Queue Mining for Delay Prediction in Multi-Class Service Processes

Arik Senderovich[b,*], Matthias Weidlich[1], Avigdor Gal[b], Avishai Mandelbaum[b]

[a]*Technion - Israel Institute of Technology, Haifa, Israel*
[b]*Imperial College London, London, United Kingdom*

Abstract

Information systems have been widely adopted to support service processes in various domains, *e.g.*, in the telecommunication, finance, and health sectors. Information recorded by systems during the operation of these processes provide an angle for operational process analysis, commonly referred to as process mining. In this work, we establish a queueing perspective in process mining to address the online delay prediction problem, which refers to the time that the execution of an activity for a running instance of a service process is delayed due to queueing effects. We present predictors that treat queues as first-class citizens and either enhance existing regression-based techniques for process mining or are directly grounded in queueing theory. In particular, our predictors target multi-class service processes, in which requests are classified by a type that influences their processing. Further, we introduce queue mining techniques that derive the predictors from event logs recorded by an information system during process execution. Our evaluation based on large real-world datasets, from the telecommunications and financial sectors, shows that our techniques yield accurate online predictions of case delay and drastically improve over predictors neglecting the queueing perspective.

Keywords: Delay Prediction, Process Mining, Queueing Theory, Queue Mining

1. Introduction

The conduct of service processes, *e.g.*, in the telecommunication and health sectors, is heavily supported by information systems. To manage such processes and improve the operation of supporting systems, event logs recorded during process operation constitute a valuable source of information. Recently, this opportunity was widely exploited in the rapidly growing research field of process mining. It started with mining techniques that focused mainly on the control-flow perspective, namely extracting control-flow models from event logs [1] for qualitative analyses, such as model-based verification [2].

In recent years, research in process mining has shifted the spotlight from qualitative analysis to (quantitative) *online* operational support ([3], Ch. 9). To provide operational support, the control-flow perspective alone does not suffice and, therefore, new perspectives are mined. For example, the time perspective exploits event timestamps and frequencies to locate bottlenecks and predict execution times.

To date, operational process mining is largely limited to black-box analysis. That is, observations obtained for single instances (cases) of a process are aggregated to derive predictors for the behaviour of cases in the future. This approach can be seen as a regression analysis over individual cases, assuming that they are executed largely independently of each other. In many processes, however, cases do not run in isolation but *multiple cases* compete over scarce resources. Only some cases get served at a certain point in time (complete execution of an activity and progress in process execution) while others must wait for resources to become available. Cases that did not get served are enqueued and consequently delayed.

A specific operational problem that is affected by the competition of multiple cases for scarce resources is *online delay prediction*. This problem refers to the time that the execution of an activity for a particular case is delayed due to queueing effects. In a simple *single-class* setting, cases are of a uniform type and form a single queue that causes delays. However, in this work, we also address the more complex *multi-class* setting. Here, cases are enqueued depending on their type, which calls for online delay prediction per case type.

Against this background, we argue that there is a need to consider the queueing perspective in operational process mining in general, and for the online delay prediction problem in particular. To address this need, we outline various methods to integrate queueing information in delay prediction, jointly referred to as *queue mining*. These techniques are grounded in a model for service logs that captures queueing related events during the execution of a service process. The techniques for queue mining introduced in this paper are described along two dimensions:

Foundation: regression-based vs. queueing models. A first set of our predictors takes traditional approaches to operational process mining as a starting point and extends an existing regression-based technique for time prediction [4] to consider queues and system load. A second set of predictors originates from queueing theory [5, 6] and leverages properties of a queueing model, potentially under a widely known congestion law (the snapshot principle).

*Corresponding author
Email addresses: sariks@tx.technion.ac.il (Arik Senderovich), m.weidlich@imperial.ac.uk (Matthias Weidlich), avigal@ie.technion.ac.il (Avigdor Gal), avim@ie.technion.ac.il (Avishai Mandelbaum)

Case types: single-class vs. multi-class. Delay prediction is sensitive to classifications of cases. In a single-class setting, all cases are of a uniform type and delay prediction relates to a single queue of cases. In a multi-class setting, in turn, cases are classified by a certain type that influences how cases are enqueued. Hence, delay prediction relates to a specific case type.

In addition to queue mining techniques that consider the outlined spectrum in terms of foundations and case types, our contribution is a comprehensive empirical evaluation of the presented techniques. We employ two large real-world datasets, one from telecommunications and one from the financial sector, and illustrate that our techniques yield accurate online predictions of case delay. In particular, our predictors drastically improve over those that neglect the queueing perspective and simple heuristics based on queueing models achieve comparable performance to complex machine learning techniques.

This paper is an extended and revised version of our earlier work [7] that focused on single-class settings only. In this work, we also consider the more complex setup of multi-class settings. To make the regression-based approach work for the multi-class setting, we also propose an extension to a prediction method based on transition systems developed by van der Aalst et al. [4] and enhanced with context factors by Folino et al. [16]. In particular, we show how a combination of characteristics of cases and the overall system state can be incorporated following a machine learning approach. Further, we extended the evaluation with experiments related to the multi-class setting.

The remainder of this paper is organized as follows. The next section provides motivation for the queueing perspective and background on queueing models and also introduces the delay prediction problem. Section 3 defines the service log as the basis to our queue mining methods. Section 4 adapts an existing method for regression-based time prediction to incorporate feature-annotations and machine learning. Then, Section 5 focuses on the single-class setting and introduces delay predictors along with mining methods for these predictors. Section 6 presents predictors and queue mining methods for the multi-class setting. Section 7 presents our experiments and discusses their results. We review related work in Section 8 and conclude in Section 9.

2. Background and Problem Specification

An example service process. For illustration, consider a service process operated by a bank's call center. Figure 1(a) depicts a BPMN [8] model of such a process, which focuses on the control-flow of a case, i.e., a single customer. The customer dials in and is then connected to a voice response unit (VRU). The customer either completes service within the VRU or chooses to opt-out, continuing to an agent. Once customers have been served by an agent, they either hang-up or, in rare cases, choose to continue for another service (VRU or forwarding to an agent).

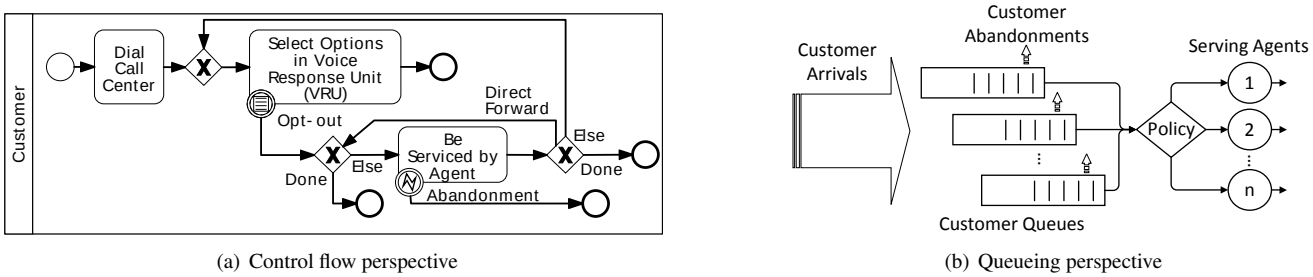
Although this model provides a reasonable abstraction of the process from the perspective of a single customer, it falls short of capturing important operational details. Customers that seek a service are served by one of the available agents or wait in a queue. Hence, activity 'Be Serviced by Agent' comprises a waiting phase and an actual service phase. Customers that wait for service may also abandon the queue due to impatience. To provide operational analysis for this service process and predict delay of processing, such queues and abandonments must be taken into account explicitly.

The queueing perspective. For the above example, only activity 'Be Serviced by Agent' involves significant queueing since the other activities do not rely on scarce resources of the service provider. Adopting a queueing perspective for this activity, Figure 1(b) outlines how the activity is conducted under multiple cases arriving at the system and, thus, emphasizes that execution time of one case depend on cases that are already in the system. In fact, Figure 1(b) presents a single-station queueing system, where customers are classified according to some case properties (e.g., whether they have a premium service contract) and the respective queues are served by n homogeneous agents.

