaves more units, accepts less type-2 customers and, as a consequence, gains less reward (on average). Type-1 customers are accepted equally.
- However, with small d, the non-adaptive algorithm gains more reward as it accepts more type-1 customers and less type-2. In this case, none of the algorithms leaves unsold units.
- For medium d, the share of units left for the adaptive algorithm becomes smaller with increasing n. The adaptive algorithm gets more reward by accepting a bit less customers of type-1 and much more customers of type-2.
- The non-adaptive algorithm is much faster than the adaptive algorithm.
- The adaptive algorithm gains larger reward more often than the non-adaptive one.
- The non-adaptive algorithm tends to leave more units unsold, so it rejects too many type-2 customers.
- The adaptive algorithm gets larger reward for $a = 0.5$.
- With growing c, the adaptive algorithm rejects more type-2 customers.

5 Conclusion

In this paper, we describe a mixed stochastic and adversarial demand arrival model and two algorithms based on this model for an online resource allocation problem, proposed by Hwang et al. (2021). We model the non-adaptive and adaptive algorithms using Python and calculate different numerical examples (manually and with coding). Moreover, we run a simulation to compare the performance of the algorithms. Our results show that the algorithms perform differently in different parameter constellations. However, on average, the adaptive algorithm gains more often more reward, whereas the non-adaptive algorithm appears to be much faster to compute. Future research could include the generalization of the algorithms for more than one type of customers or resources or the application of the algorithms to other online problems.

References

Agrawal, S., Wang, Z., and Ye, Y. (2014). A dynamic near-optimal algorithm for online linear programming. *Operations Research*, 62(4):876–890.

Andrew, L. L. H., Barman, S., Ligett, K., Lin, M., Meyerson, A., Roytman, A., and Wierman, A. (2013). A tale of two metrics: Simultaneous bounds on competitiveness and regret. In *Proceedings of the 26th Annual Conference on Learning Theory*, pages 741–763.

Ball, M. and Queyranne, M. (2009). Toward robust revenue management: Competitive analysis of online booking. *Operations Research*, 57(4):950–963.

Chakrabarti, D., Kumar, R., Radlinski, F., and Upfal, E. (2008). Mortal multi-armed bandits. In *Proceedings of the 21st International Conference on Neural Information Processing Systems*, pages 273–280.

Chen, L., Luo, H., and Wei, C.-Y. (2021). Minimax regret for stochastic shortest path with adversarial costs and known transition. In *Proceedings of Machine Learning Research*, volume 134, pages 1–36.

Cutkosky, A. and Boahen, K. (2017). Stochastic and adversarial online learning without hyperparameters. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 5066–5074.

Devanur, N. R., Jain, K., Sivan, B., and Wilkens, C. A. (2019). Near optimal online algorithms and fast approximation algorithms for resource allocation problems. *Journal of the ACM*, 66(1):1–41.

Esfandiari, H., Korula, N., and Mirrokni, V. (2015). Online allocation with traffic spikes: Mixing adversarial and stochastic models. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation*, pages 169–186.

Gupta, A., Koren, T., and Talwar, K. (2019). Better algorithms for stochastic bandits with adversarial corruptions. In *Proceedings of the 32nd Conference on Learning Theory*, volume 99, pages 1562–1578.

Hwang, D., Jaillet, P., and Manshadi, V. (2021). Online resource allocation under partially predictable demand. *Operations Research*, 69(3):895–915.

Lykouris, T., Mirrokni, V., and Leme, R. P. (2018). Stochastic bandits robust to adversarial corruptions. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 114–122.

Lykouris, T., Mirrokni, V., and Leme, R. P. (2020). Bandits with adversarial scaling. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 6511–6521.

Mehta, A., Saberi, A., Vazirani, U., and Vazirani, V. (2005). Adwords and generalized on-line matching. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, pages 264–273.

Neu, G., György, A., and Szepesvári, C. (2012). The adversarial stochastic shortest path problem with unknown transition probabilities. In *Proceedings of the 15th International Conference on Artificial Intelligence and Statistics*, volume 22, pages 805–813.