Such a queueing system is described by a series of characteristics, which, for the single-class setting, is denoted using Kendall's notation as $\mathcal{A}/\mathcal{B}/\mathcal{C}/\mathcal{Y}/\mathcal{Z}$ [9]. The arrival process ($\mathcal{A}$) is defined by the joint distribution of the inter-arrival times. No assumption regarding the arrival process is expressed by replacing $\mathcal{A}$ with G for general distribution. The processing duration of a single case ($\mathcal{B}$) is described by the distribution of service time. The total number of agents at the queueing station is denoted by $\mathcal{C}$, which stands for capacity. When a case arrives and all agents are busy, the new arrival is queued. The maximum size of the system, $\mathcal{Y}$, can be finite, so that new customers are blocked if the number of running cases is larger than $\mathcal{Y}$. In call centers, which provide our present motivation, $\mathcal{Y}$ is practically infinite and can be omitted. Once an agent becomes available and the queue is not empty, a customer is selected according to a routing policy $\mathcal{Z}$. The most common policy is FCFS (First-Come-First-Served) and in such cases $\mathcal{Z}$ is also omitted.

For a multi-class system as the one depicted in Figure 1(b), factors that depend on the customer class are parametrized. As such, one may define the arrival processes ($\mathcal{A}_i$) or service times ($\mathcal{B}_i$) per customer class i. In the presence of multiple classes, the routing policy may implement a complex protocol to govern how different queues are served by agents. However, a common policy, also exploited in this work, is the one of *priority queues*. That is, there is a total order of customer classes in terms of their service priority. In such a setting, the policy for handling customers within the same class is typically FCFS.

Queueing models may include information on the distribution of customer (im)patience ($\mathcal{G}$), added following a '$+$' sign at the end of Kendall's notation. For mathematical tractability and sometimes backed up by practice, it is often assumed that sequences of inter-arrival times, service times and customer (im)patience are independent of each other, and each consists of independent elements that are exponentially distributed. Then, $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{G}$ are replaced by Ms, which stands for Markovian. For example, a $G/M/n+M$ model assumes that arrivals come from a general distribution, service times are exponential, agent capacity is of size n, queue size is infinite, routing policy is FCFS and (im)patience is exponentially distributed. This model is directly lifted to the multi-class setting.



(a) Control flow perspective

(b) Queueing perspective

Figure 1: Example process in a call center

The delay prediction problem. The phenomena of delay has been a popular research subject in queueing theory, see [12]. The interest in delay prediction is motivated by psychological insights on the negative effect of waiting on customer satisfaction [13]. Field studies have found that seeing the line ahead moving and getting initial information on the expected delay, both have a positive effect on the waiting experience [14, 15]. Thus, announcing the expected delay in an online fashion improves customer satisfaction.

We refer to the customer, whose delay time we wish to predict as the *target-customer*. In a multi-class setting, the target-customer always belongs to one of the customer classes in the system. Further, the target-customer is assumed to have infinite patience, i.e., the target customer will wait patiently for service, without abandoning, otherwise our prediction becomes useless. However, the influence of abandonments of other customers on the delay time of the target-customer is taken into account.

Formally, the *online delay prediction* problem can be stated as follows. Let W be a random variable that measures the delay time of a target-customer. Denote by ψ the predictor for W. Then, the *online delay prediction* problem aims at identifying an accurate ψ, with respect to the root mean-squared error (RMSE), i.e. $\mathbb{E}[(W-\psi)^2]$ with $\mathbb{E}$ denoting the expectation over random variables. As an example, consider a simple predictor ψ that is defined as the average of all past delays, denoted $y_1, ..., y_n$. That is, $\psi = \frac{1}{n}\sum_{i=1}^{n} y_i = \bar{y}$. Then, in the absence of knowledge about the actual mean, the RMSE can be approximated based on the observed data. That is, the sampled prediction error $\mathbb{E}[(\widehat{W-\psi})^2]$, which is the average of the squared differences of the observed delays from the average of past delays, quantifies the actual RMSE:

$$\mathbb{E}[(\widehat{W-\psi})^2] = \frac{1}{N}\sum_{i=1}^{N}(\bar{y}-y_i)^2. \quad (1)$$

3. Service Log

Below, we first define the essential concepts of service logs. Then, we formulate the problem of *online delay prediction*, which will be solved using queue mining in this work.

3.1. The Service Log

Service events (or events for short) and service paths (or paths) are the essential components of a service log. Inspired by [10], we formally capture these concepts as follows:

Definition 1 (Service event, Service path). *Let $\mathcal{S}$ denote the set of all possible service events, i.e. unique event identifiers. Let $\mathcal{S}^*$ be the set of all finite sequences over $\mathcal{S}$. We define $\Pi \subseteq \mathcal{S}^*$ as the set of all* feasible *service paths, i.e. finite sequences of service events. We require that each service event appears at most once in some path.*

According to this definition, a path $p \in \Pi$ is a finite sequence $(p_1, \ldots, p_n)$ of length n of events, such that $p_i \in \mathcal{S}, i = 1, ..., n$. It is worth noting that, in the literature, paths are often referred to as *cases*. Service events are associated with attributes, e.g., *timestamps*, *service activities*, *service locations*, and *resources*. We model such an attribute as a function that assigns an attribute value to a service event. A set of such attribute functions, in turn, defines the schema (aka structure) of a set of service events.

Definition 2 (Attribute function, Event schema). *Let A be the domain of an event attribute. Then, the attribute function $\alpha : \mathcal{S} \to A$ assigns values of this domain to service events. A finite set $\{\alpha_1, \ldots, \alpha_n\}$ of attribute functions is called an event schema.*

For example, α may be the attribute function for service activities, i.e., the domain of the function is defined as {service start, service end}. A specific service event $s \in \mathcal{S}$ then either indicates the start ($\alpha(s) =$ service start) or end of service ($\alpha(s) =$ service end). Using the introduced concepts, we define the general notion of an S-Log.

Definition 3 (S-Log). *A service log (S-Log) is a tuple $(S, P, \alpha_{\mathcal{S}})$ where,*

- *$S \subseteq \mathcal{S}$ is the set of* observed *service events.*
- *$P \subseteq \Pi$ is the set of* observed *service paths.*
- *$\alpha_{\mathcal{S}}$ is the event schema.*

The notion of a service log generalizes the *functional* definition of an event log as presented in [11]. Below, we define the (single-station) S-Log that corresponds to event data from single-station queues.

Definition 4 (Single-Station S-Log). *The single-station service log is the tuple $(S, P, \alpha_{\mathcal{S}})$ over $\mathcal{S}, \Pi, \mathcal{A}_{\mathcal{S}}$, where*

- *$\alpha_{\mathcal{S}} = \{\tau, \epsilon, \xi)\}$ is the event schema.*
- *$\tau : \mathcal{S} \to \mathbb{N}^+$ are timestamps.*
- *$\epsilon : \mathcal{S} \to E = \{qArrive, qAbandon, sStart, sEnd\}$ are service transitions.*
- *$\xi : \mathcal{S} \to C$ are customer classes.*

4. The Model of Feature-Annotated Transition Systems

In this section, we take up existing work on regression-based time prediction that exploits annotated transition systems and provide an extension that allows for flexible integration of queueing information. We first motivate this extension and give an overview of its main steps in Section 4.1, before we turn to the details of the method in Sections 4.2 to 4.4.

4.1. Motivation and Overview

Regression-based time prediction can be approached based on annotated transition systems, as introduced by van der Aalst et al. [4]. These transition systems are directly constructed by applying abstractions to the cases recorded during process execution. In a second step, the abstract states of the transition system are annotated with performance information. Although the transition system method was applied in [4] to predict remaining times of running cases, it is a general technique that can be applied to other supervised learning problems. While the state abstractions employed by this method allow for direct integration of case specific properties, the integration of context factors, e.g., on the load in a service system and queueing information, is challenging.

To encompass context factors into the state abstraction, Folino et al. [16] extended the work on performance prediction based on transition systems and defined a state abstraction that comprised of two types of features: (1) 'internal' case properties and (2) 'external' factors that characterize system state, e.g. workload, resource availability. However, this approach has several disadvantages, such as the need to cluster over the two feature types and the targeted outcome (e.g. delays, remaining times) and the limitation to decision trees as learning techniques (we discuss these limitations in more detail when reviewing related work). Therefore, in this paper, we undertake a more flexible approach that enables the use of various types of learning techniques and combines transition systems that comprise of cases and case-specific attributes (similarly to the internal features of Folino et al. [16]) with continuous vectors of system-state factors. This results in a decoupling of the state (which remains simple and case-related) from the complex (and possibly continuous) feature vectors when applying the learning technique.

The outline of our approach based on feature-annotated transition systems is presented in Figure 2. We shall now briefly describe the proposed method. A (service) process produces events that are stored in event logs and provide with the set of all possible (labeled) events. These event-labels are the input to the first step, which is the construction of the transition system (TS). Then, the service log, along with the newly constructed transition system feed the feature selection step. In this step, relevant features, such as the queue-length, are attached to the different states of the transition system.

The resulting featured transition system (FTS) then goes into the third step of transition system annotation. Here, past values of the outcome that we wish to predict (e.g. past delays) are attached to states and features of the FTS. Lastly, the annotated transition system (AFTS) goes into the learning phase, where a prediction algorithm is applied to explain the outcome, as

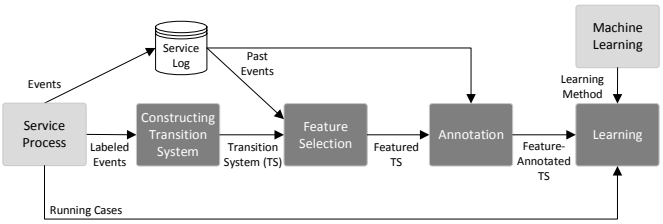


Figure 2: The featured transition system approach

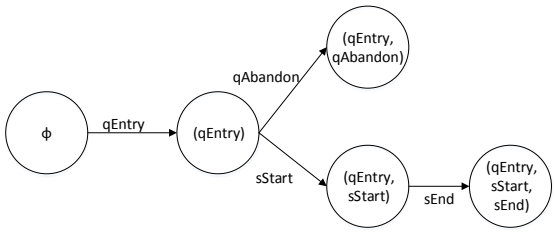


Figure 3: Single-station queue transition system

function of the states and their features. Below, we formally define each state of our approach and demonstrate the three steps using a running case of a single-station queueing system.

4.2. Step 1: Constructing a Transition System

A *transition system* is a triplet (Σ, E, T) where Σ is the set of states, E is the event-labels that stem from the service process and $T \subseteq \Sigma \times E \times \Sigma$ is the transition flow relation that describes state changes.

In the single-station service processes, the set of event-labels, E, was defined in Section 3, i.e. $E = \{$qEntry, qAbandon, sStart, sEnd$\}$. Moreover, in this case the event-label set uniquely defines the states. We referred to the set of all feasible service paths in the station as $\Pi \subseteq \mathcal{S}^*$. In order to define Σ, we first define a function π that maps service paths into a state abstraction (we shall refer to π as the abstraction function). For our running case, we choose π such that it transforms E into *labeled* traces of events, $\pi : \Pi \rightarrow E^*$. For example, a result of operating function π on a service path $p \in \Pi$ that started with queue and ended after service completion is $\pi(p) = (qEntry, sStart, sEnd)$. Other abstractions, such as the most recent event (memoryless), are also possible.

We are now ready to define the states of the transition system. The set of states Σ consists of labeled traces that result from operating π on Π, i.e. $\Sigma = Image(\pi) \cup \emptyset$ (we assume that a customer that has not yet arrived into the system is in state $\emptyset$). The last component T is defined with respect to some transition function t, $t : \Sigma \times E \rightarrow \Sigma$, which given a current state $\sigma \in \Sigma$ and a labeled event $e \in E$ returns a new state σ'. If the event is not feasible for state σ then t returns the empty set. We write T as follows,

$$T = \{(\sigma, e, \sigma') \in \Sigma \times E \times \Sigma | \sigma' = t(\sigma, e) \wedge \sigma` \neq \emptyset\}. \quad (2)$$

For the single-station case the corresponding transition system is demonstrated in Figure 3.

4.3. Step 2: Feature Selection

In the current step, each state of the TS is related to a sequence of features. In essence, it would be possible to enrich every state $\sigma \in \Sigma$ with these features, however that would cause state 'explosion' and the transition system would 'loose' its process qualities. Moreover, we aim at making a distinction between case-specific states (i.e. case type and other case attributes) and system state, such as queue-lengths.

The features that we consider are state-dependent, e.g. for state $(qEntry)$, which corresponds to customer being in a queue, we add queueing parameters, such as queue-length, while for service state, $(qEntry, sStart)$, we add customer class (since service duration is assumed independent of queue-length).

Let $\mathcal{X}$ be the feature universe, e.g. queue-length Q can be considered as one of the features $Q \in \mathcal{X}$ and denote $\mathcal{X}^*$ the finite sequences of features. We define the feature selection function as $f : \Sigma \to \mathcal{X}^*$. In other words, we attach a sequence $X \in \mathcal{X}^*$ to each state $\sigma \in \Sigma$. We refer to the resulting transition-system as the featured transition system (FTS), which can be written as (Σ, E, T, f). The feature function can be either manually defined, or it could be mined by applying feature selection algorithms on the service log.

4.4. Step 3: Annotation

Let Y be the set of outcomes that we wish to learn from a given service log (S, P, α_S), with $P \subseteq \Pi$ being the set of observed service paths ($p \in P$ could be a partial path). We may apply the function π on any prefix of $p \in P$ and get observed sequences of event-labels, $\pi(p)$. Each such sequence can be mapped into a state of the transition system, since $\sigma = \pi(p) \in \Sigma$ by definition. For state σ, we mine the observed values x of the sequence $X = f(\sigma)$. As a result, we have both the observed current TS state σ and the relevant values for features that are attached to that state, x. Eventually, we get a sample of size N (given that there were N relevant event in the log) of the pairs $(\sigma, x_i), \forall \sigma \in \Sigma, i = 1, ..., N$.

For each observed pair (σ, x_i) we also extract the observed outcome (e.g. real value of delay) from the service log. We denote this observed measure $y_i \in Y$, and at the end of the third step, for each state $\sigma \in \Sigma$ we get a sample of pairs $(x_i, y_i) \in \mathcal{X}^* \times Y^*$. For our running example, for every visit of customer in state $(qEntry)$, we observe pairs of feature values (e.g. queue-lengths, waiting times of the most-delayed customer) and the corresponding real delays that occurred in the log.

In the remainder of the paper, we show how to formulate and annotate suitable featured transition systems and learn prediction functions that can be written as $\psi \in \Sigma \times \mathcal{X}^* \to Y$. In other words, the prediction function receives a state and a sequence of feature values; then, an algorithms approximates ψ from past outcomes, and returns a predicted value (which can also be categorical for classification problems.)

5. Delay Prediction for Single-Class Settings

In this section, we focus on the delay prediction problem in the single-class setting, where customers are of uniform types. We propose mining techniques for three classes of delay predictors that implement two different strategies. First, we follow a learning-based analysis that utilizes the extended transition system framework that we presented in Section 4. Our second and third class of predictors, in turn, follow a completely different strategy and are grounded in queueing theory. For each technique, we first define the actual predictor before turning to the queue mining techniques for the construction of the predictor from a service log.

5.1. Transition System Prediction

Our first predictor exploits the feature-annotated transition systems discussed in Section 4. First, we define the outcome that we wish to learn from the service log to be the delay of a customer. Formally, let $Y = \mathbb{N}$ be the space of possible delays and (Σ, E, T, f) the feature-annotated transition system that corresponds to the initial system that we described in Figure 3. As a first baseline predictor, we use the method without any features, i.e. $f(\sigma) = \emptyset, \forall \sigma \in \Sigma$. At the annotation stage we couple the state $(qEntry)$ with the observed delays and thus receive data-based couples $(qEntry, y_i)$ with $i = 1, ..., N$ indexing past N queueing events that appear in the S-Log. As such, this predictor corresponds to the original approach for time prediction as introduced by van der Aalst et al. [4].

As a next step, we add a feature 'queue-length' $Q \in \mathcal{X}$ to the transition system. The state $(qEntry)$ in the feature-annotated transition system is then assigned with both past queue-lengths and past delays. Consequently, for the $(qEntry)$ state, we mine the observed pairs $(q(t), y_i), i = 1, ..., N$ with $q(t)$ being the queue-length at time t, which is the arrival time of the target customer (customer who's time we aim at predicting).

Queue Mining. Given an S-Log, we first construct a predictor for the feature-less transition system. For each recorded delay, attached to $\sigma = (qEntry)$, we calculate the average over past delays, which yields the plain transition system predictor ψ_{PTS}:

$$\psi_{PTS} = \frac{1}{N} \sum_{i=1}^{N} y_i, \tag{3}$$

For the featured transition system, we apply two learning methods that approximate the prediction function given the pairs $(q(t), y_i), i = 1, ..., N$. Specifically, we apply non-linear regression and use regression trees, c.f. [17, Ch. 9] and denote the two prediction functions as ψ_{NLR} and ψ_{TREE}, respectively.

5.2. Queueing Model Predictors

Our second class of predictors does not follow a regression analysis, but is directly grounded in queueing theory. These predictors relate to the $G/M/s + M$ model, so that upon the arrival of a target-customer, there are n homogeneous working agents at the station. We denote the mean service time by m and assume that service duration is exponentially distributed. Therefore, the service rate of an individual agent is $\mu = 1/m$. Impatient customers may leave the queue and customer individual patience is exponentially distributed with mean $1/\theta$, i.e., the individual abandonment rate is θ. Whenever customers do not abandon the system ($\theta = 0$), the model reduces to $G/M/s$.