Shamir, O. and Szlak, L. (2017). Online learning with local permutations and delayed feedback. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 3086–3094.

6 Appendices

6.1 Analysis of the formula for the fixed threshold θ

As mentioned in Section 4.1, the fixed threshold θ is calculated with the following formula:

$$\theta \triangleq \frac{1-p}{2-a}$$

To gain a better understanding of how the formula works, it is examined below for different parameter values. At first, the two extreme cases where $p = 0$ and $p = 1$ are analyzed. Afterwards we examine a regular case where $p = 0.5$.

- $\boldsymbol{p = 0}$: We obtain a pure adversarial model. The equation becomes the following:

$$\theta = \frac{1}{2-a}$$

  - $\boldsymbol{a \to 1}$: With increasing a, the Type-2 customers become more valuable. If a goes to 1, they become almost as valuable as Type-1 customers.

  $$\theta = \frac{1}{2-1} = 1$$

  According to lines 12 to 13 of ALG_1, a Type-2 customer gets accepted if $q_{2,f} < \lfloor \theta b \rfloor$. If $\theta \to 1$, Type-2 customers will always get accepted under the condition, that there is still inventory remaining.

  - $\boldsymbol{a \to 0}$: If a goes to 0, a Type-2 customer is almost worthless.

  $$\theta = \frac{1}{2-0} = 0.5$$

  In this case, a Type-2 customer is only accepted, if the number of already accepted Type-2 customers is $\lfloor 0.5b \rfloor$. This also means, in total a maximum of 50% of the available resources can be given to Type-2 customers.

- **$a \rightarrow 0.5$** : If a goes to 0.5 we obtain the following value for θ:

  $$\theta = \frac{1}{2 - 0.5} = 2/3$$

  In this case, similar to above, a Type-2 customer is only accepted, if the number of already accepted Type-2 customers is $\lfloor 2/3b \rfloor$. This also means, in total a maximum of 66.6% of the available resources can be given to Type-2 customers.

- **$p = 1$**: If p equals 1, we obtain a θ of 0. This is consistent with the definition of θ, because θ was designed for the other $(1 - p)$ customers, which join the adversarial group A. For $p = 1$ all customers would join the stochastic group S. This group is already covered by the evolving threshold ϵ.

- **$p = 0.5$** :
  - **$a \rightarrow 1$** :If a goes to 1, we obtain the following value for θ.

    $$\theta = \frac{1 - 0.5}{2 - 1} = 0.5$$

    In this case, the Type-2 customers are again almost as valuable as the Type-2 customers. Just like the case from above where $p = 0, a \rightarrow 1$ we would accept every Type-2 customer. But because there are only 50% of customers joining the adversarial group, we can just accept a maximum of 50% via the fixed threshold θ. The other 50% of customers are already covered by the evolving threshold ϵ.

  - **$a \rightarrow 0$**: If a goes to 0, we obtain the following value for θ.

    $$\theta = \frac{1 - 0.5}{2 - 0} = 0.25$$

    In this case, a Type-2 customer is only accepted, if the number of already accepted Type-2 customers is $\lfloor 0.25b \rfloor$. This also means, in total a maximum of 25% of the available resources can be given to Type-2 customers.

  - **$a \rightarrow 0.5$** :If a goes to 0.5 we obtain the following value for θ:

    $$\theta = \frac{1 - 0.5}{2 - 0.5} = 1/3$$

    In this case, similar to above, a Type-2 customer is only accepted, if the number of already accepted Type-2 customers is $\lfloor 1/3b \rfloor$. This also means, in total a maximum of 33.3% of the available resources can be given to Type-2 customers.

6.2 Analysis of the threshold formula used in $ALG_{2,c}$

As mentioned in Section 4.2.1, the threshold used in $ALG_{2,c}$ is defined like that:

$$\lfloor \Phi b + c(b - u_1(\lambda))^+ \rfloor$$

In the following chapter, we present the idea behind using this formula.

6.2.1 Derivation of the formula

Imagine, a Type-2 customer arrives at time λ and we face the decision to reject or accept the customer. The upper bound $u_{1,2}(\lambda)$ for the number of Type-1 and Type-2 customers is calculated. No we can differentiate between 2 cases:

- **Case 1:** $u_{1,2}(\lambda) < b$ In this case, the upper bound for the total Type-1 and Type-2 customers in the sequence is less than the actual inventory. Every customer can be accepted, regardless of the customer type. because we won't exceed the inventory.
- **Case 2:** $u_{1,2}(\lambda) \geq b$ In this case, the upper bound for the total Type-1 and Type-2 customers in the sequence is more than the actual inventory. Therefore, not every customer can be accepted. For this case, we define the following threshold.

With $ALG_{2,c}$ we want to achieve a competitive ratio of at least c.
The competitive ratio is defined by

$$c \leq \frac{\mathbb{E}[ALG(\overrightarrow{v})]}{OPT(\overrightarrow{v})}$$

Optimal offline solution OPT

For Case 2 we now calculate the optimal offline solution $\boldsymbol{OPT(\overrightarrow{v})}$:

$$\begin{aligned} OPT(\overrightarrow{v}) &= min\{n_1, b\} + a \cdot max\{b - n_1\}^+ \\ &= min\{n_1, b\} + a \cdot max\{b - n_1, 0\} \end{aligned}$$

Now we distinct between the cases $b < n_1$ and $b > n_1$.

- $\boldsymbol{b < n_1}$ **:** In this case, we have more Type-1 customers than actual inventory. The first part of the equation becomes b and the term $(b - n_1)$ becomes negative.Therefore, the second part of the equation is omitted. The optimal reward for the offline solution is only b, since we give b units to type-1 customers who each generate 1 in sales.
$$OPT(\overrightarrow{v}) \quad = b$$
- $\boldsymbol{b > n_1}$ **:** When b is greater than the actual number of Type-1 customers, we should allocate the remaining units to Type-2 customers. The first part of the equation becomes n_1, the term becomes $(b - n_1)$, because it becomes positive.
$$\begin{aligned} OPT(\overrightarrow{v}) &= n_1 + a \cdot (b - n_1) \\ &= n_1 + ab - n_1 a \\ &= n_1(1 - a) + ab \end{aligned}$$
  The optimal reward for the offline solution is the number of Type-1 customers (multiplied by 1, since Type-1 customers generate a revenue of 1). In addition, the remainder of the available product units (total inventory minus the number of Type-1 customers) is allocated to Type-2 customers and multiplied by the reward for Type-2 customers a. This is possible because we only consider the case where $u_{1,2}(\lambda) \geq b$. This will definitely give us enough customers to allocate the whole inventory to customers. Since we are looking at the optimal offline solution, possible assignment errors due to lack of information are also excluded.

By combining these two cases, we obtain the following formula:

$$OPT(\overrightarrow{v}) = (1 - a)min\{n_1, b\} + ab$$

If we now assume, that $u_1(\lambda)$ is indeed an upper bound for n_1, meaning that $u_1(\lambda) \geq n_1$, we obtain:

$$OPT(\overrightarrow{v}) \leq (1 - a)min\{u_1(\lambda), b\} + ab$$

Solution obtained by the online algorithm $ALG_{2,c}$

Now we analyze the online setting, where we don't have all information. If we accept the current Type-2 customer $\lambda \cdot i$, the maximal revenue we can get is:

$$(b - (q_2(\lambda - \frac{1}{n}) + 1)) + a(q_2(\lambda - \frac{1}{n}) + 1$$

The first part of the equation subtracts one unit from the complete revenue generated by type-1 customers, by subtracting the number of already counted Type-2 customers at time $\lambda\frac{1}{n}$ (one time step before) plus one from the total number of product units. The second part of the equation adds one Type-2 customer revenue a to the equation.