We define two delay predictors based on the $G/M/s$ and the $G/M/s+M$ models, respectively. We refer to the first predictor as queue-length (based) predictor (QLP) and to the second as queue-length (based) Markovian (abandonments) Predictor (QLMP) [18]. As their names imply, these predictors use the queue length (in front of the target customer) to predict its expected delay. We define the queue-length, $q(t)$, to be a random variable that quantifies the number of cases that are delayed at time t. The QLP for a target customer arriving at time t is:

$$\psi_{QLP}(q(t)) = \frac{(q(t)+1)}{s\mu} \tag{4}$$

with n being the number of agents and μ being the service rate of an individual agent.

The QLMP predictor assumes finite patience and is defined as follows:

$$\psi_{QLMP}(q(t)) = \sum_{i=0}^{q(t)} \frac{1}{s\mu + i\theta}. \tag{5}$$

Intuitively, when a target-customer arrives, it may progress in queue only if customers that are ahead of him enter service (when an agent becomes available, at rate $s\mu$) or abandon (at rate $i\theta$ with i being the number of customers in queue). For the QLP, $\theta = 0$ and thus the QLMP predictor (Eq. 5) reduces to the QLP predictor (4).

Queue Mining. Provided with an S-Log that is up-to-date, at time t, we extract the primitives required for calculating the QLP and QLMP. We start with the queue length $q(t)$ and the number of active servers s:

$$\begin{aligned}\widehat{q(t)} &= |\{(p_1,\ldots,p_m) \in P \mid \epsilon(p_m) = qEntry \ \wedge \\ &\wedge \quad \tau(p_m) \leq t\}|, \\ \hat{s} &= |\{(p_1,\ldots,p_m) \in P | \epsilon(p_m) = sStart\}|.\end{aligned}$$

In other words, the queue length is estimated by the number of paths that have experienced only a $qEntry$ event, while the number of servers online is estimated by the number of customers that are in service at time t. Note that the latter estimator can be inaccurate, when the queue is empty, since it does not account for idle servers.

To obtain μ and θ we first define a predicate $Q(\cdot,\cdot)$ and three relations, namely, R_1, R_2, R_3:

$$\begin{aligned}Q(s_1,s_2) &= \exists p \in P, i \in \mathbb{N}^+ (p_i = s_1 \ \wedge p_{i+1} = s_2), \\ R_1 &= \{(s_1,s_2) \in \mathcal{S} \times \mathcal{S} | \epsilon(s_1) = sStart \wedge \\ &\wedge \quad \epsilon(s_2) = sEnd \ \wedge Q(s_1,s_2)\}, \\ R_2 &= \{(s_1,s_2) \in \mathcal{S} \times \mathcal{S} | \epsilon(s_1) = qEntry \wedge \\ &\wedge \quad (\epsilon(s_2) = sStart \vee \\ &\vee \quad \epsilon(s_2) = qAbandon) \ \wedge Q(s_1,s_2)\}, \\ R_3 &= \{(p_1,\ldots,p_m) \in P | \epsilon(p_m) = qAbandon\}.\end{aligned}$$

The predicate, Q, indicates that two events $s_1, s_2 \in S$ are sequential in the same path. The first relation, R_1, contains pairs of events from the same path that correspond to service start and end activities, respectively. Similarly, R_2 contains pairs of events that are sequential in the same path and indicate waiting in queue (until abandonment or service). Lastly, R_3 contains paths that ended with an abandonment. We use R_1 to estimate the average service time, m, as follows:

$$\hat{m} = \frac{\sum_{(s_1,s_2)\in R_1} (\tau(s_2) - \tau(s_1))}{|R_1|}, \tag{6}$$

and deduce a naïve moment estimator for $\hat{\mu}$, $\hat{\mu} = 1/\hat{m}$ [19]. Lastly, we estimate θ based on a statistical result that relates it to the total number of abandonments and the total delay time for both served and abandoned customers, cf., [20]. Formally,

$$\hat{\theta} = \frac{\sum_{(s_1,s_2)\in R_2} (\tau(s_2) - \tau(s_1))}{|R_3|}. \tag{7}$$

5.3. Snapshot Predictors

The *(heavy-traffic) snapshot principle* [21], p. 187 is a heavy-traffic approximation, which refers to the behavior of a queue model under limits of its parameters, as the workload converges to capacity. In the context of the delay prediction problem, the snapshot principle implies that under the heavy-traffic approximation, delay times (of other customers) tend to change negligibly during the waiting time of a single customer [18]. We define two snapshot predictors: Last-customer-to-Enter-Service (LES or ψ_{LES}) and Head-Of-Line (HOL or ψ_{HOL}). The LES predictor is the delay of the most recent service entrant, while the HOL is the delay of the first customer in line.

In real-life settings, the heavy-traffic approximation is not always plausible and thus the applicability of the snapshot principle predictors should be tested ad-hoc, when working with real data sets. Results of synthetic simulation runs, conducted in [18], show that the LES and HOL are indeed appropriate for predicting delays.

Queue Mining. Given an S-Log that is up-to-date, at time t we mine the snapshot predictors as follows. Assuming the FCFS policy, we estimate HOL as follows:

$$\psi_{HOL} = \min_{s\in R_4} t - \tau(s),$$

where,

$$R_4 = \{s \in S | \epsilon(s) = qEntry \wedge \exists (p_1,\ldots,p_m) \in P(p_m = s)\},$$

which corresponds to customers that are currently waiting. The LES is estimated in two phases. First, we obtain the path that has $sStart$ as the most recent event:

$$\upsilon = \text{argmax}_{(p_1,\ldots,p_m)\in R_5} \tau(p_m),$$

where,

$$R_5 = \{(p_1,...,p_m) \in P | \epsilon(p_m) = sStart\}.$$

υ contains the path of the LES. Here, we assume that events are instantaneous and cannot co-occur (the difference may be in milliseconds). Finally, in order to obtain the LES, we calculate the waiting time for υ:

$$\psi_{LES} = \tau(p_\upsilon) - \tau(p_{\upsilon-1}).$$

6. Queue Mining for Multi-Class Settings

As discussed in Section 2, many single-station systems in real-life scenarios consist of multiple classes of customers. In this section, we present similar families of delay predictors as in the single-class setting, but extend the methods to accommodate for multi-class services.

Among the different customer types, we consider the following priority policy. Let $C = \{c_1, ..., c_k\}$ be the set of k customer classes and let η be the priority function that assigns each c_i a corresponding priority, i.e. $\eta(c_i) \in \{1, ..., k\}$ with 1 being the highest priority and k being the lowest. We assume that the priorities among customers are totally ordered and hence waiting customers of higher priority shall always enter service before lower-priority customers. The policy for handling customers within the same class is FCFS (First-Come-First-Served).

6.1. Transition System Predictors

Starting with predictors based on transition systems, the approach based on feature-annotated transition systems proposed in Section 4 allows for direct integration of multiple classes. First, we enrich the states of the transition system to include customer classes. Therefore, the new event label set, E' is $E' = E \times C$ and the states are defined accordingly. For example, the state $(qEntry)$ turns into k states $(qEntry, c_1), ..., (qEntry, c_k)$. The transition system that results from the first step of our approach is (Σ, E', T).

Second, we attach a vector of queue-lengths to the states $(qEntry, c_k), c_k \in C$ as the feature vectors, instead of using a single queue. Let $\boldsymbol{q}(t) = (q_{c_1}(t), \ldots, q_{c_k}(t))$ be a vector of queue lengths at time t, where $c_1, \ldots, c_k \in C$ are the priorities of the respective customer classes.

Queue Mining. Given a service log $(S, P, \{\tau, \epsilon, \xi\})$ and time t, queue length $q_{c_i}(t)$ is estimated by $\widehat{q_{c_i}(t)}$ as follows:

$$\begin{aligned} \widehat{q_{c_i}(t)} &= |\{(p_1, \ldots, p_m) \in P |\ \epsilon(p_m) = qEntry \wedge \\ &\wedge \ \tau(p_m) \leq t \ \wedge \ \xi(p_m) = c_i\}|. \end{aligned} \tag{8}$$

Once the vector of queue-lengths is mined, we apply the nonlinear regression and regression trees once more, without changing the notation of ψ_{NLR} and ψ_{TREE}.

6.2. Queueing Model Approximation

In this part, we extend the queueing model that we consider in Section 5.2. We assume that the target-customer of type $c \in C$ arrives at time t into an $M/M/s + M$ queueing system with a priority discipline, η, as defined above. Note that, for notational convenience, we consider the customer type in terms of the priority, i.e. given a customer of class c we write the resulting $\eta(c) \in \{1, ..., k\}$ with 1 being the highest priority, instead of c.