Now we combine the two observations.
To obtain a competitive ration of at least c, we accept a Type-2 customer only if:

$$\frac{(b - (q_2(\lambda - \frac{1}{n}) + 1)) + a(q_2(\lambda - \frac{1}{n}) + 1}{(1-a) \cdot min\{u_1(\lambda), b\} + ab} \geq c$$

After rearranging the formula we obtain:

$$q_2(\lambda - \frac{1}{n}) + 1 \leq \lfloor \frac{1-c}{1-a} \cdot b + c(b - u_1(\lambda))^+ \rfloor$$

The Gauss brackets are introduced because the right side of the equation may not be an integer. After replacing the expression $\frac{1-c}{1-a}$ with Θ we obtain the equation for the threshold, which is also used in the algorithm:

$$q_2(\lambda - \frac{1}{n}) + 1 \leq \lfloor \Theta \cdot b + c(b - u_1(\lambda))^+ \rfloor$$

6.3 Examples of code outputs for the non-adaptive and adaptive algorithm functions

6.3.1 Non-adaptive algorithm

Listing 6.1: Non-adaptive algorithm with a print function

```
print('Reward: ', NonAdaptivePrint(n = 25, b = 10, a = 0.8, p = 0.6))
```

```
[1.0, 1.0, 0.8, 1.0, 0.8, 1.0, 0.8, 0.0, 1.0, 0.0, 1.0, 0.8, 0.8, 0.0, 1.0,
1.0, 1.0, 0.8, 1.0, 0.8, 0.0, 0.0, 0.0, 0.0, 1.0]
1 / 25
Customer 1 accepted
b = 9
2 / 25
Customer 1 accepted
b = 8
3 / 25
Customer 2f accepted
b = 7
4 / 25
Customer 1 accepted
b = 6
5 / 25
Customer 2f accepted
b = 5
6 / 25
Customer 1 accepted
b = 4
7 / 25
Customer skipped
b = 4
8 / 25
Customer skipped
b = 4
```

```
9 / 25
Customer 1 accepted
b = 3
10 / 25
Customer skipped
b = 3
11 / 25
Customer 1 accepted
b = 2
12 / 25
Customer skipped
b = 2
13 / 25
Customer skipped
b = 2
14 / 25
Customer skipped
b = 2
15 / 25
Customer 1 accepted
b = 1
16 / 25
Customer 1 accepted
b = 0
Reward:  9.6
```

Listing 6.2: Non-adaptive algorithm without a print function

```
#return name, customers type-1 accepted, customers type-2 accepted,
#reward, units left , time, number of periods

V1 = DemandArrival(n = 25, a = 0.5, p = 0.6)
NonAdaptive(n = 25, b = 10, a = 0.5, p = 0.6, V = V1)
```

```
('non-adaptive', 8, 2, 9.0, 0, 1.6689300537109375e-05, 25)
```

6.3.2 Adaptive algorithm

Listing 6.3: Adaptive algorithm with a print function

```
print('Reward: ', AdaptivePrint(c = 0.6, a = 0.4, b = 10, n = 25,
p = 0.6))
```

```
[0.4, 0.4, 1.0, 0.4, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.4, 0.4, 0.0, 1.0, 0.0,
1.0, 1.0, 0.4, 1.0, 0.0, 1.0, 0.0, 0.4, 1.0, 1.0]
1 / 25
Customer 2.2 accepted
b = 9
2 / 25
Customer 2.2 accepted
b = 8
3 / 25
Customer 1 accepted
b = 7
4 / 25
Customer 2.2 accepted
b = 6
5 / 25
Customer 1 accepted
b = 5
6 / 25
Customer 1 accepted
b = 4
7 / 25
Customer 1 accepted
b = 3
8 / 25
Customer skipped
9 / 25
Customer 1 accepted
b = 2
10 / 25
```

```
Customer 1 accepted
b = 1
11 / 25
Customer skipped
12 / 25
Customer skipped
13 / 25
Customer skipped
14 / 25
Customer 1 accepted
b = 0
Reward:  8.2
```