We assume that during the stay of the target-customer, the arrival rate of all customer types is a multi-dimensional (and independent among classes) Poisson process with a vector of rates $(\lambda_1, \lambda_2, ..., \lambda_k)$ and an initial vector of busy servers (or customers in service) $(s_1, ..., s_k)$ with $\sum_{i=1}^{k} s_i = s$ and s being the number of servers online. We assume that the number of servers does not change during the customer's waiting time.

Service times are assumed exponential, independent of each other and of the arrival process. Service rates are represented by the vector $(\mu_1, ..., \mu_k)$ that corresponds to the k customer classes. Customer (im)patience is assumed exponential with abandonment rates $(\theta_1, ..., \theta_k)$, corresponding to the various classes. Note that the queue-lengths vector $\boldsymbol{q}(t)$ is available (e.g. mined on-the-fly) with q_j being the queue-length of the customer with *priority* j (not class j). We can think of $\boldsymbol{q}(t)$ as the original vector of queue-lengths, ordered with respect to class priorities.

For the above model, only an approximation via upper and lower bounds is possible due to the uncertainty in the order of service completions. In the evaluation section we shall calculate both bounds and check for their proximity to each other; if these bounds prove to be close enough, then we can deduce that one of them, e.g. the upper bound (for safety) can be used as a proxy for the desired QLP delay predictor. Our technique is somewhat similar to the approximation proposed in Section 4 of [22]. However, we develop bounds for queues with a strict priority discipline, while in [22] the bounds are for first-come first-served queues. We start by approximating the top-priority customers, i.e. customers with priority level of 1. Then, we inductively build upon the results from the top-priority queue to show the upper and lower bounds for general-priority customers.

6.2.1. Top-Priority Customers

We provide an *iterative* algorithm for calculating the upper and lower bounds for the expected delay of top-priority customers, i.e. we consider customers of class c such that $\eta(c) = 1$. Denote by ψ^1_{QLPU} the upper bound and ψ^1_{QLPB} the lower bound for their expected delay. Let n be the n-th iteration of our algorithm, with $n = 1, ..., q_1$ (queue-length of top-priority customers). Algorithm iterations correspond to the process of service completions, i.e. $n = 1$ is the first stage, into which the target-customer arrives, while $n = 3$ means that two customers have completed service since the arrival.

Let $(s^n_1, ..., s^n_k)$ to be the vector of customers in service (for each customer type) during the n-th iteration. For example, $s^1_1 = 3$, can be interpreted as 3 top-priority customers are being served at the first iteration of the algorithm. Let $\mathrm{argmax}(n)$ be the index of the slowest customer class in service at the n-th iteration (there is such customer as long as the queue is nonnegative).

Between consequent iterations the update rule for the vector of customers in service is:

$$\begin{aligned} s^{n+1}_1 &= s^n_1 + 1, \\ s^{n+1}_{\mathrm{argmax}(n)} &= s^n_{\mathrm{argmax}(n)} - 1, \end{aligned}$$

i.e. we assume that at each step of the algorithm the slowest customer finishes service (hence the upper bound). The other elements of the service vector remain the same between iterations.

The idea of the upper bound calculation for top-priority customers is similar to the single-class QLMP predictor:

$$\psi^1_{QLPU} = \sum_{n=1}^{q_1+1} \frac{1}{(q_1 - n + 1)\theta_1 + \sum_{i=1}^{k} s^n_k \mu_k}. \tag{9}$$

The calculation holds due to the fact that top-priority customers are 'aware' only of customers in service (non-preemptive priority) and other top-priority customers.

We shall now provide the lower bound for the delay of the top-priority queue. Let $\mathrm{argmin}(n)$ be the index of the fastest customer class in service at the n-th iteration. Now, the update rule is that fastest customers leave service:

$$\begin{aligned} s_1^{n+1} &= s_1^n + 1, \\ s_{\mathrm{argmin}(n)}^{n+1} &= s_{\mathrm{argmin}(n)}^n - 1, \end{aligned} \tag{10}$$

and then we can write the lower bound on the delay predictor ψ_{QLPB}^1 as follows:

$$\psi_{QLPB}^1 = \sum_{n=1}^{q_1+1} \frac{1}{(q_1 - n + 1)\theta_1 + \sum_{i=1}^k s_k^n \mu_k}. \tag{11}$$

Note that the difference between the bounds in the top-priority case is only the update rule of the iterative algorithm. For a general class, it will not be the case, since the upper bound will also be influenced by higher-priority customers.

6.2.2. General-Priority Customers

For a target customer of a general priority c, the approximation depends on the resulting bounds for all higher priorities. A target customer of type c 'sees' only queues of classes from the higher-or-equal priority set, $H(c) = \{j | \eta(j) \leq \eta(c)\}$, as well as the vector of customers that are already in service, $(s_1, ..., s_k)$ (we assume a non-preemptive policy). The order of service entries is the following. First, all queues that have higher priorities must enter service. Then, all customers of higher priority that arrived while customer type c was waiting enter service (in case they did not abandon). Lastly, all customers of the same priority as c that were ahead in the queue must enter service and only then the target-customer will enter service.

Given that the arrival processes of the various customer types are assumed Poisson with rates $(\lambda_1, \lambda_2, ..., \lambda_k)$, we can approximate the amount of work that higher-priority customers bring during the waiting time of the target customer. Formally, let O^c be the amount of work units that 'overtaking' customers bring into the system while target customer of type c waits. Then we can write

$$O^c = \sum_{i=1}^{\eta(c)-1} \lambda_i \times W^c \times (1/\mu_i), \tag{12}$$

with W^c being the real waiting time of the target-customer (of class c).

For the n-th iteration, we assume that we have the service vector (number of active servers that serve the various customer types), given by $(s_1^n, ..., s_k^n)$ and as before we use the $\mathrm{argmin}(n)$ and $\mathrm{argmax}(n)$ notation. Note that the service vector is already updated to the point in time for which higher-priority customers from the set $H(c)$ have left service and only similar priority customers remain in queue. This implies that even when we calculate the delay for a c type customer, we need to run the algorithm for all higher priority classes. The update between iterations for both upper and lower bound is similar to the top-priority case, as demonstrated in Equations (9) and (10). Let $j = \eta(c)$ and denote $\rho_i = \lambda_i/\mu_i$. Then we write:

$$\begin{aligned} \psi_{QLPU}^j &= \sum_{m=1}^{j-1} \psi_{QLPU}^m + \\ &+ \sum_{n=1}^{q_j+1} \frac{1}{(q_j - n + 1)\theta_j + \sum_{i=1}^k s_k^n \mu_k} + \widehat{O^c}, \end{aligned}$$

with $\widehat{O^c}$ being the estimator of O^c and given as follows:

$$\begin{aligned} \widehat{O^c} &= \sum_{i=1}^{\eta(c)-1} \lambda_i \times \psi_{QLPU}^j \times (1/\mu_i), \\ &= \psi_{QLPU}^j \sum_{i=1}^{\eta(c)-1} \rho_i. \end{aligned}$$

Therefore,

$$\psi_{QLPU}^j = \sum_{m=1}^{j-1} \psi_{QLPU}^m + \frac{\sum_{n=1}^{q_j+1} \frac{1}{(q_j - n + 1)\theta_j + \sum_{i=1}^k s_k^n \mu_k}}{1 - \sum_{i=1}^{\eta(c)-1} \rho_i}. \tag{13}$$

Note that ρ_i can be interpreted as the workload that a customer of priority i brings into the system.

For the lower bound, we do not consider overtaking and we assume that the fastest service is always completed, therefore:

$$\psi_{QLPB}^j = \sum_{n=1}^{q_j+1} \frac{1}{(q_j - n + 1)\theta_j + \sum_{i=1}^k s_k^n \mu_k}. \tag{14}$$

6.2.3. Mining Queueing Parameters

In order to complement the algorithms that we proposed in this part, we need to mine several queueing parameters from the service log. The queue-lengths vector was already presented at the beginning of the section. What remains to be extracted from the log is patience rates, service rates, number of customers in service (for each customer type) and the arrival rates. The first three components, θ_j, μ_j, s_j are trivial extensions of over the mining methods proposed in the previous sections. To each of the methods, we add the predicate $\wedge \xi(p_1) = c$ in order to differentiate customer types. For example, we estimate the number of top-priority customers in service as,

$$\widehat{s_1} = \left| \{(p_1, \ldots, p_m) \in P | \epsilon(p_m) = sStart \wedge \xi(p_1) = 1\} \right|. \tag{15}$$

For the arrival rates vector, we consider fixed time intervals during which we assume that the arrival rates are constant. For service processes in call centers as described in Section 2 and considered in our evaluation, for instance, a time interval of 15 minutes is appropriate. Let $[t_1, t_2)$ be a time-window for which we aim at calculating the arrival rate λ_j, then given a service log $(S, P, \{\tau, \epsilon, \xi\})$,

$$\begin{aligned} \widehat{\lambda_j} &= |\{(p_1, \ldots, p_m) \in P | \epsilon(p_m) = qEntry \ \wedge \\ &\wedge \ \ \xi(p_1) = c \ \wedge \ \tau(p_1) < t_2 \wedge \ \tau(p_1) \geq t_1\}|. \end{aligned} \tag{16}$$

Then, as the target customer arrives at time t we find a time-window $[t_l, t_v)$ such that $t \in [t_l, t_v)$ and use the arrival rate from that time-window for our calculations.