Listing 6.4: Adaptive algorithm without a print function

```
#return name, customers type-1 accepted, customers type-2 accepted,
#reward, units left, time, number of periods

V2 = DemandArrival(n = 25, p = 0.6, a = 0.5)
Adaptive(c = 0.6, a = 0.5, b = 10, n = 25, p = 0.6, V = V2)

('adaptive', 6, 4, 8.0, 0, 2.6941299438476562e-05, 25)
```

6.4 Comparison of graphs

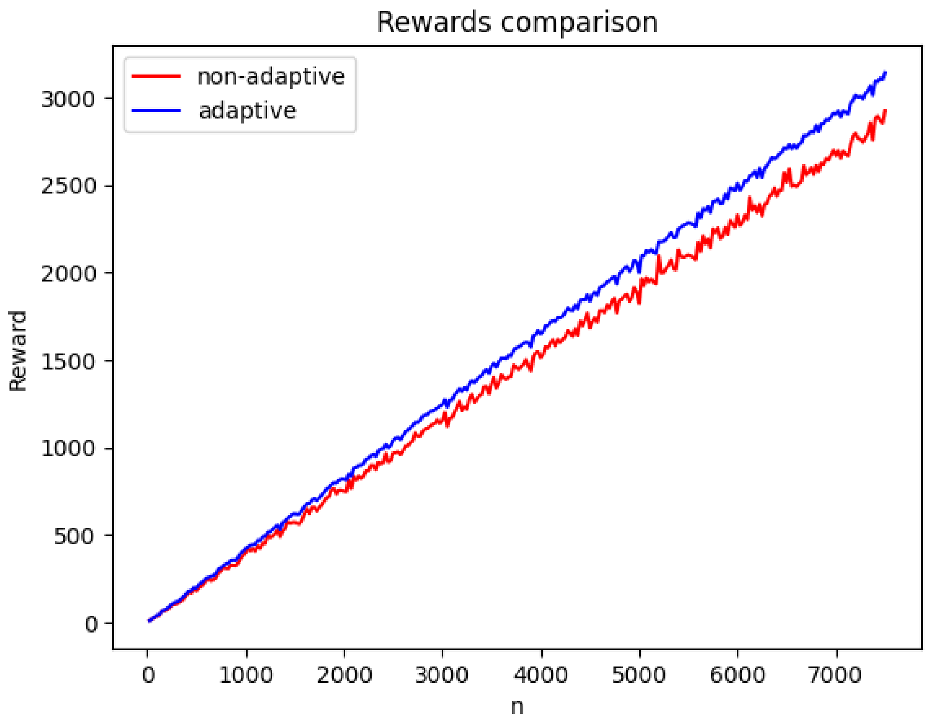

Figure 6.1: Rewards comparison

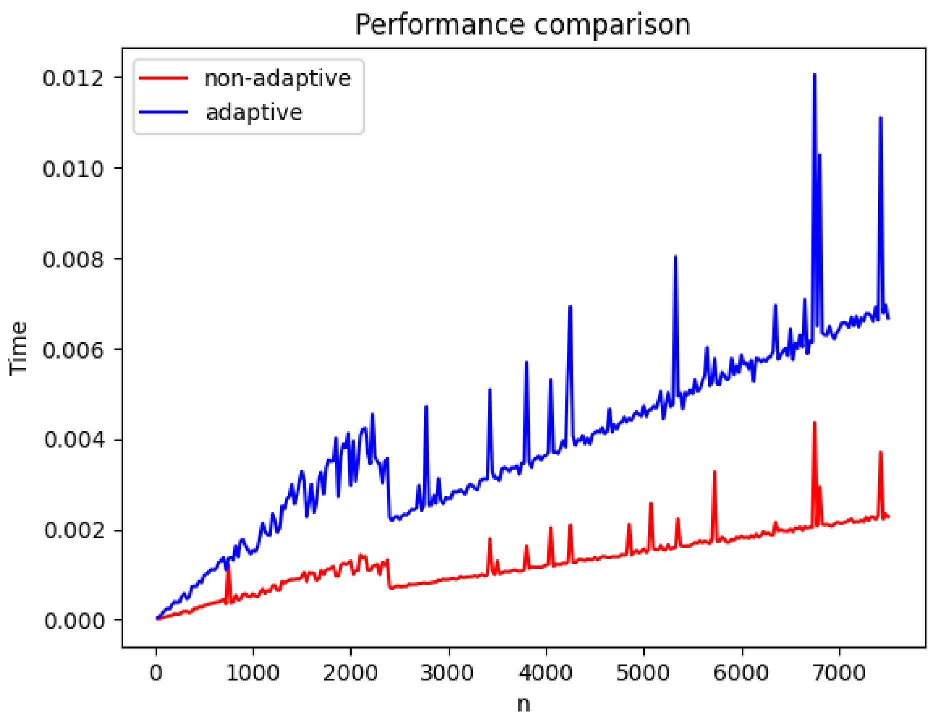

Figure 6.2: Computational time comparison