6.3. Multi-Class Snapshot Predictors

The last strategy for prediction is based on the snapshot principle, extended for multi-class settings. Previously, we have shown that the Last-customer-to-Enter-Service (LES) and Head-Of-Line (HOL) predictors yield very similar results [7], so that we focus on the HOL predictor in this section.

Instead of predicting the delay by providing the waiting time of the head-of-line (HOL) among all classes, we use a per-class HOL predictor. In other words, we mine a vector of head-of-line customers as follows: denote by $\boldsymbol{h}(t) = (h_{c_1}(t), \ldots, h_{c_k}(t))$ the vector of the longest waiting customers in each queue (the delays of the HOL), with $c_1, \ldots, c_k \in C$ as before. For a service log $(S, P, \{\tau, \epsilon, \xi\})$ and time t, this vector can estimated from the log as:

$$\widehat{h_{c_i}(t)} = \min_{s \in \{s \in S \mid \epsilon(s)=qEntry \wedge \xi(s)=c_i \wedge \exists\ (p_1,\ldots,s) \in P\}} t - \tau(s). \tag{17}$$

The HOL predictor for class $c_i \in C$ is then defined as $\psi^{c_i}_{HOL}(t) = h_{c_i}(t)$.

7. Evaluation

The current section presents an empirical evaluation of the delay predictors, for both single-class and multi-class scenarios. The evaluation shows that for data that comes from a single-class service station, the best predictors are the snapshot predictors. As expected, when applying the single-class predictors to a service log that represents a multi-class service process these predictors are less accurate and require an adjustment. When multi-class predictors are applied, accuracy is improved and both snapshot predictors and transition system methods (e.g. regression trees applied to queue-length vectors) are shown to have good predictive power.

Below, we first describe two real-world service logs that we used for our experiments (Section 7.1). Then, we describe the experimental setup (Section 7.2), and report on the main results in Sections 7.3 and 7.4 for the single-class and multi-class settings, respectively. Finally, we discuss the results in Section 7.5.

7.1. Data Description

The data for our experiments stems from two call centers: (1) a call center of an Israeli bank and (2) a call center of an Israeli telecommunication company. The data is gathered and stored in the Technion laboratory for Service Enterprise Engineering (SEELab)[1]. The experiments on the first call center correspond to the single-class scenario, since we have focused on a single type of customers. For the second call center, three customer types that represent the private sector are considered: VIP, Regular and Low priority (see [10] for further description of the dataset and the priority setting). We shall now provide a brief overview of the two datasets.

[1] `http://ie.technion.ac.il/Labs/Serveng`

Israeli Bank's Call Center. The dataset contains a detailed description of all operational transactions that occurred within the call center, between January 24th, 2010 and March 31st, 2011. The log contains, for an average weekday, data on approximately 7000 calls. For our delay prediction experiments, we selected three months of data: January 2011-March 2011 (a service log of 879591 records). This amount of data enables us to gain useful insights into the prediction problem, while easing the computational complexity (as opposed to analyzing the entire data set). The three months were selected since they are free of Israeli holidays. In our experiments, we focused only on cases that demanded 'general banking' services, which is the majority of calls arriving into the call center (89%). This case selection is appropriate, since our single-class queueing models assume that customers are homogeneous.

Israeli Telecommunication Company Call Center. The call center processes up to 50,000 service requests a day, routes requests according to various resource skills, and simultaneously queues requests across multiple sites. The center is operated with around 600-800 agent positions on weekdays and 200-400 agent positions on weekends. Further, several types of services are provided; the most common are Private, Business, Technical and Content Internet. In this paper, we focus on the Private service, which handles requests with low, regular and VIP priorities. For our evaluation, we selected three months of data to serve as our service logs, from January 1, 2008 to March 31, 2008.

We divided both service logs into two subsets: a training log and a test log. This is common practice when performing statistical model assessment [17]. We addressed each delayed customer in the test logs as the target-customer, for whom we aim to solve the delay prediction problem.

7.2. Experimental Setup

The controlled variable in our experiments is the *prediction method* (or the delay predictor). The various methods that we defined in Sections 5 and 6 are used. Further, the definition of the online delay prediction problem given in Section 2 refers to the root mean-squared error (RMSE) in order to assess the performance of a certain predictor. A data-driven approximation of the RMSE is the root of the average-squared error, RASE. It serves as the uncontrolled variable in our experiment and is defined as follows:

$$RASE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(d_i - p_i)^2}, \tag{18}$$

with $i = 1, ..., N$ being the i-th test-log delay out of N delays, d_i the real duration of the i-th delay and p_i the corresponding predicted delay. The RASE is a proxy to the difference between the real waiting time W and the predictor ψ and, thus, the lower the value the better the prediction.

7.3. Results: Single-Class Setting

Figure 4 summarizes the results that we received with the presented predictors for the single-class setting, i.e., the plain

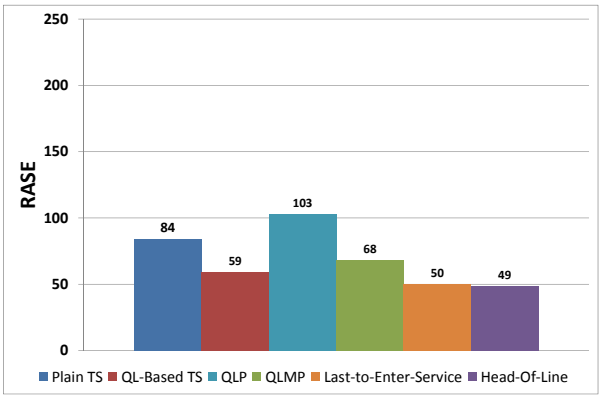


Figure 4: Prediction error for the first dataset: single-class techniques

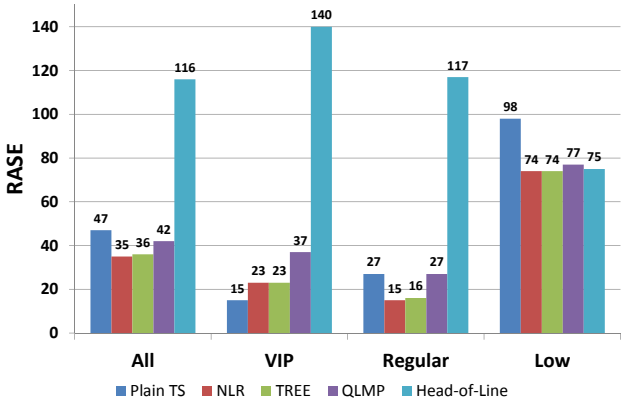


Figure 5: Prediction error for the multi-class dataset: single-class techniques

time prediction using a transition system (Plain TS), the feature-annotated version (QL-based TS) that incorporates the queue length, the queue length predictor without and with Markovian abandonments (QLP and QLMP), and the Last-Customer-To-Enter-Service and Head-Of-Line snapshot predictors.

For the methods based on transition systems, we consider past delays of customers with similar path-history, when predicting the delay of the target-customer. The problem with the Plain TS method, however, is that, when applied to our real-life process, it considers all past delays. Considering all past delays is appropriate in steady-state analysis, *i.e.*, when the relation between demand and capacity does not vary greatly over time. Transition system method that considered the queue-length performed significantly better, since it captures system load. The performance of this method was second best only to snapshot predictors.

The queueing model predictors consider the time-varying behavior of the system and attempt to quantify the system-state based on the number of delayed cases. The QLP fails in *accuracy*, since it assumes that customers have infinite patience, which is seldom the case in call center processes. We presume that the QLP would perform better for processes with negligible abandonment rates such as healthcare scenarios where customers typically have more patience.

On the other hand, the QLMP outperforms the Plain TS method. Therefore, accounting for customer (im)patience is indeed relevant in the context of call centers, and other processes in which abandonments occur [23]. In contrast, the QLMP is inferior when compared to snapshot predictors or the queue-length transition system predictor. This phenomena can be explained by deviations between model assumptions and reality.

Throughout our experiments, snapshot predictors have shown the largest improvement in accuracy (of up to 40%) over the rest. Thus, we conclude that for the considered queueing process (of a call center), an adequate delay prediction for a newly enqueued customer would be the delay of the current Head-Of-Line (HOL) or the delay of the Last-Customer-To-Enter-Service (LES). Our main insight is that in time-varying systems, such as call center, one must consider only recent delay history when inferring on newly arriving cases.

7.4. Results: Multi-Class Setting

We first present the results of operating single-class predictors on the dataset that actually features multiple customer classes. Figure 5 indicates that the previously superior snapshot predictors deteriorate in accuracy (we only depict the Head-Of-Line since the Last-Customer-To-Enter-Service predictor yielded equivalent results). This is especially true for the higher-priority customers, since the single-class method does not distinguish between Head-Of-Line delays of Low, Regular and VIP customers. We observe that across the three classes, the plain TS method works best for VIP customers, indicating that VIP delays are predictable and that the system from their viewpoint is in fact in steady state. However, for other classes, both the non-linear regression (NLR) and the regression tree (TREE) methods prevail across all customer types. For the low-priority class, the snapshot predictor is comparable to other predictors, because these customers experience a large dependency on recent events.

Figure 6 describes the prediction error when applying multi-class predictors from Section 6 on the multi-class dataset. Here, we observe that upper and lower bounds of the queueing approximation perform similarly across classes. Furthermore, we notice that after the adjustment to multi-class, the Head-Of-Line predictor is again superior to all methods across scenarios, except the regression tree method. Note that regression tree method performs especially well for the most difficult class to predict, which is the low-priority class (it 'suffers' from largest variation and dependencies on high-priority classes). Moreover, unlike the case in the single-class dataset, queueing model approximation predictors are comparable to the Head-Of-Line predictor.

To conclude the overview of the results, we compare the overall improvement of adjusting single-class methods to the multi-class scenario. Figure 7 compares single-class and multi-class methods for all customers and essentially merges the information that we gain from Figures 5 and 6. We observe a lower prediction error for all methods, when accounting for the existence of several customer classes.

7.5. Discussion

Below, we provide a two-part discussion of the results. In the first part, we discuss the difference in foundation between

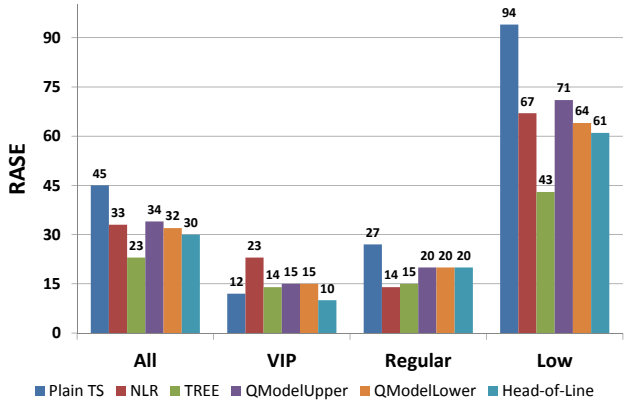


Figure 6: Prediction error for the multi-class dataset: multi-class techniques

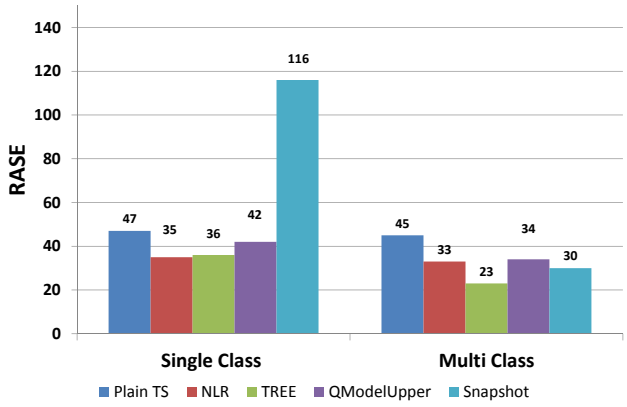


Figure 7: Prediction error for the multi-class dataset: single-class vs. multi-class techniques

the two families of predictive methods: data mining techniques that are based on the transition system and queueing predictors that are based on models, approximations and congestion laws. In the second part, we focus on the effort and consequences of analyzing a multi-class system.

Data Mining Methods vs. Queueing Methods. The single class analysis shows a slight superiority of the queueing methods (Figure 4). Indeed, both the difference in the predictive power and the complexity of the snapshot predictors seems appealing for answering operational questions such as 'how long will this customer wait?'. However, when moving to a dataset with new characteristics, namely several types of customers, the snapshot predictors turn out to lack robustness. Other methods, such as transition system techniques show reasonable results, despite their ignorance of the multiple classes.

This difference emphasizes the strong dependence of techniques that are based on specific models, such as queueing models and their approximations. The strength of model-based predictors is in their conceptual validity, i.e. how well do the assumptions fit reality. On the other hand, data mining techniques do not provide with deep insights on the reality, besides the accuracy of prediction. However, these black-box techniques are extremely robust when shifting among various datasets and scenarios. Once we have adjusted the snapshot predictor and its assumptions to the new reality, its performance became second best only to regression trees.

Queueing approximations via upper and lower bounds, as demonstrated in Section 6 turn out to be useful, especially due to their proximity to each other. This implies that one of these predictors can be used in order to predict delays. Note that the lower bound predictor is comparable to the Head-Of-Line predictor, which supports the validity of the assumptions made in the queueing model. In other words, we conclude that the queueing model parameters are suitable and indeed: service times are exponential, arrivals are Poisson (with different rates in each time-window), and (im)patience is exponentially distributed.

Single-Class vs. Multi-Class. In most services nowadays, customers are divided into classes in correspondence to, e.g., sophisticated Customer Relationship Management (CRM) tools. This could either be as function of the financial value that a customer brings into the company or according to a patient's current health status. Therefore, prediction methods must accommodate for these multi-class services. This point is strengthened by the results of our experiments with the multi-class dataset. Moreover, the accuracy of the predictors depends on both case-dependent characteristics (e.g. VIP customers have longer service times) and system-dependent characteristics that are unique to each class (e.g. customer or resource scheduling protocols). The first type can be mined via the 'case' perspective in process mining, while the second type can be inferred by applying different queue mining techniques. For example, in [10], resource-scheduling protocols are learned from data and can later be used for delay prediction or simulations of the service process.

Analyzing the classes separately can provide insights into the service process. For example, from Figure 6 we learn that the performance of the VIP system is stable, whereas low priority customers experience changes in load and thus in delay duration.

8. Related Work

Our work mainly relates to three streams of work, i.e., process mining in general, time prediction based on mined models in particular, and delay prediction in queueing theory.

Process Mining. Lately, process mining has seen a remarkable uptake, providing tools for the creation of process models from event data, over the assessment of the conformance of models and events, to extensions of models for operational support, see [3] for a recent overview. Despite the wide-spread focus on the control flow perspective, process mining techniques would benefit from additional information, such as time and resource utilization. In particular, several approaches addressed the problem of predicting process completion times for running cases. Van der Aalst et al. [24] highlight the importance of capturing resource utilization appropriately and provide techniques for mining a simulation model. The approach creates a Coloured Petri net that comprises resource and timing information and serves as the basis for time prediction. Rogge-Solti and Weske [11] follow a similar approach, but ground the analysis in a probabilistic

model, formalized as a stochastic Petri net. Then, Monte-Carlo simulation allows for predicting completion time.

Time Prediction. A generic framework for time prediction based on state transition systems constructed from process logs was developed in [4]. Furthermore, state transition annotation-based predictors haven been combined with decision trees, thus taking into account context features such as queue-length, resource availability, and customer characteristics [16]. A continuation of this line of research can be found in [25, 26], where the methods from [16] are extended and applied to low-level event logs, respectively. A general discussion on mining context from event logs can be found in [27].

The work by Folino et al. [16] is close to our model of feature-annotated transition systems as it defines states to comprise of two types of features: (1) 'internal' case properties and (2) 'external' factors that characterize system state. However, this approach has several shortcomings. First, cases are clustered over the two feature types *and* the targeted outcome, which results in an artificial (method-dependent) partition of the joint feature-outcome space. As a consequence, fine-grained details of the feature space could be lost, while non-existent values of the outcome space could be added. Further, the learning method is limited to decision trees, partially due to the discrete nature of the resulting clustering of the feature space, so that other statistical learning methods cannot be applied to the full extent [16].