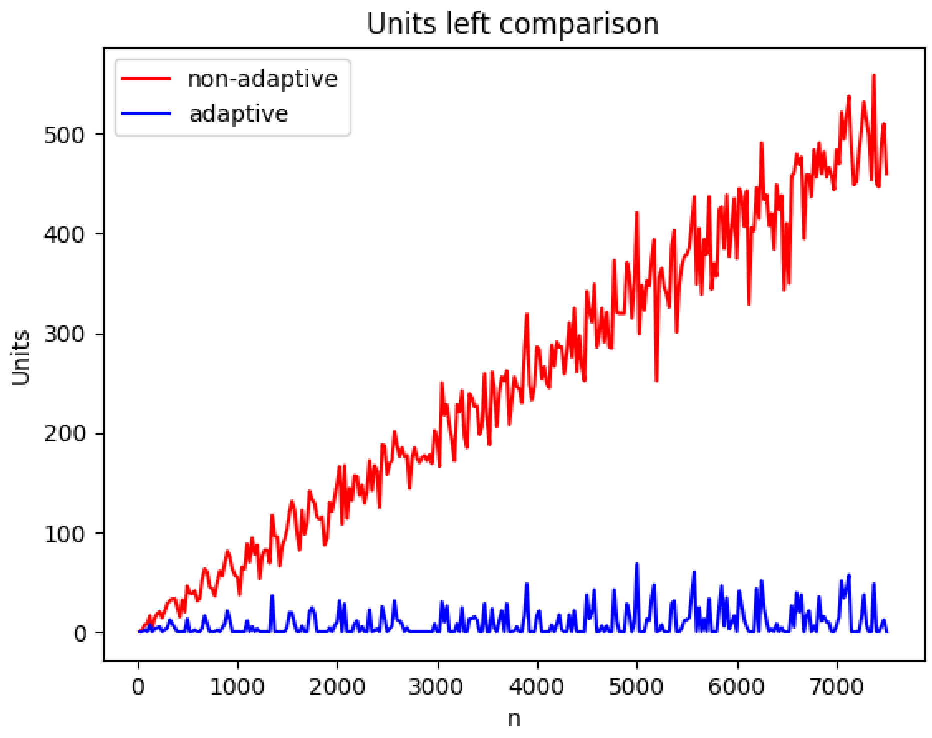

Figure 6.3: Units left comparison

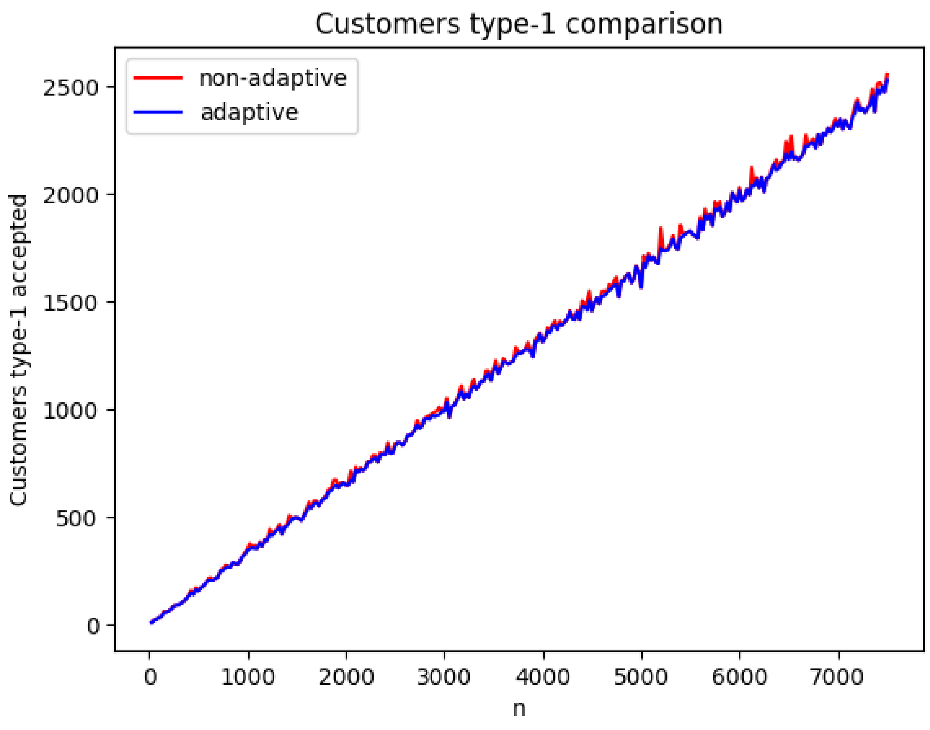

Figure 6.4: Accepted customers of Type-1 comparison

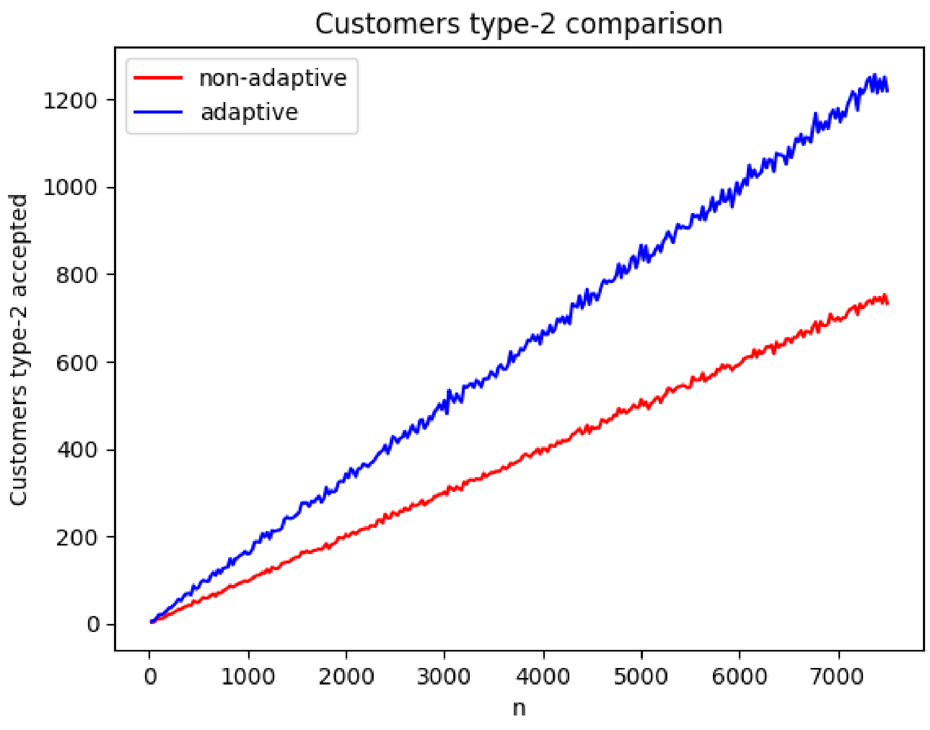

Figure 6.5: Accepted customers of Type-2 comparison