In this work, we undertake a more flexible approach that enables the use of various types of learning techniques. Further, our approach combines transition systems that comprise of process traces and case-specific attributes with continuous vectors of system-state factors to achieve decoupling of the state (which remains simple and case-related) from the complex (and possibly continuous) feature vectors when applying learning techniques.

Queueing Theory. Predicting queueing delays has been a popular research subject in queueing theory; see [12] for an overview. Statistical techniques for estimating delays were applied mainly for systems in steady-state [28, 20]. Real-time delay predictors that do not assume steady-state, in analogy to the online delay prediction problem addressed in this work, were proposed in [22, 18]. We use these predictors as a basis to our queue mining techniques and address the derivation of these predictors from event data.

9. Conclusion

In this paper, we showed how to consider a queueing perspective in operational process mining for service processes. In particular, we state the problem of *online delay prediction* and provide different techniques, jointly referred to as *queue mining*, that take recorded event data as input and derive predictors for the delay of a case caused by queueing. We addressed delay prediction for the single-class setting that assumes homogeneous customers as well as the multi-class setting that features different classes of customers.

First, we considered mining of regression-based predictors that are based on state transition systems, for which queueing information has been integrated. To this end, we took up existing methods and presented feature-annotated transition systems that separates case state and system state characteristics. This enables us to consider various features of different sizes, without expanding the state space of the transition system and allows for direct application of various machine learning techniques. We further argued for predictors that are grounded in queueing theory and presented mining techniques for predictors that emerge from a queueing model, either based on queueing theory or the snapshot principle.

For all predictors, we tested accuracy using real-life service logs from the telecommunications and financial sectors. Our experiments show that predictors incorporating queueing information or directly grounded in queueing models improve accuracy by 30%-40% on average compared to the plain regression-based method. Also, we observed that the multi-class methods are superior to single-class methods in their predictive power. Regression trees that build upon the extended transition system method provided with the most accurate predictions, improving over the snapshot predictors by almost 25%.

In future work, we intend to expand queue mining to that stem from complex service processes with several stations, *i.e.*, process activities that comprise of resource delays. The natural models, when considering such processes, are *queueing networks*. These models are often mathematically intractable and thus the analysis of queueing networks resorts to simulation or approximation methods in the spirit of the *snapshot principle*.

References

[1] W. M. P. van der Aalst, T. Weijters, L. Maruster, Workflow mining: Discovering process models from event logs, IEEE Trans. Knowl. Data Eng. 16 (9) (2004) 1128–1142.

[2] W. M. Van Der Aalst, Workflow verification: Finding control-flow errors using petri-net-based techniques, in: Business Process Management, Springer, 2000, pp. 161–183.

[3] W. van der Aalst, Process Mining: Discovery, Conformance and Enhancement of Business Processes, Springer, 2011.

[4] W. M. van der Aalst, M. Schonenberg, M. Song, Time prediction based on process mining, Information Systems 36 (2) (2011) 450–475.

[5] R. W. Hall, Queueing Methods: For Services and Manufacturing, Prentice Hall, Englewood Cliffs NJ, 1991.

[6] G. Bolch, S. Greiner, H. de Meer, K. S. Trivedi, Queueing networks and Markov chains - modeling and performance evaluation with computer science applications; 2nd Edition, Wiley, 2006.

[7] A. Senderovich, M. Weidlich, A. Gal, A. Mandelbaum, Queue mining - predicting delays in service processes, in: M. Jarke, J. Mylopoulos, C. Quix, C. Rolland, Y. Manolopoulos, H. Mouratidis, J. Horkoff (Eds.), Advanced Information Systems Engineering - 26th International Conference, CAiSE 2014, Thessaloniki, Greece, June 16-20, 2014. Proceedings, Vol. 8484 of Lecture Notes in Computer Science, Springer, 2014, pp. 42–57.

[8] BPMN 2.0, Object Management Group: Needham, MA 2494 (2011) 34.

[9] D. G. Kendall, Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded markov chain, The Annals of Mathematical Statistics 24 (3) (1953) pp. 338–354.

[10] A. Senderovich, M. Weidlich, A. Gal, A. Mandelbaum, Mining resource scheduling protocols, in: S. W. Sadiq, P. Soffer, H. Völzer (Eds.), Business Process Management - 12th International Conference, BPM 2014, Haifa, Israel, September 7-11, 2014. Proceedings, Vol. 8659 of Lecture Notes in Computer Science, Springer, 2014, pp. 200–216.

[11] A. Rogge-Solti, M. Weske, Prediction of remaining service execution time using stochastic petri nets with arbitrary firing delays, in ICSOC. Proceedings. Vol. 8274 of Lecture Notes in Computer Science., Springer (2013) 389–403

[12] E. Nakibly, Predicting waiting times in telephone service systems, Master's thesis, Technion–Israel Institute of Technology (2002).

[13] M. B. Houston, L. A. Bettencourt, S. Wenger, The relationship between waiting in a service queue and evaluations of service quality: A field theory perspective, Psychology and Marketing 15 (8) (1998) 735–753.

[14] Z. Carmon, D. Kahneman, The experienced utility of queuing: real time affect and retrospective evaluations of simulated queues, Tech. rep., Working paper, Duke University (1996).

[15] R. C. Larson, Perspectives on queues: Social justice and the psychology of queueing, Operations Research 35 (6) (1987) 895–905.

[16] F. Folino, M. Guarascio, L. Pontieri, Discovering context-aware models for predicting business process performances, in: R. Meersman, H. Panetto, T. S. Dillon, S. Rinderle-Ma, P. Dadam, X. Zhou, S. Pearson, A. Ferscha, S. Bergamaschi, I. F. Cruz (Eds.), On the Move to Meaningful Internet Systems: OTM 2012, Confederated International Conferences: CoopIS, DOA-SVI, and ODBASE 2012, Rome, Italy, September 10-14, 2012. Proceedings, Part I, Vol. 7565 of Lecture Notes in Computer Science, Springer, 2012, pp. 287–304.

[17] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning, Springer Series in Statistics, Springer New York Inc., New York, NY, USA, 2001.

[18] R. Ibrahim, W. Whitt, Real-time delay estimation based on delay history, Manufacturing and Service Operations Management 11 (3) (2009) 397–415.

[19] L. Schruben, R. Kulkarni, Some consequences of estimating parameters for the m/m/1 queue, Operations Research Letters 1 (2) (1982) 75 – 78.

[20] L. Brown, N. Gans, A. Mandelbaum, A. Sakov, H. Shen, S. Zeltyn, L. Zhao, Statistical analysis of a telephone call center, Journal of the American Statistical Association 100 (469) (2005) 36–50.

[21] W. Whitt, Stochastic-process limits: an introduction to stochastic-process limits and their application to queues, Springer, 2002.

[22] W. Whitt, Predicting queueing delays, Management Science 45 (6) (1999) 870–888.

[23] N. Gans, G. Koole, A. Mandelbaum, Telephone call centers: Tutorial, review, and research prospects, Manufacturing & Service Operations Management 5 (2) (2003) 79–141.

[24] W. van der Aalst, J. Nakatumba, A. Rozinat, N. Russell, Business process simulation: How to get it right, BPM Center Report BPM-08-07, BPMcenter. org.

[25] F. Folino, M. Guarascio, L. Pontieri, Discovering high-level performance models for ticket resolution processes, in: R. Meersman, H. Panetto, T. S. Dillon, J. Eder, Z. Bellahsene, N. Ritter, P. D. Leenheer, D. Dou (Eds.), On the Move to Meaningful Internet Systems: OTM 2013 Conferences - Confederated International Conferences: CoopIS, DOA-Trusted Cloud, and ODBASE 2013, Graz, Austria, September 9-13, 2013. Proceedings, Vol. 8185 of Lecture Notes in Computer Science, Springer, 2013, pp. 275–282.

[26] F. Folino, M. Guarascio, L. Pontieri, Mining predictive process models out of low-level multidimensional logs, in: M. Jarke, J. Mylopoulos, C. Quix, C. Rolland, Y. Manolopoulos, H. Mouratidis, J. Horkoff (Eds.), Advanced Information Systems Engineering - 26th International Conference, CAiSE 2014, Thessaloniki, Greece, June 16-20, 2014. Proceedings, Vol. 8484 of Lecture Notes in Computer Science, Springer, 2014, pp. 533–547.

[27] W. M. P. Van der Aalst, S. Dustdar, Process mining put into context, Internet Computing, IEEE 16 (1) (2012) 82–86.

[28] C. M. Woodside, D. A. Stanford, B. Pagurek, Optimal prediction of queue lengths and delays in gi/m/m multiserver queues, Operations Research 32 (4) (1984) pp. 809–817.